Transputer Instrumentation
for
Particle Flow Measurements.

E. Mills.

This thesis is submitted to the Council for National Academic Awards as part fulfilment for the qualification of Doctor of Philosophy.

Department of Electrical and Electronic Engineering.
Trent Polytechnic Nottingham
Burton Street
Nottingham.

Collaborating Establishment:

Central Electricity Generating Board, CERL
Kelvin Avenue
Leatherhead
Surrey.

September 1989.

I would like to dedicate this work to my grandfather

Walter Leigh Barnett, BSc, MA, FRIC

who never saw the success of his four grandsons.

Transputer Instrumentation
for
Particle Flow Measurements.

E. Mills.

Abstract.

This project concerns a **pulsed charge injection** technique for particle flow measurements in 14 mm diameter pipes. A parallel processing system containing 6 transputers was designed and built in order to study this technique.

Transputers offer significant advantages as single chip microprocessors, including the hardware logic for internal parallel processing as well as 4 high speed communication links for multiprocessor parallel processing environments.

This thesis covers work from 68000 multiprocessor designs to the final instrument incorporating 6 transputers. Pictorially the transputer system can be visualised as an octahedral structure composed of a data acquisition unit connected to a processing array of 4 transputers which in turn is connect to a controlling transputer.

The data acquisition unit consists of 8 analogue input channels which are multiplexed into an adaptive quantization unit producing an 8-bit answer and gain in 3 μs/channel. This application uses only four of these channels and the data from them is distributed by the acquisition unit to four separate transputers for signal processing. The results from these processors are returned to a final transputer which controls the entire system.

This system offers a powerful, (60 MIPS), compact, cost-effective solution giving simultaneous data capture and processing. The use of this system has led to the generation of an empirical model for the measurement of mass flow rates in the range of 0.2 gs^{-1} to 6 gs^{-1}, with a wide range of velocities, from 3.5 ms^{-1} to 14.0 ms^{-1}, giving an accuracy of ±15% of full scale in each of three sub-regions.

Analysis of the results obtained show the distinct possibility of being able to expand the system into the realm of particle size distribution determination. With further work it is felt that this and a closed loop paint spraying/control system could be achieved.

Acknowledgments.

I would like to thank my supervisor, Dr B.C.O'Neill, for his unfailing enthusiasm and support throughout this project. Also thanks to Dr J.Horrocks of CERL, for the interest shown in this work and Dr L.Haydock.

I would also like to thank the technical support staff in the Department of Electrical and Electronic Engineering for their help and to my fellow researchers for their friendship and support.

Special thanks to my family for their support and understanding throughout this research, and to my parents for also reading the thesis during its preparation.

This work has been funded by SERC with further support from the SERC/DTI Transputer Initiative.

TABLE OF CONTENTS

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

LIST OF FIGURES

LIST OF FIGURES [continued]

LIST OF FIGURES [continued]

LIST OF TABLES

1. Introduction.

Many applications in industry could benefit from an accurate particle mass flow measuring instrument. There are currently numerous techniques which claim to offer a solution to this problem, but as with most things, there are some drawbacks in these techniques and also certain situations which can make them inappropriate.

Optically based systems for the application of particle mass flow analysis all have the problem of keeping a clean window through which examination of the flow may take place. Other techniques for mass flow measurement can only give an indication of the actual flow rate due to their primary assumptions. Techniques such as Doppler fall into this category. With Doppler techniques the assumption made is that the powder is transferred in uniform packets, and thus by knowing the speed of these packets, the flow rate may be determined.

Techniques such as capacitance noise monitor the fluctuation of noise in the powder flow and in many cases this noise is not soley related to the flow rate. Some techniques actually impede the flow of the particulate or require the diversion of the flow for a short period, or maybe a continuous diversion of part of the flow. The simplest form of this being to divert the flow into a balance for a given period so that the change in the balance reading over a given time gives the actual flow rate.

What follows is a look at how two industries can obtain benefits from a mass flow measurement instrument. This is followed by a brief explanation of how this research sets out to tackle this flow rate measurement problem.

1.1 Applications of a Particle Mass Flow Rate Meter.

1.1.1 Electrostatic Powder Coating Industry.

Electrostatic paint spraying techniques produce articles which have a uniform coating of paint. To coat a metallic object it is first connected to ground potential. Epoxy paint, in the form of a powder, is pneumatically transported to an electrostatic paint spray gun which charges the paint particles. The particles are attracted to the workpiece and form a dry coating on the surface. Under some circumstances the thickness of the coating is controlled by a **self–limiting** process of electrostatic repulsion but for most industrial applications the thickness is controlled by the density of the spray and the length of time of its application.

Another unusual property of the electrostatic paint process is that particles which would normally pass by the object being sprayed, are actually attracted to the reverse of the object by the electrostatic field. This means that given ideal conditions an object such as a bicycle frame can be entirely coated without the need of turning the frame to point towards the spray head during the process, as in normal wet paint spraying technique.

After the objects have been coated with the powder they are heated in a furnace for approximately 20 minutes so that the epoxy resin melts and forms a good bond to the metal object. Once this bond has been established, the final coating is very resilient. Partially coated objects become hard and expensive to recover, due to the resilience of the coating.

Currently in the powder coating industry there is no method of directly controlling the powder flow rate of the epoxy paint used in electrostatic powder coating process. Instead the flow rate is controlled indirectly by normal adjustment of the air flow in the pipe. Typically, in a real situation, the flow rate is set-up at the initial testing stage of a spray run and is changed only when problems occur. Controlling the flow rate in this manner can give rise to articles not being coated with powder, or just being partially coated.

Problems of lack of coating are normally detected when the objects are removed from the conveyor after they have passed through the furnace some 20 minutes after spraying has occurred.

Another drawback of this lack of control is that partial blockages, formed by the build up of powder within the spray unit can occur. To reduce the effect of these the operator sets the initial spraying level high and consequently under normal conditions the amount of powder sprayed is greater than that required to coat the object. This lack of direct control reduces the efficiency of the coating process in the automatic plant.

1.1.2 Power Generation Industry.

The power generation industry has several applications for this type of instrument. One application is for the determination of the efficiency of their electrostatic precipitators. At present the efficiency of such installations is measured by skilled workers taking air samples at various stages within the precipitator itself. The accuracy of the final result is dependant upon the accuracy of each measurement and the basic experience of the operator. An automatic measuring system which moves through the precipitator measuring the cross-sectional efficiency of the unit would be of great interest to the generating board and precipitator users alike.

Another application is for analysis of exhaust gases in power station chimney stacks. The instrument would be used to measure how clean the exhaust gases were before they entered an exhaust gas driven turbine. The proposed turbine would be working in the exhaust of coal burning units. The gases would be scrubbed and passed to the turbine. With close turbine tolerances any particles passing through the cleaning stage could cause considerable damage to the turbine itself. The instrument could detect the presence of the particles and could be used to shutdown the turbine before damage is caused.

1.2 Aims of the Project.

The basis for the research is the preliminary work carried out by Dr B.C.O'Neill and C.A.Willis [1,2] on the pulse charge injection technique for the determination of mass flow rate measurement. The pulse charge injection technique operates by injecting charge into the flowing particle stream and then detecting the amount of charge transported by the particles to downstream sensors. It was shown that there is a relationship between the charge injected into the flowing particle stream, that transported by particles and the mass flow rate of the particulate.

Introduction.

The purpose of this work is to:

A. Create an instrument to continuously monitor the mass flow rate of particles flowing in a gas without interruption of, or impedance to, the actual flow of the particulate.

B. To extend the applications of such an instrument by the examination of flow rates not previously explored by O'Neill and Willis.

   This instrument would be used in multi–point sampling of power station ducts containing smoke or solids in suspension.

C. To enhance the signal processing of the instrumentation created in order to create an accurate reading of the flow rate.

D. Expand the instrument, allowing it to be used at high temperatures and pressures.

E. Extend the technique into the realm of particle size determination.

A successful outcome of this research program will help to improve the efficiency of the automatic electrostatic spraying plants by reducing the costs of recycling both coated products and the powder itself. The final instrument can also be used in the protection of exhaust gas turbines, and measuring the efficiency of electrostatic precipitators.

With the turbine protection application the density of particles would be lower than that previously explored by O'Neill and Willis, and the pressure and temperature of operation would be much higher. The testing of such an instrument would be carried out in collaboration with CERL.

1.3 Layout of Thesis.

The thesis contains a general review of mass flow rate measurement techniques, chapter 2, followed by the history of the project before the onset of the current research program, chapter 3.

Chapter 4 contains a look at the development design path of the research, which covers the whole of the project's design phase. Greater details of various items within the design follow in chapters 5 and 6.

Next, chapter 7, is a description of the chosen system with operating details of the various processing elements followed by review of the software developed in order to sustain the research, chapter 8.

Chapter 9 contains details of the results obtained with conclusions and suggestions for future developments following in chapter 10.

Each section is written in such a manner that the ideas behind it are self–contained and additional relevant information may be found in the appendices after the main body of the thesis.

2. Review of Mass Flow Rate Techniques.

There are many different ways of measuring the mass flow of powders in pipes, but none are as well established commercially as those in use for single phase flow measurements of liquids. There are many different publications on techniques of measuring the mass flow rate of powders. The major techniques are reviewed in this chapter and cover a wide range from optical to impact, capacitance noise to nuclear magnetic resonance. This project is concerned with another technique called '**pulse charge injection**'. With this technique charge is imparted on to the particles flowing in the pipe and by comparing the amount of charge injected to that carried downstream by the particles the flow rate of the powder can be determined. What follows is a short review of some of the other techniques to give an insight into the other ways of measuring mass flow. Each technique will give references for further reading on that particular method which should give the reader a deeper understanding of the technique and methods employed to exploit it.

2.1 Capacitance Noise.

2.1.1 Overview.

The sensors used in this technique are embedded in the walls of the transport pipe and use part or all of the cross section of the pipe to form the dielectric of a capacitor. If the permittivity of the pipe changes due to the change in density of the powder, then there is a change in the capacitance. This change is then used to determine the mass flow rate.

2.1.2 The Technique and its Development.

In 1967 Beck et al. [3] proposed to use capacitance noise technique to measure mass flow rates.

Mass flow rate can be measured if two parameters are known:

1. Velocity of the powder **v** (cm/s).

2. Solids loading per unit length of conveyor **w(t)** (g/cm).

The proposed system had two types of transducers, one for loading and the other for the velocity information. The loading was found using standard gamma radiation absorption techniques.

Velocity information was derived by the cross–correlation of two capacitive transducers placed close to each other along the axis of the pipe.

In cases where the permittivity of the powder is either known or can be measured, then both the velocity and loading can be determined from the capacitive transducers.

Because the velocity of the powders is constantly fluctuating, an optimum time for the cross–correlation process needs to be determined. The preferred method of calculating correlation results would be to use an on–line digital computer. The use of this is justified as more and more plants are using control computers. The frequency spectrum of the transducers suggests a fast computer input selection unit would be required using transistor switches.

In 1969 Beck et al. produced details on the actual progress made in this approach [4]. By this time they had actually made use of a computer in the system. Plant tests were made by post processing of U.V. galvanometer recordings. Sample data from the charts was punched onto paper tape and the correlation function computed off–line. The computer loading for the cross correlation computation was 300 ms with a block of 295 data values from each transducer at sampling intervals of 500 μs.

In 1971 Beck et al. produced a new variation on the flowmeter, [5], which had only one transducer and therefore did not use the cross–correlation techniques. This was based on the observation that the noise is proportional to the flow rate.

With this new approach the output of the transducer is rectified and smoothed. This smoothed value is now passed to a meter drive for direct indication of the mass flow rate. The smoothing was variable between 5–300 s to cover a wide range of conditions.

This new version of the system used a new capacitive transducer which is only sensitive to variations in capacitance due to the transit of particles and is not affected by changes in standing capacitance due to powder sticking to the electrodes and other slow changing effects, hence no zero adjustments. This was achieved by standard filtering techniques.

In 1976 Green described a capacitance noise process using FM signal processing techniques instead of the AM used previously [6].

Here an oscillator is constructed using the capacitor formed by the transducer in parallel with an inductance. The frequency of oscillation varies with the random fluctuations in the flow. An FM demodulator gives voltage changes for the effective changes in frequency caused by the changes in capacitance. The demodulators are followed by a.c. current amplifiers configured to remove unwanted slower changes before another current amplifier and display.

Green stated that the flow can be regarded as a mixture of an average flow and a superimposed smaller irregular flow. The irregular flow gives rise to the flow noise. Experiments found that flow noise is proportional to the mass flow rate. These variations were tracked using FM demodulation.

Green concludes by saying that more investigation is justified by the findings of the improvement over AM transducers.

In 1981 Cardon, Green and John presented further work on this technique giving 7 applications where this flow meter is being used [7]. They conclude by outlining future developments; using paired transducers coupled to a cross–correlator with a microprocessor handling machine control.

By 1982 Green et al. published work which detailed a microprocessor based mass flow rate measurement of solids in pneumatic conveying systems [8].

The microprocessor was provided by the 6502 based PET computer which sampled both the concentration and velocity signals. From these it computes and displays the mass flow rate and total mass. The PET provides outputs suitable for controlling the feed rates.

2.2 Ultrasonic Methods.

The ultrasonic method measures the difference of transmission time of two ultrasonic transmitters. One transmitter transmits up the flow while the other transmits down the flow. The difference in time will yield the velocity of the flow. Again as for the Doppler techniques this can be used as an indication of the flow rate.

Kwan and Beck [9] tested both a capacitance noise technique and an ultrasonic technique in a gravity conveying system. The ultrasonic method they used was to pulse a piezoelectric transducer with a 40 KHz square wave. The ultrasonics passed across the flow to another piezoelectric transducer used as a receiver. The system monitors the disturbances in the ultrasound due to the presence of the flow.

In their experiments, wheat flour particles and p.v.c. granules 3 mm in diameter, were used with the two flowmeters at varying distances from the source of the flow and at varying inclines to it. When testing with the p.v.c. granules both systems produced satisfactory results close to the source (0.5 m), but at 3 m, both experienced saturation of flow noise. When wheat flour was used the flow stuck to the surfaces of the ultrasonic transducers and the performance of the system was greatly affected. The capacitance noise meter however, showed good results at 6 m from the source of the flow.

2.3 Electrostatic Noise Technique.

The electrostatic method detects the electrostatic charge built up on the particles by tribo, (frictional), charging. Work by King et al. [10, 11], used sensors embedded in the walls of the transport piping to pick–up the electronic noise generated by the flow. The system produces a measurement of flow rate and d.c. streaming currents.

Beck and Hobson used an electrostatic noise technique for the production of an Explosion Risk Meter for Pneumatic Conveyors [12], except in this case a similar transducer was connected to an a.c. voltmeter which was used to trigger an alarm to indicate the possibility of an explosion. Their work produced results which show how the electrostatic charge varies with moisture content and mass flow rate.

The meter had varying results for moisture content and mass flow rate and they concluded that:

a) Charge proportional to 1/humidity.

b) Charge proportional to mass flow rate.

c) Mass flow rate relation flattens at higher mass flows.

The flattening of the signals at high flows was attributed to particle collisions.

2.4 Impact Techniques.

This technique is based on the principle of counting the number of particle impacts on a sensor. The number of impacts being proportional to the flow. The sensor used for the impact detection should be small enough to allow only one particle to impact on it at any one time, yet sensitive enough to detect all the impacts.

Work done by Mann and Crosby [13], used a piezoelectric transducer and produced measurements of local particle flow rate, local particle velocity and the local velocity distribution among individual particles. The sensors they used needed to be calibrated for the particular particles in use. The particles they used were polyester spheres some 6.35 mm in diameter, weighing 0.15 g. The life time of the sensors used ranged from one to several hundred hours.

2.5 Doppler Techniques.

The Doppler techniques are based around the detection of a frequency shift in the reflected signal compared with the transmitted signal. A source transmits a signal into the flow and this signal is reflected back by the particles in the flow. Since the particles are moving the frequency of the transmitted signal changes as the particles approach and pass.

2.5.1 The Technique.

Basically a source of radiation is needed for this technique. Sources used are commonly laser pulses or microwaves. Doppler techniques yield velocity information. If you know that packets of a fixed amount of particles are transferred down the pipe then the average velocity of these packets will give the actual flow rate of the particles.

With the laser pulse technique there needs to be a clean window in the pipe so that the laser pulses can be transmitted into the flow. In a system such as the powder flow system, powder will build up on the window blocking the transmission path of the laser, hence the method is unsuitable for this application.

In addition microwaves transmitted across the flow can also be used to detect the density of the flow. With this method there is less need for a clean window as the microwaves can more readily pass through the walls of the pipe.

2.6 Nuclear Magnetic Resonance, NMR, Technique.

With NMR the particles are detected by sensing the interactions between an applied electromagnetic field and the magnetic moment of the subatomic particles of interest.

2.6.1 The Technique.

In the NMR process [14], the particles of interest are subjected to a strong fixed magnetic field where they undergo nuclear magnetization. The particles now enter a second magnetic field where resonance occurs when the correct radio frequency signal is applied to the resonator circuitry within the second field. A modulator, also within the second field's circuitry, creates a modulating magnetic field which in turn creates demagnetized pockets within the flow. A detection circuit now picks up the magnitude of the NMR response, which is proportional to the number of appropriate nuclei, (hence the particle density), within the detection field. Also the modulation effects from the demagnetized pockets is used to determine the velocity of the flow.

Knowing the number of particles and the velocity of them the actual flow rate can be calculated.

2.7 Electron Spin Resonance, ESR, Technique.

Electron spin resonance or electron magnetic resonance, EMR, is the same as NMR except here the response is not proportional to the number of appropriate nuclei but proportional to the number of unpaired electrons.

The work done by King and Rollwitz [14], in measuring the flow rate of pneumatically conveyed coal, combines the techniques of NMR and ESR for flows up to 15% by volume with velocities ranging from 8 to 30 ms^{-1}.

2.8 Gyroscopic Methods.

The principle of the gyroscopic mass flowmeter is that the total angular momentum of a circular loop of transport pipe is proportional to the mass flow rate of the particles flowing inside the pipe.

2.8.1 The Technique.

A section of tubing carrying the flow is bent into a circular loop and suspended vertically. This is then vibrated about its vertical diameter. A torque appears about the horizontal diameter of the loop having the same frequency as the input vibration and an amplitude proportional to the mass flow rate.

In the work done by Decker [15], an early version of this flowmeter used a constant speed motor drive to excite the system via a slot–and–eccentric–pin linkage. This meant that the moment of inertia of the loop included the total mass of the fluid and gave a resonant frequency dependent upon the density of the fluid. Another version of the flowmeter was created to eliminate this resonance. This version had a constant torque drive and a torque feedback loop to eliminate the resonance and hence the system's sensitivity to the density of the fluid.

2.9 Coriolis Force.

With the Coriolis force mass flow meter the transport pipe is arranged as part of a tuning fork which is excited. The particles within the excited section of pipe undergo Coriolis acceleration and cause angular deflections in the pipe. These deflections are detected and used to form the mass flow rate reading.

2.9.1 The Technique.

The Coriolis force mass flow meter is a non–intrusive meter insensitive to changes in temperature and pressure and can be constructed from varying types of material to reduce wear by the particular flow it is to be used on.

The flowmeter described by Tullis and Smith [16] employs a 'U' shaped pipe and a 'T' shaped leaf spring as opposite legs of a tuning fork. The fork is excited by an electromagnetic forcer and this subjects the particles within the pipe to a Coriolis type acceleration. The resultant forces create an angular deflection in the 'U' shaped pipe proportional to the stiffness of the pipe and the actual flow rate within it. The angular deflections are picked up optically twice during each cycle of the tuning forks oscillation. The pulses produced by the optical pick-ups are width modulated according to an amount governed by the flow rate. The width modulated signal is used to gate an oscillator feeding a counter and the resulting count is used as an indication of the actual mass flow rate.

2.10 Pulsed Neutron Activated Techniques.

The pulsed neutron activated technique can produce velocity and average density information about a two-phase flow based on the detection of short term radioactive tracers created by bombarding the flow with fast neutrons.

2.10.1 The Technique.

With the pulsed neutron activated method, Kehler [17], PNA, short lived radioactive tracers are formed in situ by irradiating the fluid with fast neutrons. By detecting these radioactive tracers and feeding the transducer outputs to a multichannel analyzer, mass flow velocity information may be calculated from the transit time of the tracers. Since the half life of the tracers created by the PNA method is short, (typically in the order of minutes), by measuring the total activity of the tracers the average density of the flow may also be obtained.

2.11 <u>Heat Transfer.</u>

Work has been carried out by Moriyama et al. [18] on a mass flow meter using heat transfer. Their work was on the development of a non−intrusive differential temperature sensing method for the measurement of mass flow rate in the region of 0−1,000 kg/h with mass flow ratio of 0−47.2 kg/kg. They were trying to analyze the relationship between mass flow, the heat transfer coefficient and the time constant of the meter, using dense phase 100 μm aluminium oxide powder.

2.12 <u>Optical Methods.</u>

The technique used by Mitchell et al. [19] in the transportation of coal in nitrogen, works with particles in the range of 0.5 to 2.0 mm. A beam of light is projected across the transport tube. Particles cutting the beam cause the detected light intensity to drop at the other side of the pipe. The output of the light detector is converted to an on/off digital pulse. The modulation created by a stream of particles is related to the actual flow rate.

In 1986 Morikawa et al. [20] worked on an optical fibre probe for the measurement of velocity and particle concentrations. Their work used fibre bundles with alternate fibres sourcing light, while the others received the reflected light. As particles pass over the fibre area, light is reflected from the particles and back down the receptor fibres. If the particles are moving with constant velocity then the pulse train received along the fibres has a constant frequency. Thus the frequency of the pulse train created by the combination of the pulses received by the fibre bundles is proportional to the velocity of the particles.

Generating the particle concentration requires further processing of the received fibre signals. The intensity of the reflected signal from a particle is proportional to its height above the fibre. Setting a threshold level on the received signal effectively sets the maximum height viewable by the fibres. Having a fixed height and fixed area, as defined by the fibre array, there is now a fixed viewing volume. By counting the number of particles entering this volume, the particle concentration is found. Determination of density is found by knowing the average mass of the particles and the concentration, hence the density can be calculated.

2.13 Related Areas.

A good visualization of the actual flow in the pipes may prove vital for the development of a good measuring instrument. Work in the visualization field has included the use of holograms. An example of such work for the study of two–phase flows is provided by Shorin [21]. In this work the authors used dual–beam holographic apparatus. They set–up their equipment using natural particle source like the spores of the puffball giving particles in the size range of 3–4 μm, and corn blight for 7–8 μm. Using this apparatus they examined water droplets in an aerosol form, and found that for droplets with a radius of ≤ 10 μm and velocities to 10 m/s, the droplets actually behaved like solid spheres, bouncing off a plate without splitting apart. Further background on the holographic visualization of particle flow can be found in a survey done by Trolinger [22].

The examination of low density powder flowing in a pipe could lead to the visualization of problems encountered in the measurement of this situation. A real–time situation which could help would be the combination of the real–time three dimensional colour visualization techniques built up Robinson et al. [23], and the powder flow rig used in this research. Creating windows looking into a well lit flow tube could produce three dimensional images of the transportation mechanism of the powder, answering questions about the cross–sectional profile of the flow. Looking near the sensors, the build–up of powder on the pipes walls may be seen and its re–entry into the flow observed. Examination by three dimensional x–ray techniques, also developed by Robinson [24], may further help in this study, by the elimination of the clear viewing and illumination windows. Multi–phase flows with differing particulates may be easily observed providing the particles have a different x–ray signature.

3. Project History.

The project has been devised by Dr B.C.O'Neill. During the early 1980s O'Neill was joined by C.A.Willis. Their object was to devise a means of detecting the mass flow rate of particles flowing in a pipe without interrupting the flow of the particles. During the first stage of their study they looked at the capacitance noise technique. Their work [1,2] showed problems in these techniques so they created a novel technique of pulsed charge injection. Their original work on this technique led to a development of an a.c. charge injection unit for the instrument. Using a.c. problems were found in the lack of pulse control, so a d.c. injection unit was developed. From this development all their work, and the current work on the project, has stemmed. In July 1983 this work was patented by the British Technology Group, B.T.G, [25].

3.1 Overview of the Previous Experimental Work.

In the initial study carried out by O'Neill and Willis [2], an experimental rig was constructed to convey powder pneumatically from a hopper through a purpose built charging and detection unit and then through a 14 mm (i.d.) pipe. Figure 1 shows the experimental apparatus used and figure 2 shows the arrangement of the charging and detection unit.

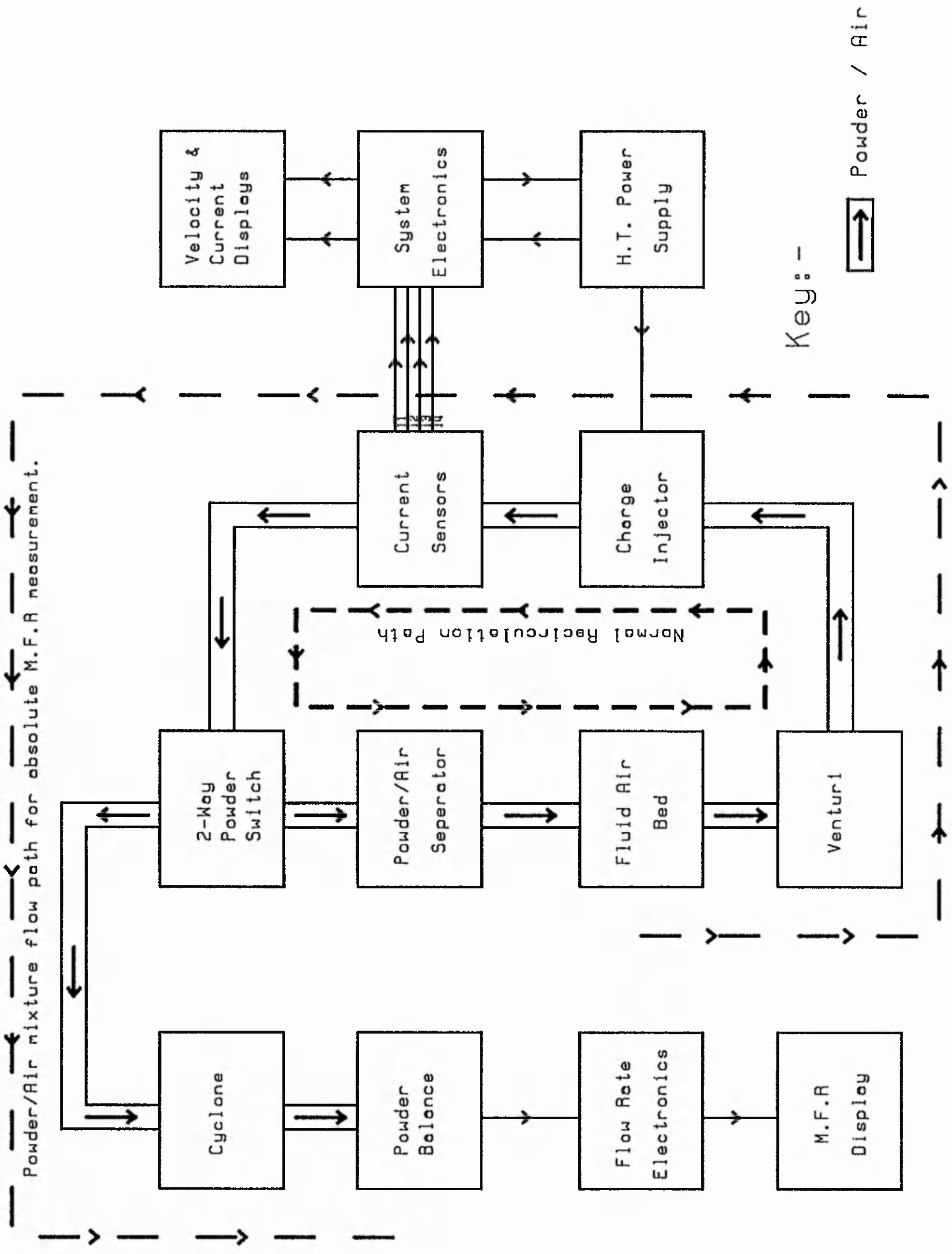

Figure 1 Previous Experimental Apparatus.

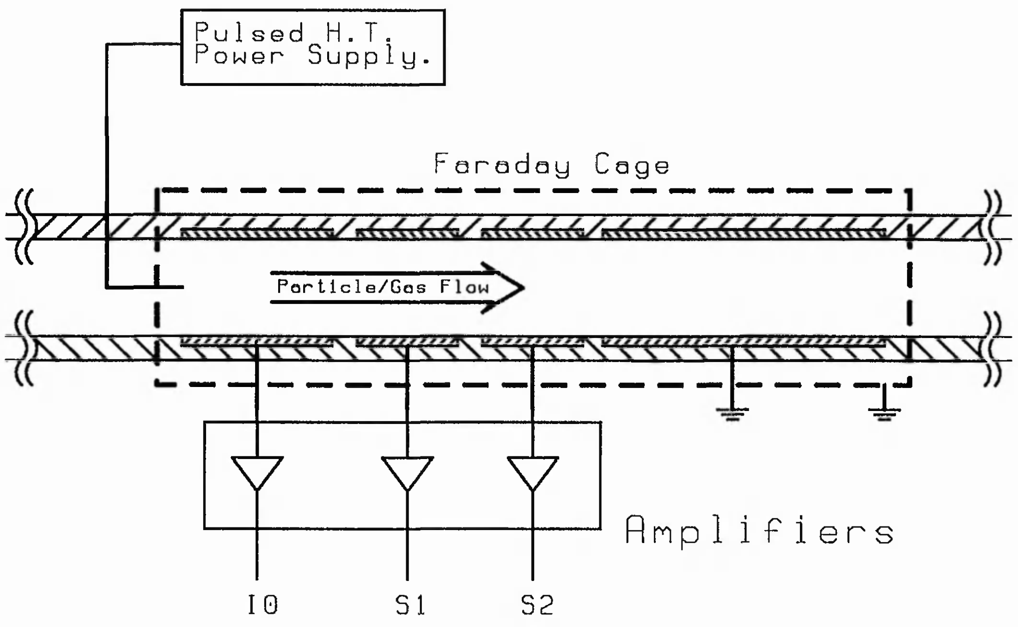

Figure 2 Previous Experimental Sensor Unit.

The powder was then separated from the air and returned to the hopper. It was therefore possible to run this system for several hours. It was also possible to switch the flow for a short period of time to an electronic balance to measure the absolute flow rate.

The charging unit consisted of an injection electrode which was a needle located along the axis of the pipe and a collecting electrode which formed part of the wall of the pipe. The charged powder was detected downstream by two adjacent electrodes similar to the collecting electrode. Further downstream a fourth electrode was used to neutralize any excessively charged powder.

Charge was injected into the system by applying a high voltage pulse between the injecting electrode and the collecting electrode. This pulse was obtained from an electronic voltage source which could control the magnitude and duration of the pulse.

The two downstream electrodes acted as '**Faraday cages**' to detect the net charge arriving and leaving the pipe, any charge neutralized on it, any charge settling on it. These charges were detected by current amplifiers. For mass flow rate measurements the current amplifiers were connected to peak detectors with a long time constant which were used to track the peak amplitude of the current pulses.

3.1.1 Current Waveforms.

Figure 3a shows a 'typical' current waveform caused by the application of a voltage pulse to the injecting needle. This current may be divided into two components. The first component is the current flowing in the circuit due to the capacitance of the system (see figure 3b). The second is the sharp step in the waveform which is due to the onset of charge injection into the powder/air stream. The capacitive current was simulated by electronic circuitry and then subtracted from the original current signal to obtain the injection current which was controlled to be a duration of 1 ms, figure 3c.

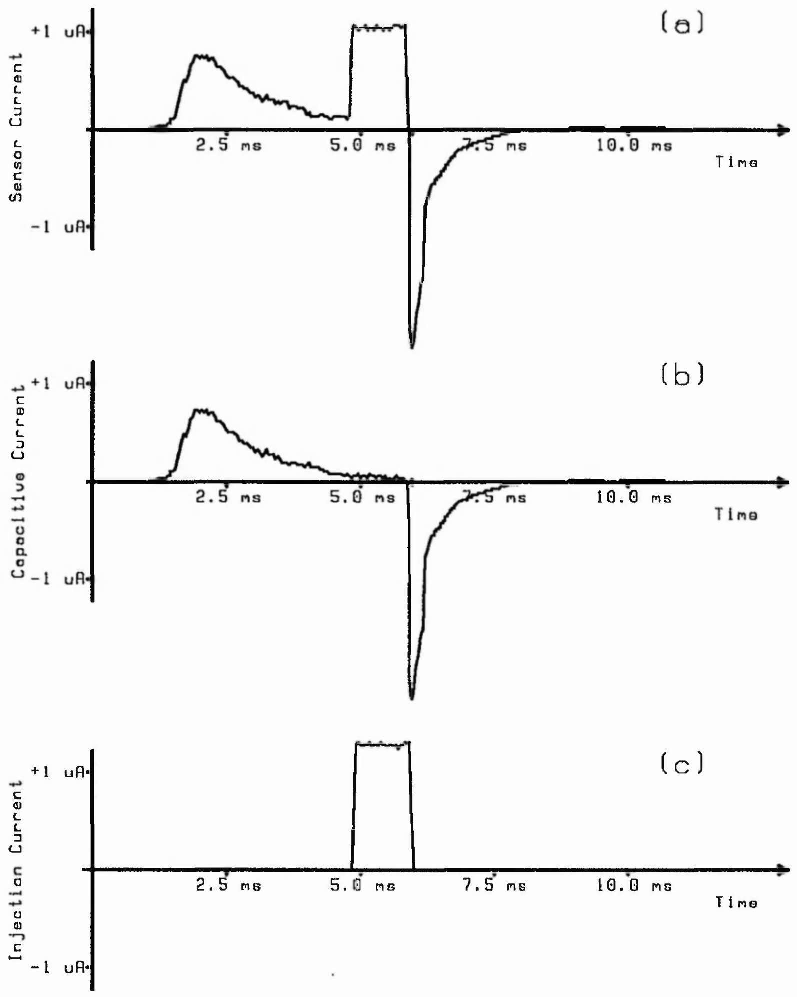

Figure 3 Injection Current Waveforms.

Figure 4 shows three traces for the first downstream sensor for low, medium and high flow rates, in the range of 0.5 to 6.0 grams per second, respectively. The second sensor gives a similar waveform except smaller in magnitude and delayed in time because it is further away from the injection electrode.

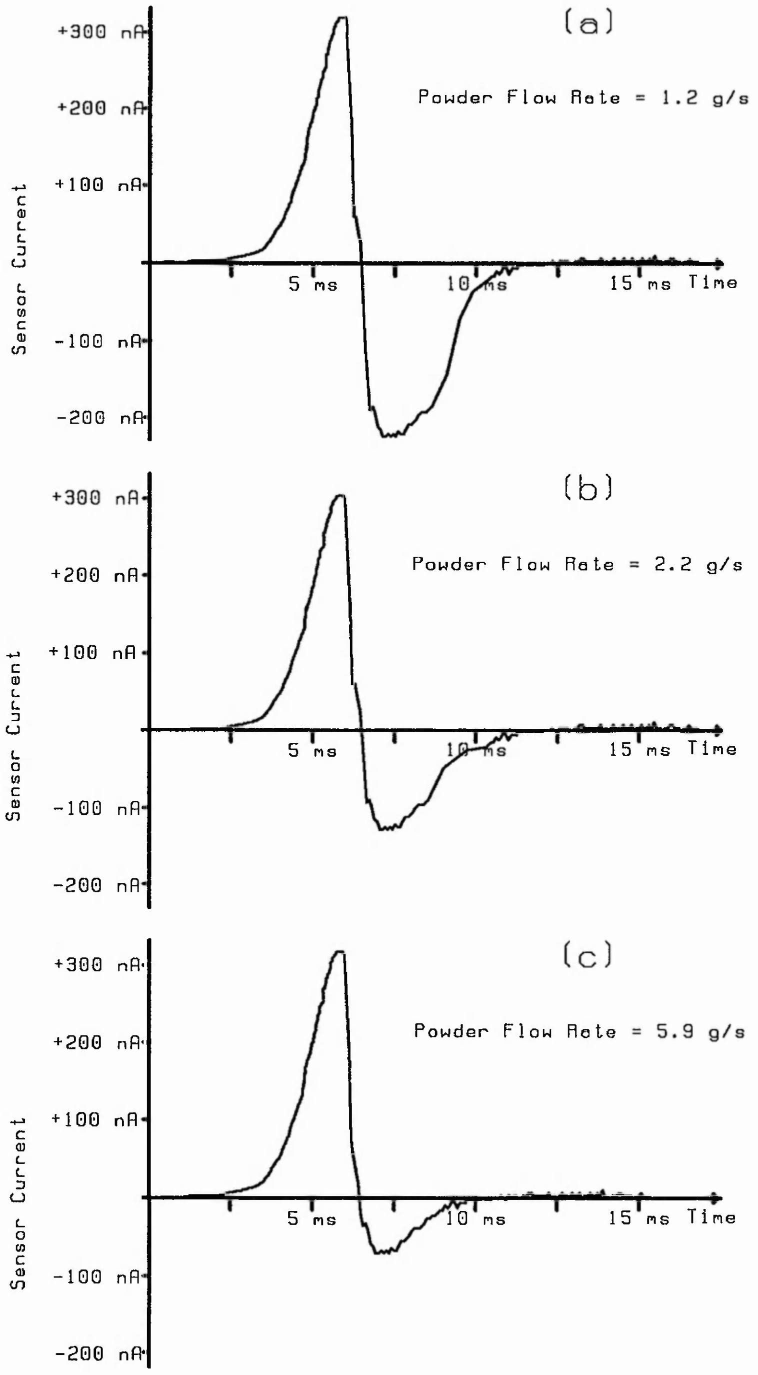

Figure 4 Current Waveforms at S1 for Various Flow Rates.

3.1.2 Mass Flow Measurements.

To obtain mass flow rates the mean peak height of the injection current and the sensor currents I_2 & I_3 were measured at various flow rates. The data obtained from a large number of such measurements was analysed on a mainframe computer using an interactive statistical modelling package called GLIM. The results of the analysis yielded two models for producing a mass flow rate from the peak values obtained from the sensors:

$$R_1 = 10xI_2/I_{inj} \quad \text{for low flow rates, and}$$

$$D_{22} = ((I_2-I_3)/I_{inj})^{1.4} \quad \text{for medium flow.}$$

where $I_{inj} = I_0$ and I_2, I_3 = current from sensors S_1 and S_2 respectively.

Two models were needed as the data had to be separated into two regions to produce accurate models for low and medium flow rates. Both models produce flow rates to within 10% of the measurable flow rate.

The separation of the data into regions was carried out by a computer program using the sensor data. This opened the way for a totally automatic processing system for continuous mass flow measurement.

3.1.3 The Basic Instrument.

After the initial work carried out by C.A.Willis, two teams of French students carried out some preliminary work on a hybrid analogue/digital system based around a 6809 microprocessor, [26,27,28]. This work showed that the 6809 processor could only carry out the basic processing and was not adequate for the additional processing necessary to obtain accurate measurements.

3.2 <u>Conclusion.</u>

The work done by C.A.Willis has shown the possibility for developing a mass flow instrument. The French students have shown that using a single 8–bit microprocessor gave insufficient computing power to perform the calculations needed for a commercial instrument. New types of system must be developed in order to achieve the aims of attaining a commercial mass flow instrument.

4. Development Design Path.

After the developments made by C.A.Willis [1,2], the project moved to a stage where realization of an instrument was needed. From this realization further investigation of the technique could continue leading to new theories.

4.1 Possible Design Options Investigated.

Looking back on the previous work done, there were three choices of progress:

1. Hybrid system comprising a balance of analogue and digital hardware.
2. Large analogue hybrid with small processor control.
3. Full digital multiprocessing system with a minimum of analogue interface components.

4.1.1 Equal Hybrid System.

The preliminary work was an equal hybrid system where the main characteristics of the input signals as detailed by Willis, were extracted and then processed by a microprocessor. This work had shown that a 6809 microprocessor was not powerful enough for this job and so this processor, and most of the other 8-bit microprocessors, should be ruled out for any future work on equal hybrid systems. This left 16/32-bit microprocessors. To utilize the initial work done, a 6800 compatible microprocessor interface would be required. This was one of the reasons why the 68000 family of devices were chosen.

The main characteristics of the sensor currents used by C.A.Willis were the peak values of the currents. The analogue systems used by Willis and the French students had the disadvantage that when charged powder which had built up on the pipe walls, broke loose and entered the flow, a large current spike was generated. This spike caused an error in the peak detectors because although the spike was in fact a peak, it was however, an undesirable peak which must be disregarded to obtain a true reading. The spike also had another side effect. This was that the peak detectors also averaged the input signals to avoid spurious readings due to the fluctuations in the actual powder flow. The averaging in the detectors was disrupted by the spike so giving large false readings for many cycles to follow.

A new hybrid system of this type would have to reset the peak detectors each cycle and would then need a signal conditioning phase such as digital filtering. The microprocessor would still be responsible for the actual flow rate calculation after this.

4.1.2 Large Analogue Hybrid System.

The large analogue hybrid system would take the previous hybrid idea one stage further. In this system the microprocessor load would be vastly reduced by analogue devices actually calculating the mass flow rate from the peaks found by the detectors.

A large hybrid system would have all the problems associated with an equal hybrid system. Detailed analysis of the sensor signals would be required in order to determine the necessary processing. Since the flow rate equations could not be easily updated and the true nature of the signals to be processed had not been determined, the large analogue hybrid system will not considered until the full system requirements have been determined.

4.1.3 The Multiprocessor System.

The multiprocessor system would comprise one microprocessor which captures the waveforms and extracts the basic characteristics. This processor would then pass the extracted data to one or more processors which would perform the flow rate calculations. Whilst the second processor calculates the flow rate the first could start processing the next cycle.

4.2 Choice of Design.

Using the basic system developed by Willis, the sensor output currents were examined using a high speed digital storage scope and a chart recorder. It was found that all the signals obtained from the transducers contain a large element of noise and also have spikes due to extra charged powder re-entering the flow from the pipe walls. In a hybrid system, fast reacting non-linear filters would be needed to cut the high frequency spikes before they enter the peak detection unit. The large hybrid system would be the fastest of all the systems except that the equations would all be fixed and the system would not be easily adaptable to new situations.

When lower flow rates are introduced there becomes a poor signal to noise ratio. With a powerful multiprocessor processing unit which has adaptable analogue input stages, there would be more chance of yielding useful information about the flow. A total digital system would also be reprogrammable for different situations.

It was decided that on grounds of flexibility a total digital system would be implemented. This system would be used throughout the research phase with the possibility of returning to a hybrid system modeled on the digital system and its final processing technique.

4.3 STAGE 1. Design of the Multiprocessor System.

4.3.1 Choice of Microprocessors.

Based on previous arguments regarding the choice of microprocessor, it was decided that a dual Motorola 68000 based system would meet the requirements. Motorola 68000 microprocessors were chosen over 68008 processors available within the department, because of their speed. The 68008 is a 68000 16/32-bit microprocessor with only an 8-bit external data bus. The instructions for the processor are all 16-bit instructions so the 68008 would need twice the time to fetch the instructions needed. In practice the 68008 is not half the speed of a 68000 but there is a significant speed reduction when dealing with 16-bit information.

Data exchange between the two 68000 microprocessors would be via a dual port RAM for speed. With a dual port RAM, data to be exchanged can be easily ordered by writing randomly to different addresses in the RAM. When it is time to transfer data a write to a special RAM location would cause an interrupt on the other processor. Using FIFOs the data to be exchanged would need to be carefully ordered by the sending processor so that it matches the order expected by the receiving processor. This could mean withholding results calculated now until others are computed so that when written they form the correct sequence.

Another mode of transference would be to use 68000s and RS232 interfaces. If the communications overhead become too large and starts to slow down the system, then direct memory access units (DMAs) could be used. Adding an extra RS232 interface to each 68000 would increase the circuit complexity and also the data transfer time and so is not desirable.

If 6800 device compatibility was not required then 68070 microprocessors with their built in RS232 interfaces could be used. The 68070 is basically an enhanced version of the 68000. The 6800 compatibility has been sacrificed to allow the addition of a built in clock generator, on–chip memory management unit, 2 DMA channels, an RS232 interface, I^2C serial bus interface, (inter–chip communications), a 16–bit timer/counter and two 16–match/count/capture registers. Another advantage of the 68070 processor is that the I^2C bus could be used for the data exchange, thus saving the RS232 ports for instrument/terminal communication.

4.3.2 Choice of Analogue to Digital Conversion System.

During the development of the 68000 based system several designs of analogue to digital converter system were evaluated. Basically, for the range of input signals that the system needs to handle, a converter system with either a large range or the ability to adapt to the signals, had to be chosen. The systems evaluated included linear quantizers and adaptive quantizers. For resolution, a 16–bit converter system would be desirable to cover the range, but at any given instance an 8–bit converter would give the resolution required if the range was altered to match the input signal. Work was done to simulate an 8–bit adaptive quantizer and evaluate it against a 16–bit linear quantizer. Another primary concern was that the converter system should be fast.

4.3.2.1 Analogue to Digital Converter Theory.

For accurate conversion the maximum rate of change of the input signal must not be larger than half a least significant bit, LSB, in the converter, otherwise the output of the converter will not represent the true value of the input signal. In successive approximation type converters this could lead to a totally wrong output due to the conversion technique.

With a 16–bit converter with a conversion time of say 8 μs, and an input range of 10 volts, the maximum frequency of an input signal of say, $5\sin(\omega t)$, without the use of a sample and hold, is given by:

$$\begin{aligned} y &= 5\sin(\omega t). \\ dy/dt &= 5\omega\cos(\omega t). \end{aligned}$$

The maximum rate of change of the input signal is:

$$\begin{aligned} dy/dt &= 5\omega \\ &= 10\pi f \ Vs^{-1}. \end{aligned}$$

Now one LSB $= 20/2^{16}$ V.

$\approx 305\ \mu V.$

So the maximum acceptable rate of change is 152 μV in 8 μs, which gives 19 Vs^{-1}.

$$\therefore \qquad 10\pi f = 19$$

Hence the maximum frequency, f, $\approx$ 0.6 Hz.

Where **y** represents the input signal, and **dy/dt** gives the rate of change of the input signal.

As the frequency components in the flow system are far greater than 0.6 Hz, for a large converter an accurate sample and hold circuit would be needed.

Typically a "**fast**" 16–bit converter was classified as 32 samples per second using dual slope techniques or 60 μs for a converter with track and hold amplifier using successive approximation with an effective accuracy of 14–bits.

4.3.2.2 The Actual Conversion System Choice.

With any conversion system chosen it would be undesirable to have four such conversion systems, one for each signal, so a multiplexed system would be needed; the undesirable factors being circuit space, added system complexity and cost. Now with fast signal multiplexing the effective signal bandwidth is increased, thus the effective conversion time of any analogue to digital converter must also be increased.

Due to the difficulties in finding a fast 16–bit analogue to digital converter and even an accurate sample and hold circuit, the adaptive quantization circuit was chosen. This all led to an 8–bit adaptive quantizer based around an 8–bit '**Flash Converter**' being developed for the system as the larger analogue to digital converters typically took 3–12 µs for 12 bits, to convert a signal whereas the flash converter has a 50 ns sample/hold and convert time.

4.3.3 Work on the Chosen System.

For the 68000 work a 68000 development system was needed. This system was based around the Atari 1040 ST because the Atari could offer a single unit solution with easy access to a keyboard and screen. The features of the Atari could also be utilized in any final instrument design.

The Atari computer was not originally in a form which would allow development to take place so the computer needed modification. A circuit was designed which enabled the user to cause interrupts, take control of the computer busses, and to interface memory and/or peripherals to the system. This circuit was designed to fit inside the Atari computer for practical reasons. Before the insertion of this unit, the Atari would not allow any of these features due to its memory management system which knows exactly what the Atari should have in its memory areas and does not allow units which it has no knowledge of.

The first version of the adaptive quantizer was a complete stand alone Data Acquisition Unit which gave an effective 16–bit resolution, (8–bits and gain), answer in 480 ns. With further refinement and close consideration to circuit timings, this time was reduced to 380 ns. With this high speed converter system and a 8 MHz 68000 processor based unit it was found that the software could, reading and storing at its fastest, reduce the sampling speed for 4 signals to 12 μs. This gave, over a 20 ms period, some 1666 samples per signal. The next stage of the system software took another 80 ms of processing time to extract the characteristics of the waveforms. The data should now be transferred to the other 68000 for final data processing before the flow rate for a system could be displayed or control of a spraying system started.

For a practical controlling system the unit should be able to service up to eight units giving an update in information within 1 second. It can be clearly seen that a single 68000 based system may just meet this criteria but leaves no extra processing time for future development.

4.4 Transputer Designs.

During the development of the 68000 based system a new type of microprocessor was studied. This was a high speed processor which can be easily linked by a two wire system to upto four other processors. Each of these communication links has an in-built DMA controller for high speed communications upto 20 Mbps. This new processor is called a transputer and is designed for use in parallel processing applications. The Polytechnic's Computing Services installed a software **Transputer Development System**. The equivalent 68000 processes were written for the transputer in Occam, the native language of the transputer, and timings evaluated. With more investigations a new, previously unexplored, design route opened.

The basis of the system is a sensor head which contains four sensors. To develop a control system a digital instrument is needed which has the capability to make decisions on the results calculated.

The optimum transputer system, optimized for speed, would comprise 5 transputers. There would be a Processing Array of four transputers for real–time data processing where each transputer would handle one signal and be fed pre–formatted 16–bit data from a high speed adaptive analogue to digital conversion unit. The Processing Array would then feed the characteristics to a final transputer for flow rate calculation and feedback control.

As the optimum system involves four adaptive quantization units, and four acquisition units have already been ruled out during the 68000 work, the next choice is a system where all the transputers are connected to one signal conditioning unit. The problem with a system of this nature is the interconnection between the transputers and the analogue to digital converter. There are two main ways in getting the data from the converter system to the transputers. The first is to have a multiplexed bus where the transputer busses are multiplexed into the converter system or four separate access ports on the converter system, one for each transputer. The second is to have the output from the converter clocked into data latches which feed Inmos Link Adaptors which in turn are connected to the transputers via their links.

The first solution would involve a complex board construction and would defeat the object of having minimum transputer processor systems linked together to form the Processing Array. The second solution would keep the minimum processor arrangement and give a fast converter and distribution unit.

With any system there is a need for the detection of charge injection. In the 5 transputer based system one of the transputers in the Processing Array would be used for the injection detection. Another solution would be to have six transputers where the sixth transputer would be incorporated in the quantization unit to replace the complexity involved in the link adaptor solution. This additional transputer could also be used for data processing and the actual converter system would be further reduced by the addition of a programmable logic device which with the transputer, would replace a large amount of the original conversion circuit.

4.4.1 Final Decisions. Transputer Vs 68000.

The flow system is to handle 16–bit data, (8–bit with a dynamic range of 16–bits), and process it. For speed the arithmetic should therefore be done with more than 16–bits at a time. For this reason the T212 16–bit transputer had to be ruled out leaving the T414 32–bit transputer or the proposed T800 floating point transputer.

As the data from the analogue to digital converter was in a fixed point representation of the real input signals, the data could be handled easily with integer arithmetic so there was no need for the T800 floating point transputer on the Data Acquisition Unit nor on the data Processing Array. Since the data represents real signals, and powers of this data are to be taken, then to preserve accuracy floating point arithmetic is needed for the final computation element. The T414 processors handle floating point arithmetic with software routines taking up extra memory space. The T800 floating point transputer has its own floating point unit inside it and the floating point routines of the T414 are micro–coded in the T800 and take up no external memory. The floating point arithmetic is also a lot faster on the T800 than on a T414. For this reason the T800 is to be used in the final processing stage. If the T800 was not ready by the time the system was built, then a T414 could be used and replaced at a later date, as the T414 and T800 are pin compatible. The T414 software would only need recompiling for the T800.

Preliminary work on the Vax based Transputer Development System using Proto Occam, showed that with a multiprocessor unit comprising four processing nodes, one for each signal, the transputers could not only handle the data throughput but also do the first stage data processing with the 68000 initial sampling time of 20 ms. A total saving of 80 ms by doing real–time data processing. A further advantage of this system was that as the data was processed in real–time there was no need for large memories for data storage, in fact the transputer program and variables could easily fit within the transputer's internal 2K of RAM leading to a minimum system which basically consists of four bare transputer chips.

This all led to a transputer system with an octahedral structure, figure 5, comprising a transputer controlled Data Acquisition Unit, which distributes data to a Processing Array, and a real–time data Processing Array of four transputers. The transputers in the array then pass the results of their calculations over to a final processing and control unit for final computation.

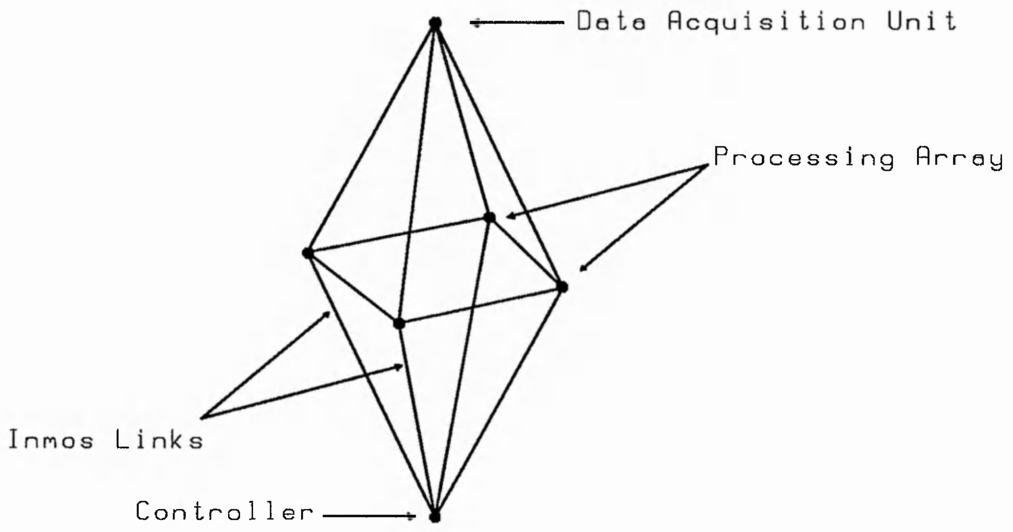

Figure 5 Diagrammatic Representation of the Transputer System.

The system is now a fast pipeline structure with real time data processing continuing in parallel with the data collection. The final results can then be calculated in parallel with the start of the next cycle. This arrangement provides a powerful basis for the research.

Although the loss of the 68000 work would be a great set back for the project it is felt that this solution will withstand the greater computing demand to follow in the data analysis sections of this research. This new design will enable the testing of many different ways of processing the data and could later lead to the reconsideration of a hybrid system after the signal processing methods have been identified.

4.5 STAGE 2. Realization of the Design.

The first design for the Controller section of the system incorporated many extra features such as a reconfigurable digital signal processing section, a 68000 development system area with dual port RAM data exchange to a transputer development section. This design had to be dropped due to production difficulties. Attention was focused on the Data Acquisition Unit and real-time Processing Array before a final design of the controlling section was completed. What follows is a discussion of the design and construction of the various elements making up the transputer design.

4.5.1 System Software.

The software for the transputer system was created on the Vax using Proto Occam. The system was simulated by adding a simulation of the Data Acquisition Unit and analogue inputs. The simulation confirmed the design theory.

When one of the off-shoots from the original Controller was ready, software developed on the Vax was downloaded by using a Transputer Development Workstation written in 68000 assembler on the Atari 1040 ST and an Atari to Inmos Link Conversion Unit developed for the task. It was at this point in the project that problems were found with the Vax system. The code extraction and downloading had difficulties which the Computing Services could not help with. The problem was in the code extraction. This arose as full access to the Vax system was not available. It was about this time that we were able to obtain backing via the SERC/DTI Transputer Initiative in the form of a IBM Transputer

Development System, TDS, running Occam 2 on a T414 host processor. Then all the Vax software was ported to the IBM and converted to Occam 2. The original programs which had difficulties in extraction and downloading from the Vax, now worked.

From the original Vax system simulation, system software was developed. An IBM Transputer Development System was purchased for research as the SERC/DTI system was being returned as the loan period was nearing expiry date. This development system had a T800 host and was used as a server for the transputer based system in the flow project.

4.5.2 The Data Acquisition Unit.

The design of the data acquisition was based around an erasable programmable logic device, EPLD, containing 1210 programmable logic elements, an analogue signal multiplexing unit, an adjustable gain section feeding an 8-bit flash converter and a T414 32-bit transputer.

During the design phase the EPLD logic was simulated using QUICKSIM which is Mentor Graphics Logic Simulator running on an Apollo. The simulation confirmed the logic function of the EPLD. The logic array was then programmed using the Altera Logic Programming package on an IBM PC.

The circuit board was laid out using Mentor Graphics BOARD STATION. The output of the package was a two times artwork for the printed circuit board. This artwork was completed and sent away for production of the printed circuit.

The board was then built up and tested. Operating difficulties caused a redesign of the signal conditioning unit and reprogramming of the logic array. Several new designs of signal conditioning elements were created but none put into practice as subsequent changes in the software removed these errors and enhanced the entire system. These changes are as follows:

a. The calibration software provided the system with the knowledge of the true gains of the circuit and any offsets associated with them.

b. By modifying the adaptation technique used by the system so that it keeps a fixed gain for one entire injection cycle, only 8-bit data was needed to be transferred from the Acquisition Unit to the Processing Array. Once the results were calculated they could be adjusted to take into consideration the true gains and offsets. By not using intra-sample adaptation the data transmission requirements have been reduced.

4.5.3 The Real-time Data Processing Array.

The real-time Processing Array comprises four T414 transputers without any other logic chips, (except those providing a clock and reset). The transputers have all their unused signal pins terminated with resistor networks to reduce the chance of static damage.

The printed circuit board was laid out using SMARTWORK which is a basic pcb layout package for the IBM PC. The artwork was again two times artworks and were completed by hand. The pcb board, because of its simplicity of design was produced within the department.

All four processors on the board now run the same program greatly simplifying programming of the system.

4.5.4 The Main Control Unit.

The Controller design was entered into the Apollo for printed circuit board layout by BOARD STATION. The board design evolved from the original high specification through various levels of complexity to its final implemented form. During this design path the board became a compact 6 layer printed circuit with integral power planes. Due to the financial constraints of the project it was felt that this board should be built using non–multilayer technology thus leaving money available for future circuit boards, if they are needed.

The board was change to a double–sided pcb measuring some 18 inches by 10 inches and was planned to contained a T800 32–bit floating point transputer, 1 Mb of high speed static RAM and a full expansion bus.

At this stage an offer was made of a free demonstration of a photoplotter. This offer was taken up due to the time which would be saved, although it was known that the track widths on the power rails were not of the required size for the current capacity of the circuit. When the board was finally complete, extra power line supports were added to reduce the volt drops across the circuit board.

During operation of the circuit, the T800 based Transputer Development System found difficulties in communicating with the T800 onboard the Controller. The problems with communication did not exist if the development system could talk to the Controller's T800 via a T414 transputer. This problem of communication was exactly like the problems highlighted in the release notes of earlier versions of the T800. Since both the T800 in the Controller and that of the Development System were the new T800C full specification versions of the T800, this communication problem should not be occurring. When the T800 was replaced with a T414 the whole system ran without the communication problem.

Nearing the end of this research, the problems of communication returned. Problems with the memory onboard the Controller also appeared. As time went on the board started becoming more and more unreliable, until it was deemed to have failed completely. The problem with the circuit lay in its physical size. During the life of this project the system was moved about from the development area to the actual laboratory area. It was also moved within these areas. The penalty of this movement and board size is that the fine tracking and vias started to fracture causing intermittent loss of signals.

The design was reviewed and again returned to the Apollo. This time the board was vastly reduced in size by returning to a 6 layer printed circuit board and increasing the packing density beyond that previously explored. This board had two internal power planes, increasing reliability. Extra features were added to the system so that the busses of the transputer could be taken over and the RAM onboard the card could be interrogated. Further reliability and circuit security were added along with these extra features by the inclusion of a 1200 gate EPLD. This device replaced some 9 integrated circuits on the board reducing the number of device interconnections by on–chip routing.

The photoplots were now produced in-house and the final board became 11x7 inches, with 4 signal layers and two power layers, containing approximately 2000 holes, (vias and I.C. pads).

5. General Developments within the Project.

5.1 Atari Modifications.

During the work on the 68000 based systems, a 68000 development system was required. Most of the 68000 development systems available were either too expensive or were based around the 8–bit bus version of the 68000, the 68008. This application calls for high speed processing of 16–bit data so the 68008 based units were no good for development. The Atari 1040 ST was an 8 MHz 68000 based microcomputer but did not have the ability to be easily expanded.

The basic concept behind the Atari work which followed, was to generate a 68000 system which would allow the transference of software written on it, to a 68000 based target system. The Atari offered a small package with the ability to generate code for a target system. The other features of the Atari could be used in any final instrument design. The keyboard and screen would provide a means of operator interaction and the disc drive, a way in which the system software could be updated.

A target for the system was chosen. This was a GESMPU14 offering a 68010 processor running at 8 MHz, dual RS232 interfaces, a real–time battery backed clock and calendar, provision for a floating point co–processor, 256x4 bits of non–volatile RAM, plus sockets for EPROMs.

The GESMPU14 also offered an industrial standard bus interface, the G96. The G96 bus is upward compatible with the G64 and so together could offer a way to a wide variety of standard computer boards for input/output.

The Atari has built–in protection circuitry which stops the user accessing areas of memory unknown by the operating system. This circuitry needed to be defeated to free up the Atari's memory space. To overcome the non–expandability of the Atari 1040 ST due to the protection circuitry and the lack of a buffered expansion port, a complete circuit board was designed which accommodated the 68000 processor, dual buffer sets, and a special arbitration unit which would arbitrate between the Atari's internal hardware and the user's hardware and hide the operations which it knows about from the Atari's protection circuits.

What follows is a description of the hardware used to defeat the Atari protection logic and a look at how it was fabricated.

5.1.1 Description of the Hardware.

The modification board consists of a new site for the Atari's 68000 central processing unit, (CPU), buffers for all the CPU's signals and an arbitration unit. The arbitration unit has the job of enabling the required buffer set and masking the error of accessing user modules from the protection circuits. The buffers are quite straight forward and are therefore not described.

The arbitration unit consists of 3 arbitrators.

1. Memory arbitrator.
2. Interrupt arbitrator.
3. Bus arbitrator.

These 3 arbitrators all arbitrate between the existing Atari hardware and any user added units. Without this unit the Atari could not be expanded as the internal memory management system halts the processors operation when any access is made to a location which violates its idea of the system memory map.

5.1.1.1 Memory Arbitration Unit.

The memory arbitration unit allows units unknown to the memory management of the Atari to be interfaced to it. This unit relies on the external memory address decoding and the Atari's own decoding to decide whether the Atari's memory management has come to the right decision about any given memory access. The unit controls the '**Bus Error**' signal given to the CPU when an access is made to an unknown memory location, and also controls the buffers for data flow and the data acknowledgments passed to the CPU.

Table 1 shows the results of the arbitrator for a given access.

Atari	External	Result
No Error	No Error	No Error
No Error	Error	No Error
Error	No Error	No Error
Error	Error	Error

TABLE 1 Bus Arbitration.

The result is in fact the effective bus error signal passed to the CPU.

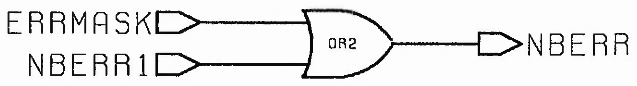

Figure 6 Memory Arbitrator.

Figure 6 shows the simple logic inside the EPLD which handles the memory arbitration. The EPLD is covered in the fabrication section, section 5.1.2. The signal **ERRMASK** is generated from the **NDTACK** signal created by the user logic, and the signals **NBERR1** and **NBERR** are the original Atari signals, **NBERR1** being connected to **NBERR** in the Atari.

5.1.1.2 Interrupt Arbitration.

The interrupt arbitrator arbitrates between the Atari internal interrupt devices and any external interrupt generating devices. To do this, the arbitrator compares the internal level of interrupt with that of the external device. The Atari was given the highest interrupt priority, so if the Atari's level is greater than or equal to the external level then the Atari's level is passed as the interrupt level to the processor. Otherwise the external interrupt level is passed to the processor. The CPU has a processing level in its status register. It compares the interrupt level it is given with that in its status register. If the value in the status register is greater than the value present on the interrupt lines then no interrupt takes place, otherwise it acknowledges the interrupt. Figure 7 shows the interrupt level arbitrator. Signals **NA**$_n$ are the original Atari interrupt lines. The interrupt lines to the processor are now **NIPL**$_n$.

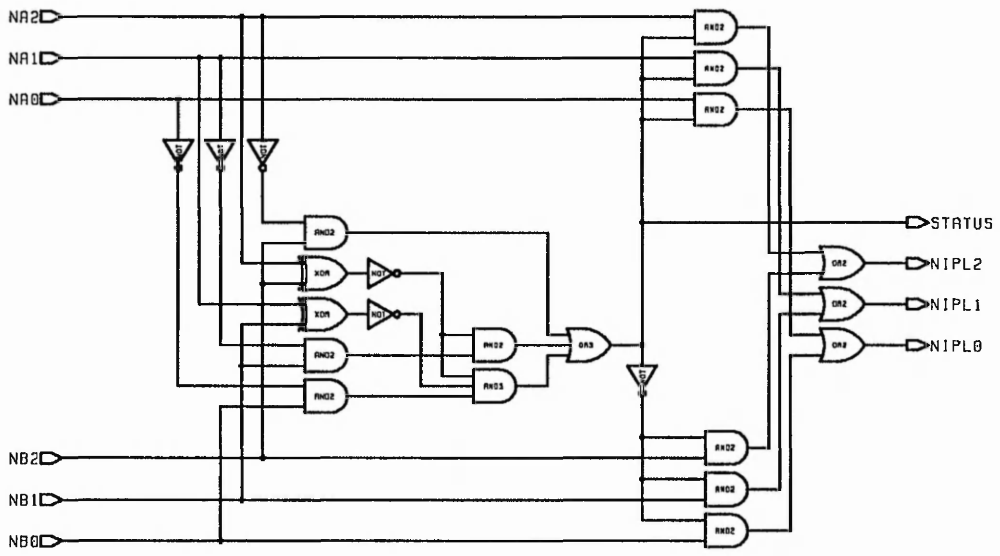

Figure 7 Interrupt Level Arbitrator.

As a diagnostic feature the arbitrator has the output of the interrupt request comparitor available on the pin labelled **STATUS** on the pin diagram of the EPLD, appendix F. **STATUS** indicates which interrupt, either external or internal, has the highest priority.

When this acknowledgment is given the arbitrator masks off the acknowledgment cycle from either the Atari or the external device depending upon the state of the arbitrator's interrupt level comparison at the time. This logic is shown in figure 8. Signal **NENA** goes to the enable on the tri-state buffer buffering the 68000 signals, **NAS** and **FC**$_n$ to the Atari's logic and **NENB** does the same for the external logic.

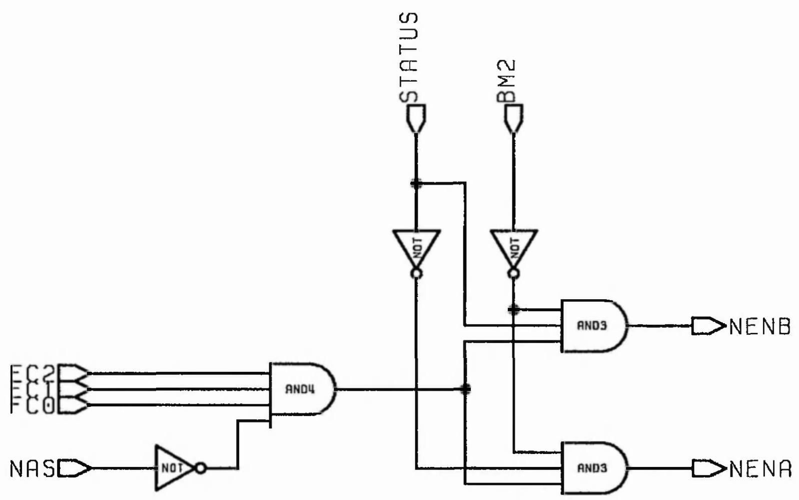

Figure 8 Interrupt Mask.

5.1.1.3 Bus Arbitration Unit.

For bus arbitration the Atari was again given the highest priority. The Atari's circuitry is connected to the signals labelled **XXX**$_1$. The actual priority arbitration only lasts until the rising edge of the next clock cycle whereby the EPLD commits the system to the service of one or the other bus requests. Looking at the state diagram in figure 9, the branches off the centre represent the application of the bus requests. The state of each bus request is only important when the clock has a 0 to 1 transition where the state changes from the inactive centre to one which follows a logical path in granting the bus. Figure 10 shows the main logic of the bus arbitrator. Due to the internal design of the EPLD extra external logic was needed to create the bus master enable signals, **BM1EN** and **BM2EN**. The EPLD does not have the ability to have separate clocks on the register elements and since two elements are already fed by the 8 MHz system clock, and as the other elements require two different clocks, they were implemented externally.

The state diagram, figure 9, follows on the next page and figure 10 showing the main logic of the bus arbitrator follows after that.

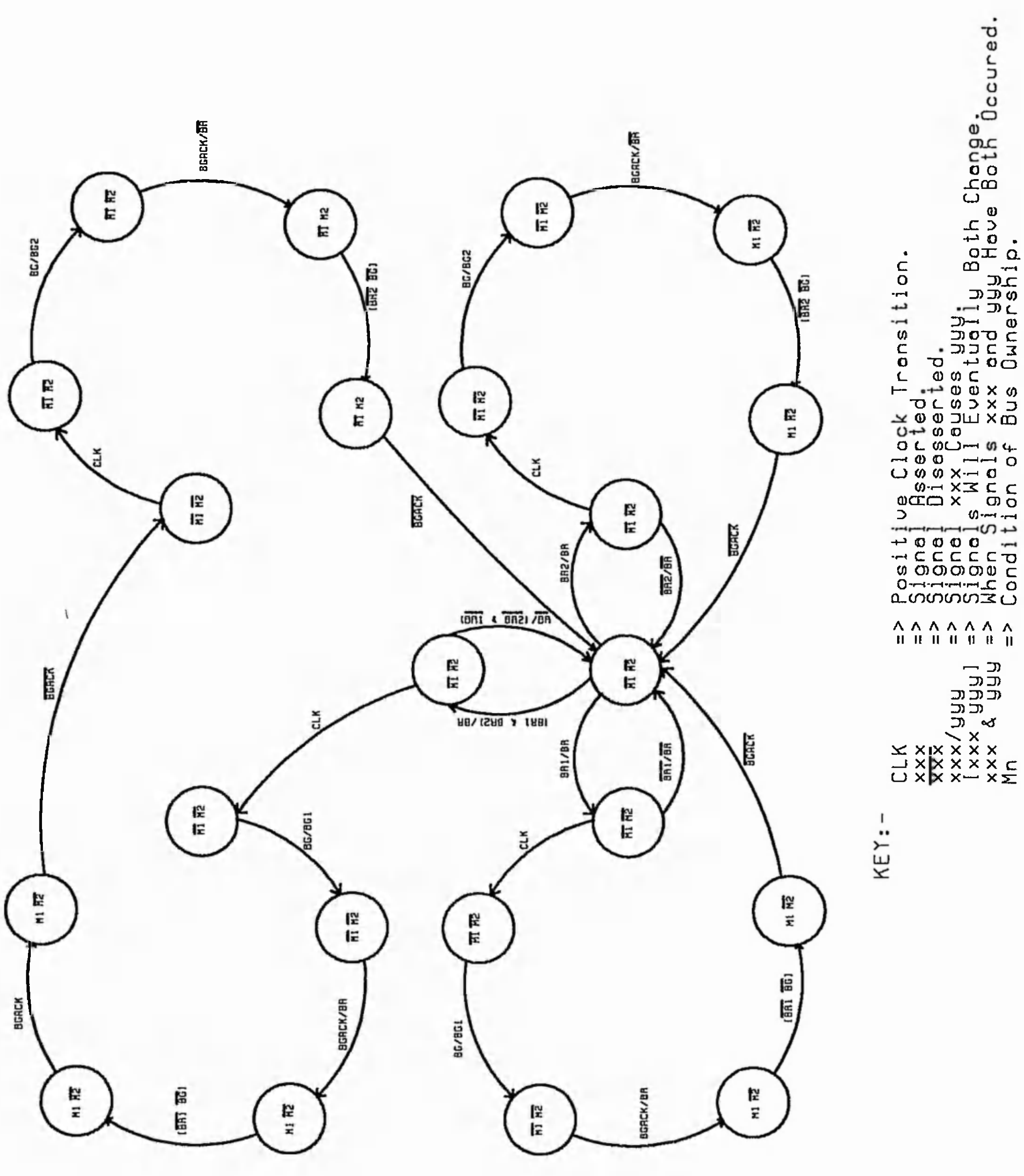
KEY:-
CLK => Positive Clock Transition.
xxx => Signal Asserted.
$\overline{\text{xxx}}$ => Signal Disasserted.
xxx/yyy => Signal xxx Causes yyy.
[xxx yyy] => Signals Will Eventually Both Change.
xxx & yyy => When Signals xxx and yyy Have Both Occured.
Mn => Condition of Bus Ownership.
CLK
BGACK
BG/BG1
BG/BG2
BGACK/BR
BR1/BR
BR2/BR
[BR1 BG]
[BR2 BG]

Figure 9 Simplified State Diagram of the Bus Arbitrator.

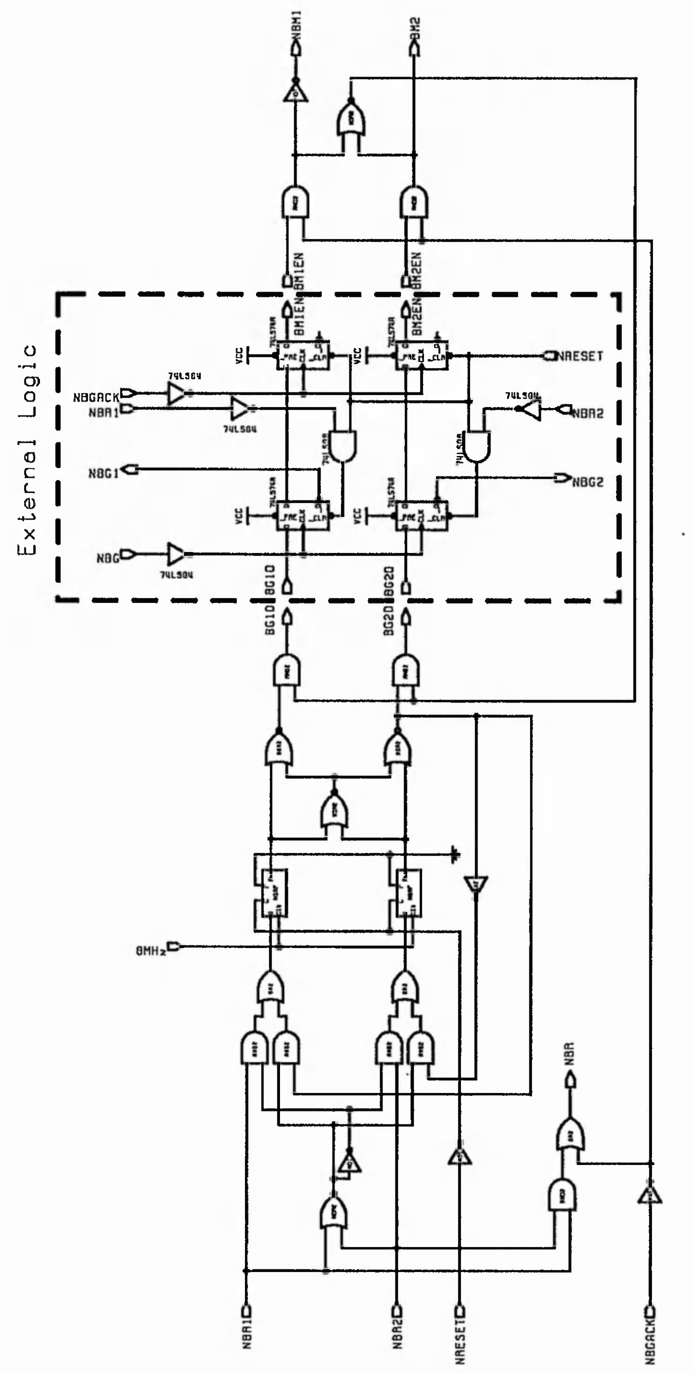

Figure 10 Bus Arbitration Unit. Main Logic.

When the external system takes control of the bus, some of the buffers in the system need to reverse their normal direction, for example the address bus no longer comes out of the Atari but flows in allowing the external system to interrogate the internal memory of the Atari. The logic for this is shown in figure 11.

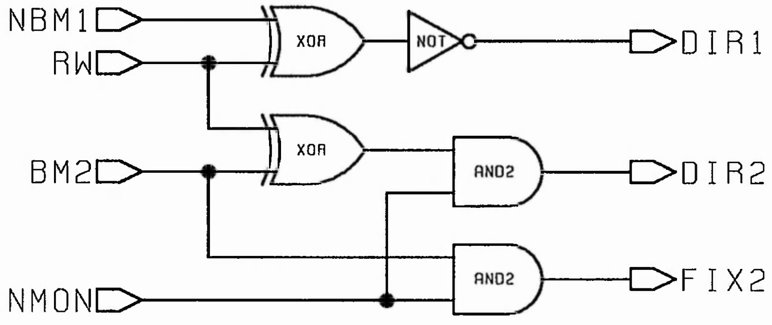

Figure 11 Bus Arbitration Unit. Buffer Control.

In figure 11 **DIR1** controls the buffers for the Atari side of the 68000 CPU and **DIR2** controls the buffer direction for the externals system. To aid monitoring of the busses within the Atari, an extra signal **NMON** has been provided which when active causes the normally fixed inward facing buffers, (facing into the Atari), to change direction and allow the signals to flow out of the Atari.

5.1.2 Fabrication.

For compactness and simplification of the PCB, the bulk of the arbitrator was constructed using an erasable programmable logic device, (EPLD). There were two slightly different versions of the arbitrator. The first was incorporated into a PCB. There appeared to be problems with this board so some slight modifications were made on it before it was abandoned. The second board was a wire wrapped version. This too seemed to have problems. During the testing of both boards the Atari was reassembled to check that the faults were on the boards under test and that no damage had occurred to the Atari. It was during one of these routine checks that the Atari demonstrated the same behaviour. The problem lay in the chip sockets that housed the Atari's custom chips. The board had been flexed during the tests of the circuits and the custom chips had worked loose. When the modification boards had been removed the Atari board had relaxed and the sockets made contact with the chips. Flexing the Atari board gave the same problem, the sockets did not make contact with the chips but flexed the other way, contact was made. The Atari was repaired by the supplier and the modification board ran without any problems, however after the Atari board had suffered some more flexing the problems re–occurred.

The wire wrapped board had an additional problem of noise produced by the TTL buffer chips and the wrap itself. Each chip on the board was decoupled by a 0.1 uF ceramic disc capacitor across the supply rails close to the device. The problem was most evident during disk formatting although the disc power supply had a filter to remove transient spikes. It was found that the operation of the modification board was aided by the removal of one set of data buffers. It was proposed that the board be rebuilt using minimal buffering on the Atari side of the circuit and full buffering on the external side to aid cable drive and reduce damage due to misuse. This would reduce the noise created by the buffer chips and also the power supply loading. The technology of the buffers could also be changed from TTL 74LS' series to TTL 74F', this would further reduce the supply loading and also reduce delays caused by the addition of the buffers. The reduction of the supply requirements would mean that the Atari's supply would be able to supply the power to the whole system whereas a heavier loading would increase the load over the 1 amp rating of the bridge rectifier in the Atari power supply.

At this stage assessment of alternate systems were considered and the development of the interface ceased in favour of the alternative systems.

5.2 G96 68010 Transputer Interface.

After abandoning the Atari development, work was directed to the creation of a transputer based system. The first idea tried incorporated a 68000, T414 32-bit transputer and an A100 digital signal processor. The printed circuit board for this was too complex for the pcb packages available within the Polytechnic and so this too was abandoned.

The next alternative based on the same lines, involved the interfacing of a separate transputer board and 68000 card, the GESMPU-14. The interface would provide the 68000 card with 64K of static RAM, four Inmos link adaptors to allow easier connection with other transputer systems, an IEEE-488 industrial standard instrument interface and a dual-port RAM for communication to the transputer system.

The main logic of the interface was contained within an EPLD. This device was programmed to decode the address bus of the 68010 and produce the necessary chip enables for all the interface devices. Logic on the card was provided to allow the link adaptors to generate and interrupt and the actual interrupting device requests were priority encoded and made available to a port readable by the 68010. This readable port would allow the link adaptors to generate **autovectored interrupts**. Acknowledgments were generated within the EPLD, and it controlled the buffers onboard the interface card.

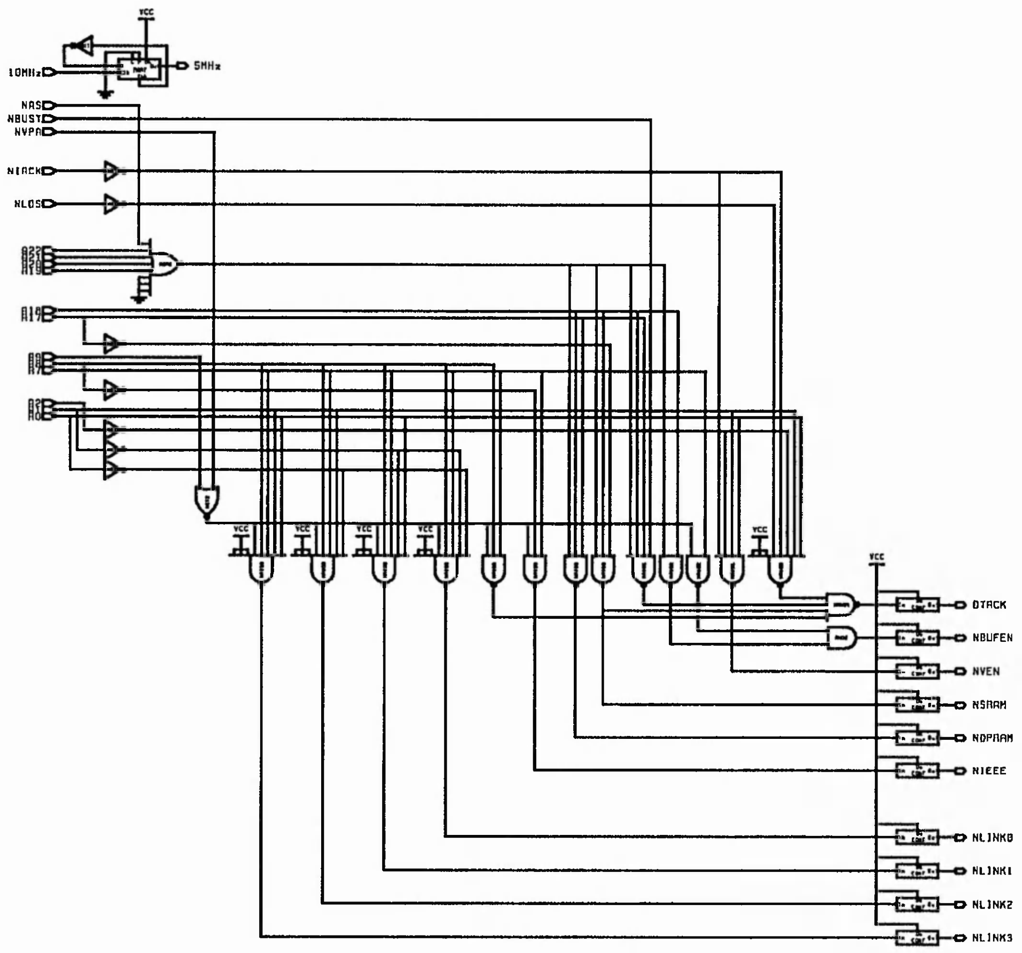

Figure 12 68010 GESMPU14 Interface EPLD.

Figure 12 shows the internal logic of the EPLD. The signals **NLINK**$_n$ are the chip selects for the CO12 link adaptors [33]. **NIEEE, NDPRAM**, and

NSRAM are the chip selects for the IEEE–488 interface logic, the dual–port RAM, and static RAMs respectively. **DTACK** is the data acknowledge strobe for the 68010 and is connected to the processor via an open–collector inverter. **NBUFEN** is used to enable the tri–state data buffers on card, while **NVEN** is an autovector interrupt enable for the GESMPU14 card. **NIACK** also forms part of the GESPAC interrupt system and is supplied by the GESMPU14. **NBUSY** is a busy signal generated by the dual–port RAM when there is contention involved with an access made to a particular location within it. The dual–port RAM was also connected to the interrupt system of the GESMPU14 as it can generate interrupts when the system connected to the other port of the RAM accesses special locations within it. The actual equations of the EPLD and its pin layout may be found in appendix F.

This work too stopped in favour of a full transputer based solution allowing the entire system software to be developed under one programming environment. By using just transputers one program could be written and debugged, covering all the processors in the network. The problems of processor synchronization and inter–processor communications being greatly reduced.

5.3 Atari to Inmos Link Converter.

When the work started on transputers, the only support for them at the Polytechnic came in the form of a development package comprising an '**Occam Programming System**', **OPS**, and '**Transputer Development System**', **TDS**, running Proto Occam on the Computing Services Vax. Later an IBM PC/AT was purchased by the Department of Electrical and Electronic Engineering running Occam 2 on a T800 based Inmos BOO4 within the PC, after previously borrowing an IBM PC/AT running Occam 2 on a T414 based BOO4, from the '**Transputer Initiative**' loan pool run jointly by SERC/DTI.

Using the Vax OPS system, programs could be written in Occam and run on the Vax with the Vax simulating the multi–transputer arrays. To use the actual transputer based systems a means of data transfer between the Vax and the transputer boards is needed. There is also a need for a means of operator input and visual output. To accomplish these goals the original Atari 1040 ST computer which underwent major changes in the 68000 development phase of the project was used as a terminal connected to the Vax system and to the target transputer system. The idea behind this being that the programs were written under OPS and then passed to TDS, recompiled and extracted. The extracted code could then be transferred from the Vax to the Atari's disc unit and later from the disc to the actual transputers.

Software was written for the Atari so it could communicate with the target system in several different ways. For initial tests of transputers systems the Atari could be used to '**Peek**' and '**Poke**' the transputer after reset [33]. The actual **peek** and **poke** was incorporated in '**memory fill**' and '**memory examine**' routines. A third routine for memory search was also included so that after the transputer had run a routine, if it failed to produce a visible result, then the memory could be searched and the final value found. Looking at the memory using **memory examine** it is quite easy to disassemble the raw transputer code and find where the workspaces are placed and confirm if the addresses obtained by the search for data actually puts the data where it should be, if not, then the final value required was not generated. An example of a **peek** cycle and the interpretation of it, may be found in appendix E.

5.3.1 Circuit Description.

The interface card for the Atari is based around the CO11 Inmos link adaptor operating in mode 1, [33]. In this mode it has a separate input and output port. The link adaptor is connected to the Atari via the ROM cartridge socket. Because a ROM is a read–only device the Atari detects if an attempt is made to write to the ROM and causes a bus error which suspends the operation of the computer. With the special access routine written, the bus error interrupt vector could be reprogrammed to point to a routine which would override the bus error if it occurred during the special access routine. The problem with this was that the bus error suspended the access before the write is completed.

To overcome this the address bus is used to contain the data when a write operation is required. This is accomplished by using the two 68000 data strobes, the upper data strobe, **UDS**, and the lower data strobe, **LDS** to indicate whether the operation is either a read or a write. When UDS is active, the operation is deemed to be a write and the address bus contains the data as well as the port address. With LDS active the operation is a read and the address bus only contains the port address and the port outputs data onto the lower 8–bits of the data bus. Figure 13 shows the arrangement of the address and data bus with the link adaptor. The signals **NREAD** and **NWRITE** are created by the decoding of the address bus and UDS and LDS, not shown.

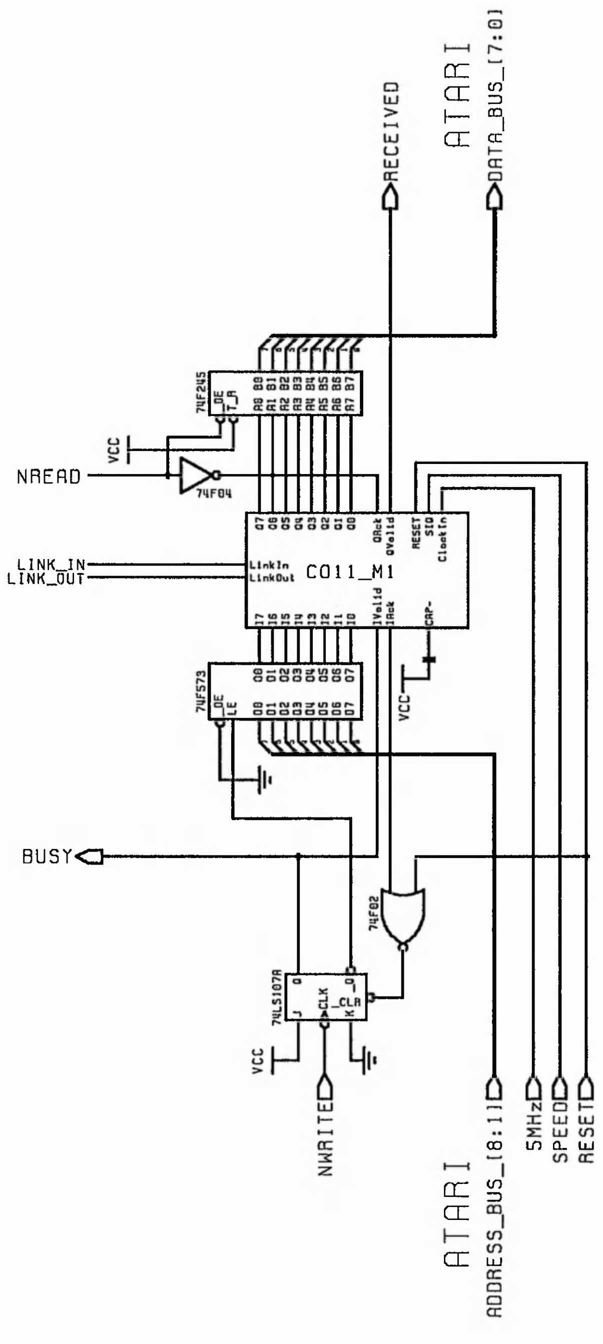

Figure 13 Link Module.

The signals **BUSY** and **RECEIVED** form part of the status register which releases data onto the lower 8–bits of the address bus when read in the way previously described.

A write operation to the port causes **NWRITE** to trigger the J–K flip–flop and set the **BUSY** line high. The rising of the **BUSY** line causes latching of the data by an 8–bit transparent latch, (the 74F573 in figure 13), which until then has been letting the address bus flow into the inputs of the link adaptor. **BUSY** is also connected to the adaptors **IVALID** line indicating that data is valid. The data is now transmitted down the link and when this is acknowledged by the receiving unit the J–K is cleared and **BUSY** returns to an inactive state.

When the link adaptor receives a byte of data, **QVALID**, and hence **RECEIVED**, go high indicating that data has been received. A read from the port causes **NREAD** to pulse **QACK** on the adaptor and also causes the release of data onto the data bus. The adaptor now acknowledges receipt of the byte.

The addressing format used is:

[1111|1010|pppx|xxxd|dddd|dddr] in binary

where r = 1 for read 0 for write
d = data (if any)
p = port number
x = don't care in this application, could be used for extending the port number.

When a read from address $FAnnnn is performed, where **nnnn** forms an even number, an actual write operation is carried out and the lower 8-bits of the address bus are latched by the device and used as data, (the least significant bit of the address generates the strobes and is not included in the address bus).

When a read from address $FAnnnn is performed, where **nnnn** forms an odd number, a read is carried out and the addressed device places data on the lower 8-bits of the data bus.

General Example.

If data $EA hex is required to be written to device 5 then a read from address $FAA1D4 is done.

If data is to be read from device 5 then it is read from address $FAAxxn.

Where xx = don't care and n = 1,3,5,7,9,B,D or F

The interface board has two Inmos links and one status register. The devices are mapped in memory as follows:

Link 0 read based at $FA0001, write based at $FA0000
Link 1 read based at $FA2001, write based at $FA2000
Status register at $FA4000

The status register is read only and hence only one base address. Link 0 is a 10 Mbps data link while link 1 is 20 Mbps. The different speeds are created by connecting the signal SPEED in figure 13 to VCC for 10 Mbps and 5 MHz for 20 Mbps.

Programming Example.

```
LEA $FA0000,A0              Point to link 0.
          .
          .
ADD.W D0,D0                 Double the value of the data to be written.
MOVE.W 0(A0,D0.W),D0        Write the data by reading a false value.
          .
          .
MOVE.W 1(A0),D0             Read data from link 0.
          .
          .
```

The programming example shows a general read and write operation for the link adaptor interface.

Note:– In the example above, no checks were made with the status register to check for the arrival of data or the completion of the previous transmit. This should be done in a real situation.

5.4 Printed Circuit Board Production.

Due to the nature of the circuits designed for this project, (large numbers of high speed signals and a high device count with high density packaging to reduce board area), printed circuit boards have been used in development rather than conventional prototype boards consisting of either wire–wrap or other point to point wiring systems.

Three pcb packages were used during this project, SMARTWORK, RGARPH and Mentor Graphics BOARD STATION. The latter two were purchased in the course of this project and considerable time was spent on familiarization of the packages. Since the packages themselves were new each of them had its own problems.

The production of the printed circuit boards has been a time consuming task added to by the fact that the pcb packages were new and had hidden problems which were only found by using them. More details on the pcb packages and the problems encountered in producing pcb can be found in appendix A.

6. The Data Acquisition Unit.

The Data Acquisition Unit is probably the most important part of the whole flow system after the sensor unit, for without accurate signal conversion the results of the system will be unreliable irrespective of the amount of data processing undertaken.

There have been several full designs of this processing level, including fully self-contained discrete versions offering high speed, 380 ns, and microprocessor based units. In each case the acquisition unit is designed so that there is one A/D converter and gain unit and the sensor signals are switched to it using analogue multiplexers. The output signal from this final multiplexer is effectively modulated by a 0.3 MHz square wave, this being the inter-signal sampling rate. To reduce any slew rate effects of the multiplexer itself, the actual signal is attenuated before rather than after the multiplexer, thus reducing the size of the signal to be transferred.

All the acquisition units try to produce the largest value out of the analogue to digital converter, which is actually an 8-bit high speed '**Flash Converter**', by increasing the gain of the circuit feeding it. This effectively increases the range of the flash converter but not the resolution which remains at 8-bits. This converter is now an adaptive quantizer which uses the assumption that for any given sample of a signal, the following sample will be of a similar value to the previous sample.

The actual flash converter has its own sample and hold circuit and when operated, it samples the signal and passes the voltage to 256 voltage comparitors which compare it against voltage taps in a 256 resistor,

resistor chain. Next there is a 256 to 8 line converter which encodes the result of all the comparisons.

The design of the unit is such that it could be used for a wider range of applications other than particle flow measurements.

6.1 Discrete Versions.

The Data Acquisition Unit was originally designed using discrete circuit elements. The two versions developed were based on the assumption that between any two samples of the same signal there will be a high correlation between the two points. The principle of the circuit is that it takes a sample at a predetermined gain. The circuit now uses the sample value obtained and the gain used to determine the gain to be used for the next sample of that signal. This is a one–word memory adaptive quantizer, increasing or decreasing the input gain of the circuit depending on the signal.

There were two basic designs using this approach. The first was based on the total completion of one conditioning cycle before another started and had a cycle time of 480 ns. The second used '**cycle overlap**' to increase speed which gave a cycle time of 300 ns. Both circuits effectively track the signal and every sample taken causes a change in gain by a factor of 2, if needed.

6.1.1 Operation of Discrete Circuit Version 1.

The start of the cycle causes the output of a 4-bit counter to be latched by a 4-bit latch. The latched output selects the signal of interest by feeding analogue multiplexers. The count also forms an address for the required gain which is fetched from a high speed RAM. The output of the RAM feeds another analogue multiplexer which selects the feedback resistor in the actual gain circuit. The RAM output also feeds a 4-bit up/down counter which is used to calculate the next gain for that signal, and two 8Kx8 EPROMs which give the final formatted answer. After a delay of 80 ns the analogue signal is allowed into the gain circuit, originally the input to this section is held at ground potential while the gain is selected to avoid any stray signals causing the amplifier to overload. The 8-bit flash converter is now activated. The output of this goes to the EPROMs which form the actual result required and two 8-bit comparitors via an absolute value approximator. Using a true absolute value generator was considered, this giving a true two's complement if the signal value is negative, but for speed the approximation of just a one's complement on negative values was considered to be sufficient for this purpose. The comparitors decide if the 4-bit counter generating the gain should count up, down, or not at all. A pulse is now generated to cause the counter to calculate the next gain to be used on that signal. The result of this is stored in the high speed RAM and the cycle is finished.

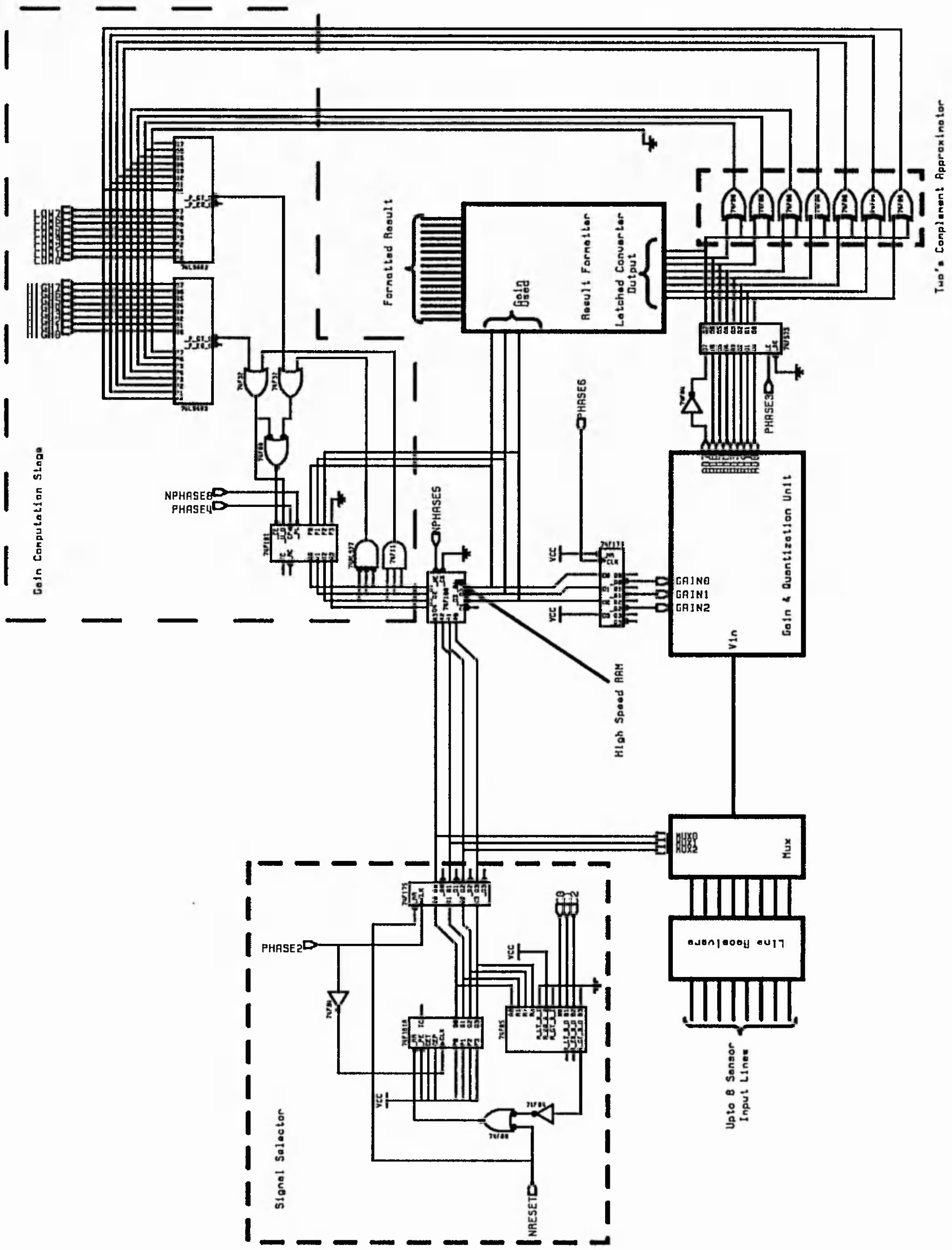

Figure 14 Discrete Quantization Circuit Version 1.

Figure 14 shows the form of this first circuit. The EPROMs are hidden inside the block called the '**Result Formatter**'. The detail of the '**Gain and Quantization Unit**' has not been shown for simplification. This detail includes the resistor selector and the actual A/D. The start of the cycle is indicated by a positive going pulse on **PHASE2** in the diagram.

6.1.2 Operation of Discrete Circuit Version 2.

The principle of the second circuit is similar to that of the first, the only difference being that the calculation of the next gain to be used on that signal, is done in parallel with the conditioning for the next signal.

In this circuit, figure 15 showing the basic form, the end of the previous cycle starts the next cycle. The output of the 4–bit counter is latched. This selects the signal of interest and also forms the address for the required gain. The gain is fetched from a high speed RAM. The output of the RAM feeds the resistor selector and also the 4–bit up/down counter which is used to calculate the next gain for that signal. Here the RAM output also feeds another 4–bit latch which in turn feeds the two 8Kx8 EPROMs. Again the signal now passes through the gain and A/D conversion stage. During this time the RAM was idle in the first circuit, here though it is used for the storage of the gain calculated by the 4–bit up/down counter. The up/down counter has the old gain value latched into its internal flip–flops 20 ns after the start of the cycle. This gain value is held by the 4–bit latch which feeds the EPROMs contained in the **Results Formatter**, (see figure 15 page 75). The old output from the A/D converter has been latched in an 8–bit latch which feeds the two 8–bit comparitors, via the absolute value approximator. A pulse is now generated to cause the counter to calculate the next gain to be used on that signal. The result of this is stored in the high speed RAM. Whilst all this has been happening the actual signal has not yet reached the A/D converter. The address of the gain to be used is now changed back to its original value ready for the end of this cycle. The conversion takes place. The result of the A/D conversion is latched in the 8–bit latch and the gain used is latched into a 4–bit latch.

Note in figure 15:–

1. Again for simplification certain elements of the circuit have been shown as a block.

2. In appendix D there is a third version of this type of circuit which uses a different new form of gain unit. This circuit gives a cycle time of 260 ns, and the appendix presents a simulation of the circuit using QUICKSIM.

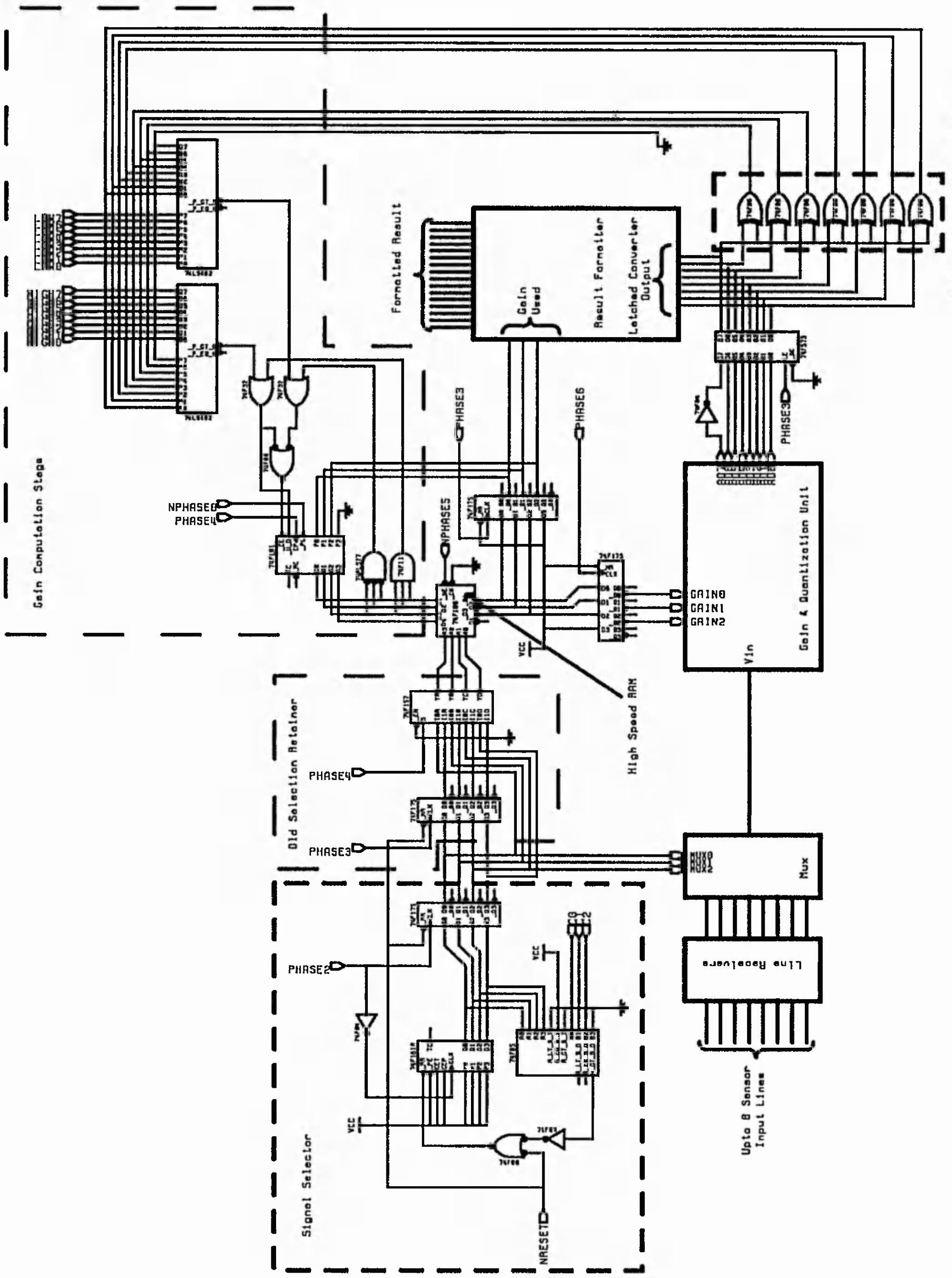

Figure 15 Discrete Quantization Circuit Version 2.

6.2 Transputer Based Acquisition Units.

The transputer based acquisition units form the heart of the flow system. There are two designs of transputer controlled Data Acquisition Units. The first unit was based on the discrete versions except here the first sample was always at unity gain and then the system adapted to the result from that for that sample only. The next design broke from the '**adaptation for each sample**' mode and used adaptation after a complete block of samples based on the values contained within the block. The first unit was constructed and tested. The second unit evolved from the first and actually utilized the circuit board of the first.

The actual gain sections of the transputer units differ from the discrete versions, the main difference being that they comprise three stages to give the required gain rather than just one. Each stage has a fixed feedback resistor and the gains can be unified by switching in a parallel resistor in the feedback circuit.

6.2.1 Transputer Based Acquisition Unit Version 1.

This version of the acquisition unit can be regarded as a one word memory quantizer which stores the previous value quantized and uses it to adapt to the signal by choosing a multiplier to multiply the current gain by, and the process continues. This is not actually used as a true one word memory quantizer as only two samples of the signal are taken, the first is at unity gain hence the multiplier chosen is in fact the next gain to be used.

6.2.1.1 Circuit Operation.

The signals from the analogue system interface, which comprise the actual interface to the sensor head, arrive at the transputer board via a system multiplexer, allowing one transputer system to control many flow systems, and are attenuated to ±1.28 volts before going into the final multiplexer, as per all the other designs. Next comes the transputer controlled gain circuit.

The A/D converter is connected to the transputer via an erasable programmable logic device, EPLD, which controls the whole hardware signal conditioning process, including the signal multiplexer, by access from the transputer. In normal sampling conditions the EPLD is clocked by the transputer so that the process cycle is:

1. Unity gain and the signal to be converted are selected.

2. The signal is quantized and the result is stored in the EPLD and the gain to be used is calculated.

3. The transputer now reads the value of the gain that is to be used, in the form of **N** where 2^N is the actual gain.

4. Now the transputer reads from the A/D via the EPLD. At the end of this read the EPLD returns to process state 1.

At any time the transputer may reset the EPLD so that unity gain and signal zero is selected.

6.2.1.1.1 Gain Calculation.

The calculation of the gain is quite simple. The incoming signal, which is reduced from the transmission level of ±10 volts maximum to ±1.28 volts maximum, has half the full scale range of the A/D, 1.28 volts, added to it as the A/D can only handle positive voltages. The signal now has a range of 0 to +2.56 volts. This is now quantized by the A/D.

The effect of this process is that for an input of 0 volts the output of the A/D is 128 decimal and any signal less than 0 volts is less than 128 decimal and any signal above 0 volts is also above 128. So now in binary any positive voltage will have bit 7, (the most significant bit), set and any negative number will have a zero. For the gain calculation the absolute value of the signal is used.

Rather than taking the two's complement of a negative number to give its true absolute value, the one's complement is taken instead reducing the amount of circuitry needed, and the time taken, for the calculation of the absolute value. Due to this the absolute value is obtained by using the inverse of the most significant bit of the converted signal to control the inversion of the remaining bits by an exclusive OR process. So if the number is negative, the inverse of bit 7 will be one, thus causing the other bits in the word to be inverted.

The absolute value is passed to a 7 to 3 line priority encoder which produces the required gain as a value **N** which when fed to the gain circuit produces a gain of 2^N. The encoder has inverted outputs and so **N** is in fact '**8 – the highest bit number set**'. The following table shows the encoder output, **N**, and the gain it represents for different input values:

Input	N	Gain
1xxxxxx	0	1
01xxxxx	1	2
001xxxx	2	4
0001xxx	3	8
00001xx	4	16
000001x	5	32
0000001	6	64
0000000	7	128

where x indicates a don't care state

TABLE 2 Gain Encoder Output.

The operation of the circuit was simulated using Mentor Graphics **QUICKSIM** before being programmed into the erasable programmable logic array. Figure 16 shows a block diagram of the EPLD containing the gain computation elements and signal multiplexer control.

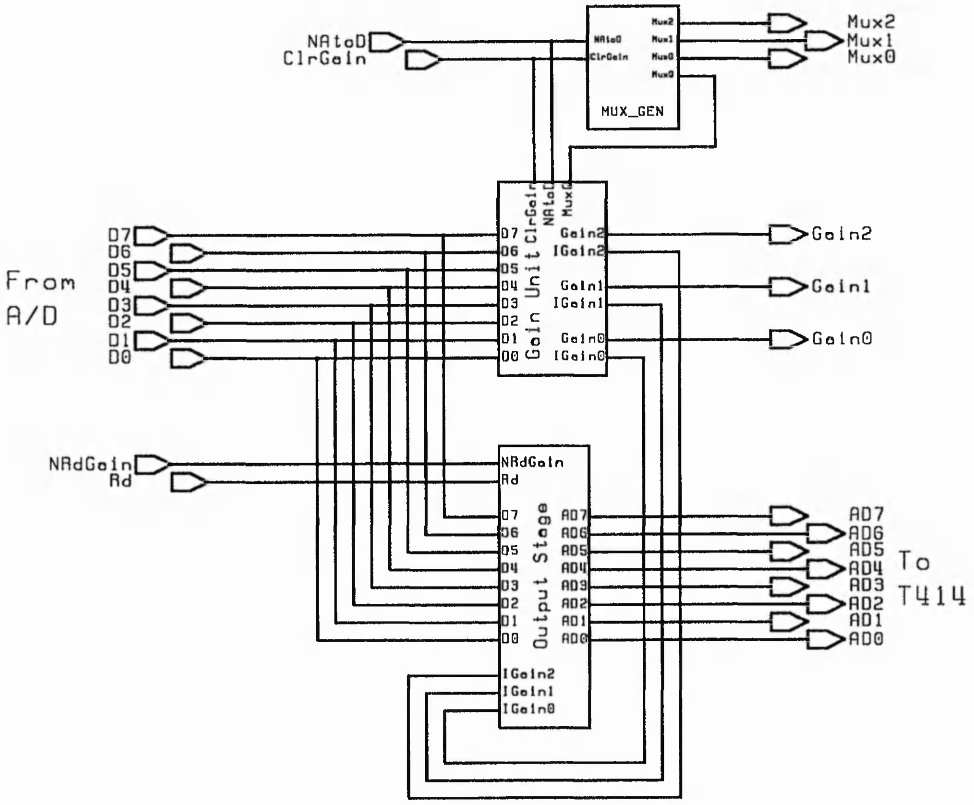

Figure 16 Block Diagram of the EPLD.

The EPLD is closely linked to the flash converter. Signal **NAtoD** is used as the flash converter's clock to cause it to convert the analogue input to a digital representation. The gain calculation unit within the EPLD also uses this signal to control the operation of the gain section. When the system is running in its normal mode, where adaptation is required, the gain calculation stage monitors the **NAtoD** line and switches between unity gain settings and calculated gain. After a unity gain sample the EPLD calculates the gain needed to create the largest output of the A/D and latches this in a gain latch. This latch feeds the internal circuitry which in turn feeds the gain selection pins of the analogue gain section and the output stage of the EPLD which is interfaced to the transputer. The gain calculated can be read by the transputer and so can the actual A/D output. This switching is controlled by the **NRdGain** and **NRd** signals. The calculated gain can be forced to unity at any time by the application of **ClrGain** signal. **MuxQ** is the internal count of the **NAtoD** signal and controls internal switching of the gain outputs between unity and the calculated gain. Figure 17 shows the gain unit with its calculation stage followed by a controlled gain latch.

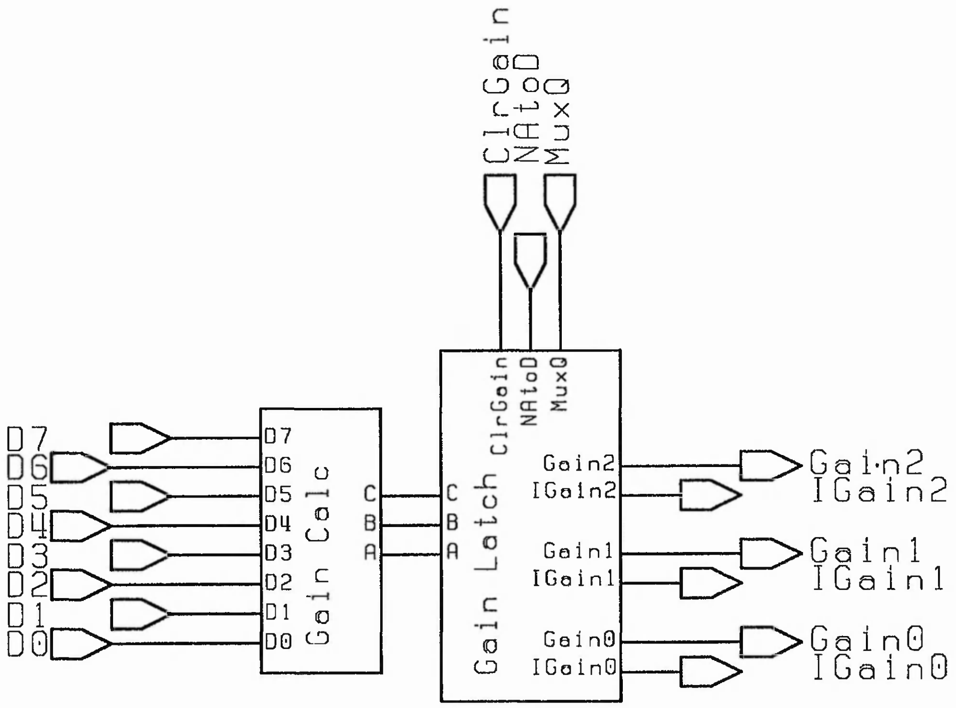

Figure 17 Block Diagram of the Gain Unit within the EPLD.

Inside the gain calculation stage the A/D's output is transformed to a gain request by the process previously discussed. The absolute value approximator can clearly be seen on the left-hand side of figure 18. The rest of the logic forms the 7 to 3 line priority encoder.

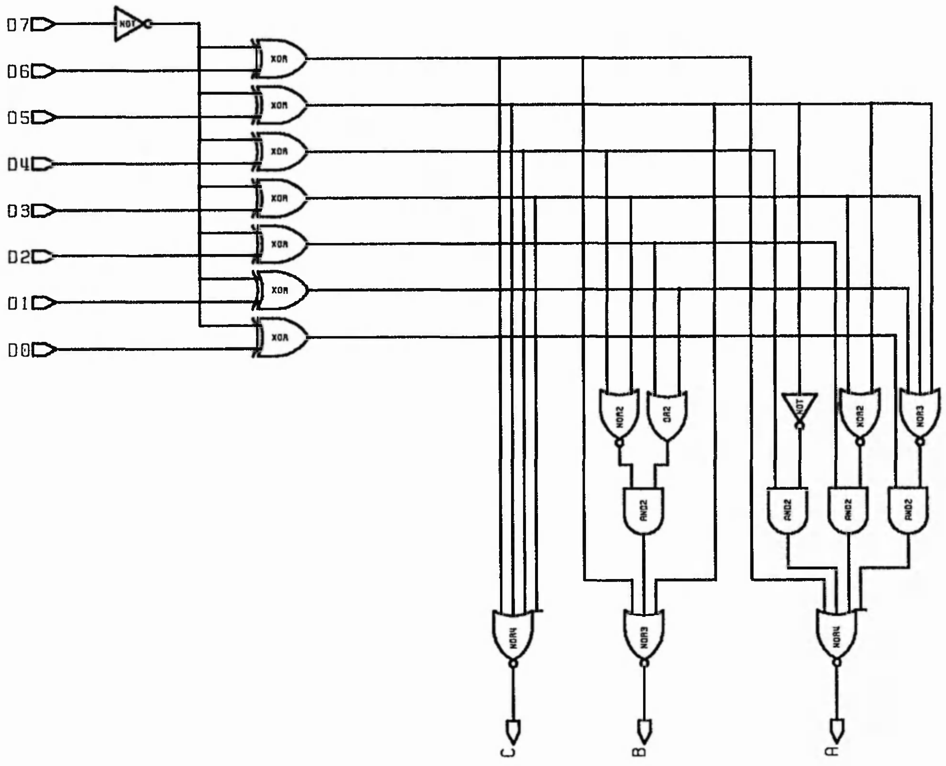

Figure 18 Gain Calculation Stage. Logic Level.

The actual gain elements are wide band op-amps which have fixed gains of 16, 4 and 2. Each op-amp has another feedback resistor which is connected in parallel with the first via a switch. When the switch is closed, by a logic one on its control input, it inserts the additional feedback resistor which changes the amplifier gain to unity. The switches are arranged so that the MSB of **N** controls the gain of the 16 times amplifier, while the LSB controls the gain of the times 2 amplifier. Figure 19 shows the gain section of the circuit with its three amplifiers with switched parallel feedback resistors. The signal from the multiplexer goes in to the gain section, seen at the left hand side of the figure, the gain is used and the signal goes off to the conversion section, figure 20.

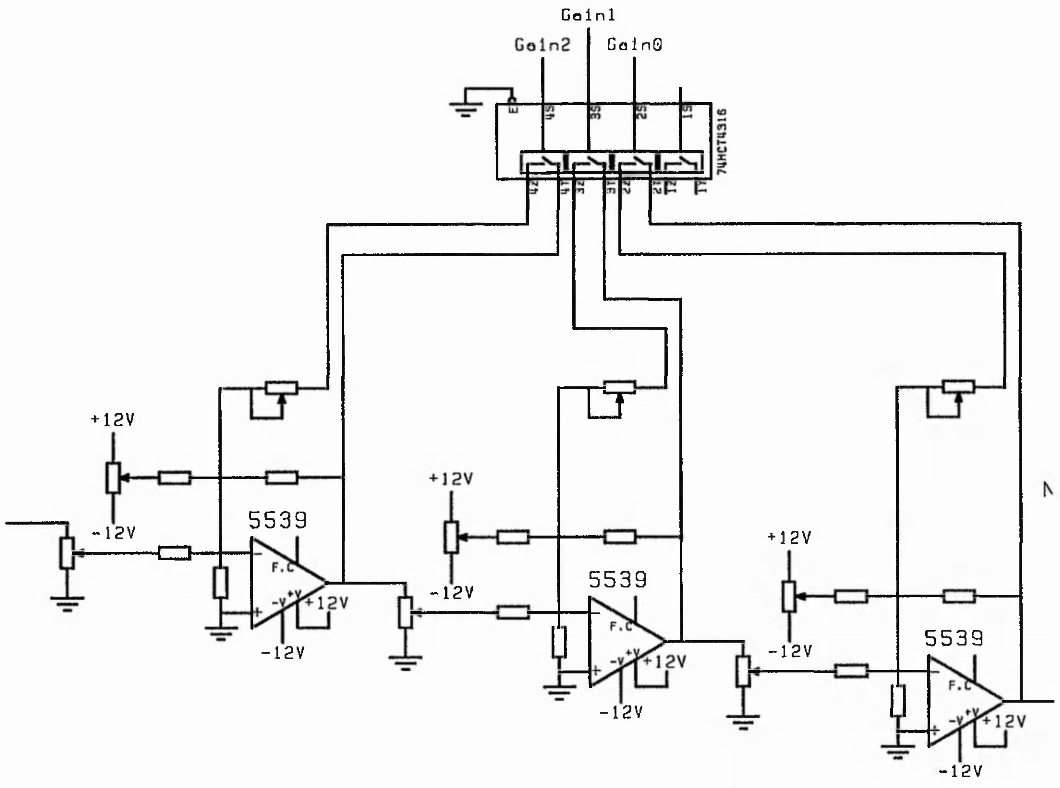

Figure 19 Gain Section.

The offset voltage for the A/D input is added after the gain circuit so that it is not multiplied by the selected gain. Figure 20 shows the flash converter linked to the EPLD. The signal from the gain section comes into the circuit at the top left. It goes through the op–amp which adds in the half full–scale offset to the signal before it reaches the converter. The '**Auto Gain PLA**' is the actual EPLD and is closely linked to the converter to reduce noise from being induced in the lines between the converter and the EPLD.

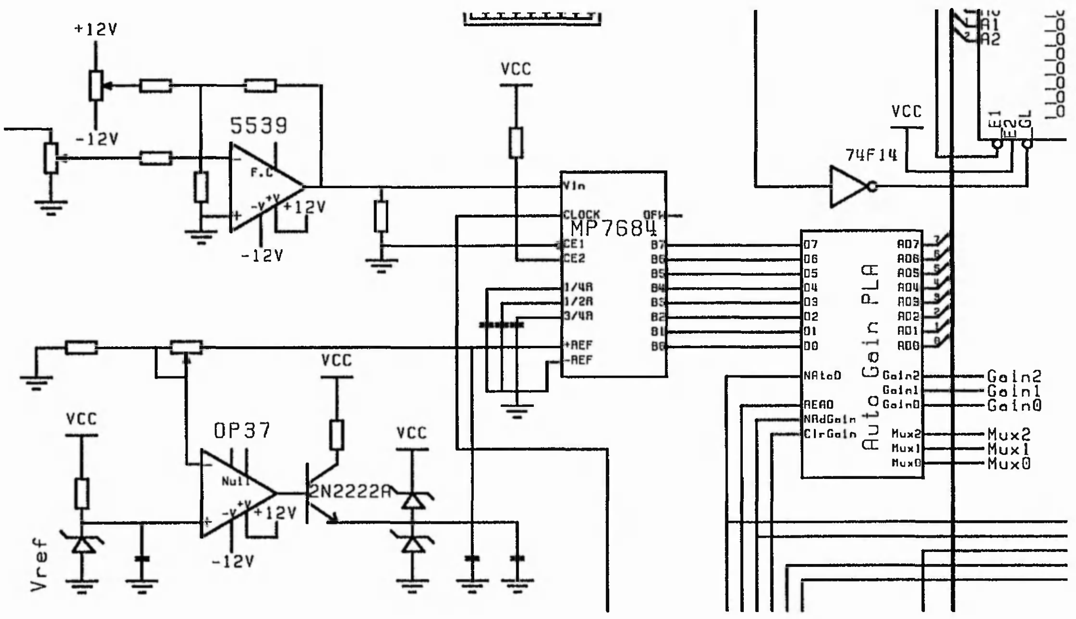

Figure 20 Flash Converter Linked to the EPLD.

What follows is a look at how the software in the system, operates the data acquisition system.

6.2.1.1.2 Software Control.

The transputer has the job of detecting the actual injection. It is signalled when the H.T. supply is turned on, and given the number of samples to be taken. It now starts hunting for the injection.

During the hunt for the injection, the transputer integrates the signal I_0 and tracks it using coarse quantization to find the peak due to the capacitive air effect. When the peak is found it now tracks the voltage as it drops due to discharge. When the sample just read is greater than the previous one then this point is taken as the injection point. During the search phase, the number of samples allowed in the system cycle is reduced for each sample taken. If injection is not found by the time half the number of samples are used then the transputer stops searching and informs the Processing Array and the Controller that there will be no data coming from this cycle. If, however, injection is found then the transputer transmits to the processing element which deals with the injection current, the partial integral obtained so far. The transputer then broadcasts to the whole Processing Array how many samples are to follow, this number is the number left after the search for injection. During first phase of a cycle the transputer resets the EPLD in order to keep it at unity gain and signal I_0. This speeds up the process as the remaining signals should hold no useful information until after injection has been reached.

Once injection has been established the second phase starts and the operation of the transputer changes to one of data formatter and distributor. Now it takes each sample in turn and converts the gain used along with the reading obtained, into a 32-bit data word. This word is now transmitted along one of the Inmos links to further transputers in the Processing Array which would do all the processing needed between samples.

This method of data transference increases system speed and hence the number of samples available in a given amount of time. If all four signals were processed on one transputer then it could take up to four times the amount of time to process samples from all 4 signals as compared to four transputers each processing one sample of a particular signal. Using different links to transmit each sample to a different transputer means that there is no bottle neck on any one link as each link can transmit independently. The system could be reconfigured so that 2 or even all 4 of the links are connected to one transputer. If all 4 were connected to one transputer then there would be no links to connect other devices to the system, but more importantly it would be effectively like just the first transputer doing the sampling and the processing.

6.2.2 Transputer Based Acquisition Unit Version 2.

The second version of the transputer based acquisition unit arose from the first due to operating difficulties. Although the idea of the circuit was valid, the actual implementation led to offsets being introduced when the gain was switched. These offsets made the actual switching unreliable, and so for speed the operation of the circuit was modified. The EPLD was reprogrammed so that the gain adaptation was calculated by the controlling transputer in the system network after one complete cycle of the system, based on the readings obtained. The Controller had the ability to calibrate the whole analogue subsystem and justify the results obtained to take into consideration the actual circuit gain used, (as calculated during calibration), and the actual offset (as measured during the calibration phase). The operation of the second version has actually reduced the data bandwidth requirements and so increased the data throughput of the system.

Several modifications to the gain circuit were proposed which would allow the system to return to the original full adaptation mode. The final circuit modification is given below:

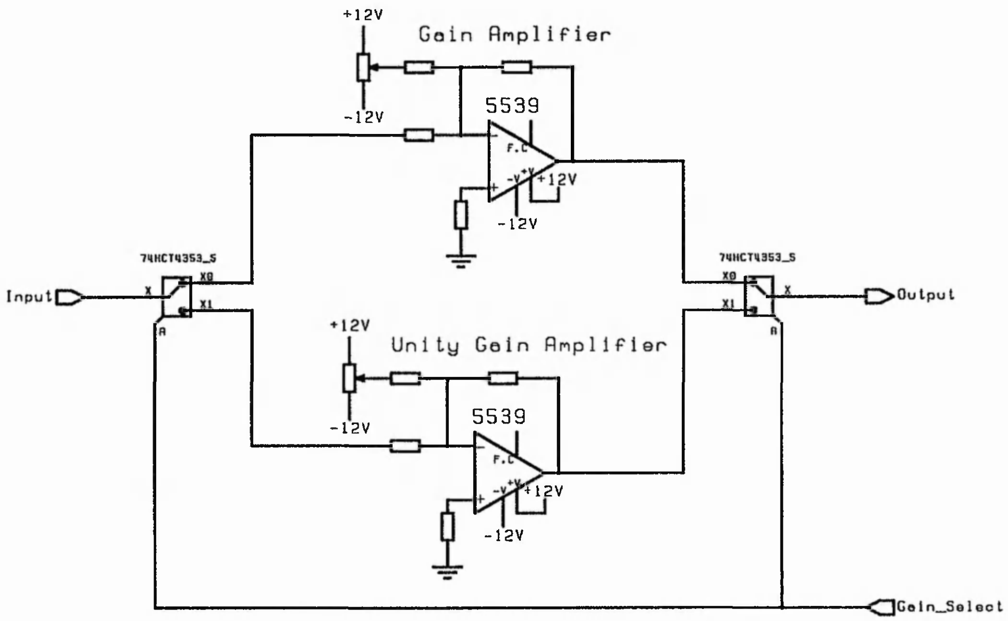

Figure 21 Gain Selection Stage.

This circuit element will easily interface to the first version's gain control provided by the original EPLD. The circuit operation is simple, either the unity gain amplifier is selected or the true gain required is selected. Both paths offer signal buffering and the ability to adjust out any offsets present in the circuit at that point.

Note:- The amplifiers can not simply be a.c. coupled as this application requires d.c. measurement of the sensor readings.

For the actual second version the amplifier modifications, though desirable, were not included because the offsets of the system were eliminated by full circuit calibration by the transputer system. During the calibration phase the system used every channel in the last multiplexer with an input signal connected to zero volts. All the gain and channel combinations were used to create an array containing the channel offsets for different gains. By using this array method the assumption that '**each channel is identical therefore any one channel could be used for calibration**', was not used. This idea was extended to calibrate the system gains by using known accurate calibration signals. This created another array containing the true gain information, and using this array and the offset array, any value read could be converted into a true signal value.

The task of adaptation was given to the controlling node in the system. After each cycle of samples the Controller would decide, using the gain used and the maximum and minimum sample values, whether an increase or decrease in gain was required. It was no longer assumed that a change in gain would actually give a factor of two change in the signal, the actual prediction for the new maximum and minimum values used the true value of any new gain being considered, thus giving a better adaptation.

7. The Final System Hardware Design.

7.1 The Data Acquisition Unit.

The Data Acquisition Unit and analogue interface are very closely linked. Due to the design of the acquisition unit, where it is not dedicated to the sole task of flow control data acquisition, a further system interface is needed. Figure 22 shows a block diagram of the Data Acquisition Unit and its connection to several flow systems.

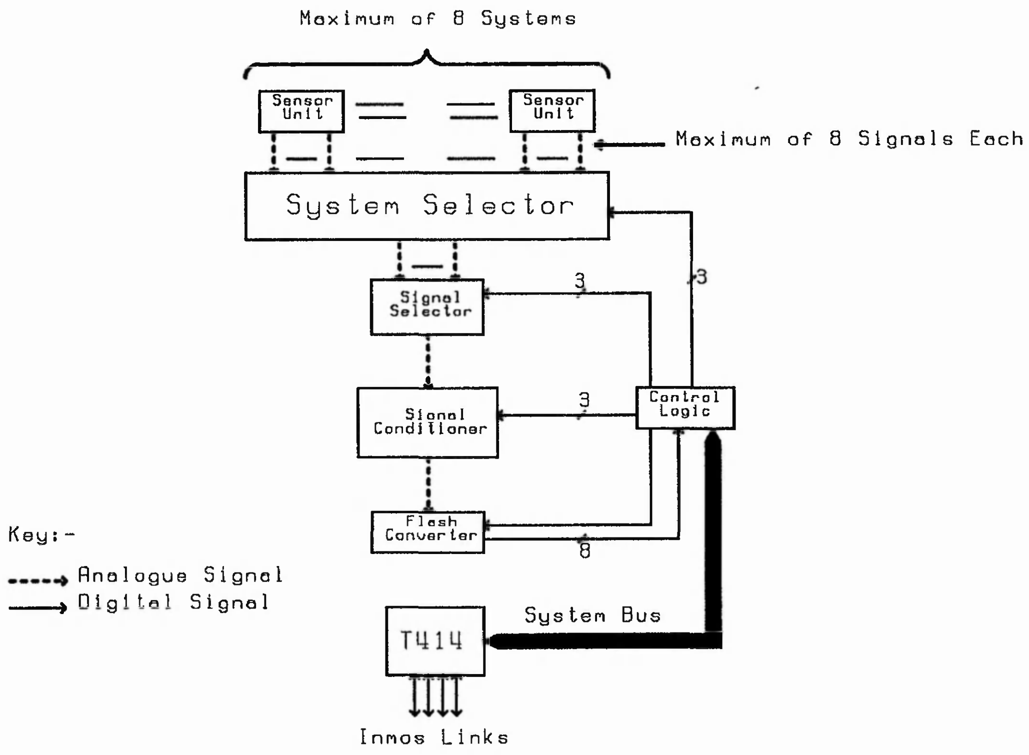

Figure 22 Block Diagram of the Data Acquisition Unit.

In Figure 22, the **Sensor Unit** refers to both the sensor head, (figure 23), and the sensor head interface units, (figure 24). The **System Selector** and the **Sensor Units** are not part of the actual circuit board of the Data Acquisition Unit, but form part of the analogue interface to the system.

What follows is a description of the analogue interface of the system and the operation of the transputer controlled Data Acquisition Unit. The section on the analogue interface will include the full signal path from the sensors to the actual transputer board.

7.1.1 Analogue Interface to the Sensors.

The analogue interface of the system starts at the sensors with current to voltage converters.

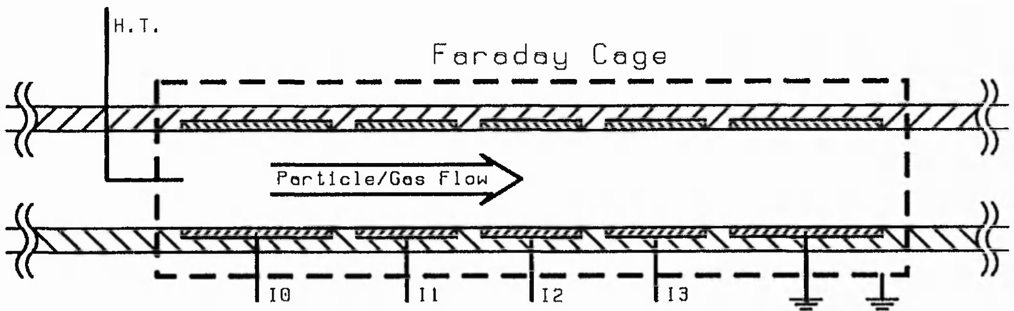

Figure 23 Sensor Head.

Figure 23 shows the sensor head. A high voltage supply controlled by the transputer system is connected to the injection needle. When the supply is pulsed a corona discharge is obtained and charge is injected onto the passing particles. Sensor current I_0 is proportional to the amount of charge injected into the flow. The sensors downstream measure the amount of charge carried by the flow. Currents I_0 to I_3 all pass through identical current to voltage converters as outlined in figure 24.

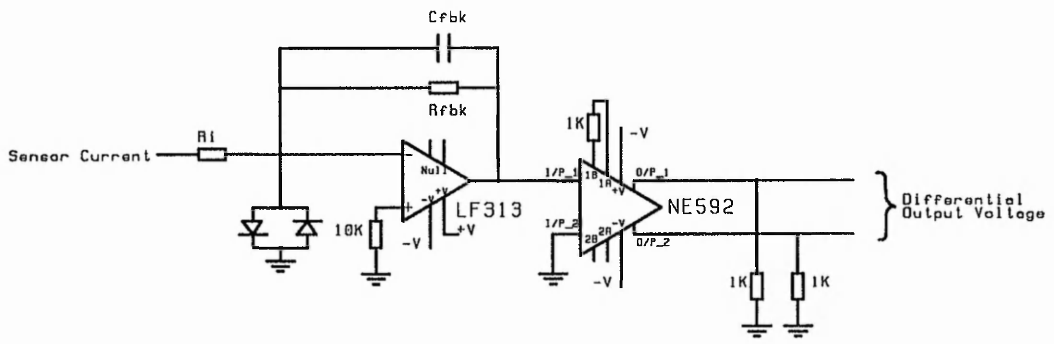

Figure 24 Current to Voltage Converter.

The output voltage is connected to differential receivers on the multiplexer board via twisted pair cables. The first stage multiplexers select one complete system, with a maximum of 8 signals, from 8 possible systems and is referred to as the **System Selector** in figure 22.

7.1.2 Transputer Interface.

The transputer interface itself is composed of both hardware and software. The software is needed to drive the hardware in the correct sequence. The analogue and digital sections of the hardware are described below, followed by a brief look at the software control.

7.1.2.1 Analogue and Digital Sections.

The signals from the **System Selector**, figure 22, are attenuated by 50% before going into the final multiplexer, the **Signal Selector**. The output signal from signal selector is effectively modulated by a 0.3 MHz, this being the interchannel switching rate. To reduce any slew rate effects of the multiplexer itself, the actual signal is attenuated before, rather than after, the multiplexer thus reducing the size of the signal to be transferred and hence any delay time needed to be added in order to ensure that this level is attained.

Next comes the transputer controlled gain circuit and the analogue to digital converter, the **Signal Conditioner** and **Flash Converter** in figure 22.

The flash converter is connected to the transputer via an erasable programmable logic device, EPLD, which controls the whole hardware signal conditioning process by accesses from the transputer. The EPLD is now only a simple latch for the gain information written by the transputer, and a tri-state buffer for the A/D converter, and the transputer writes a request for the next cycle and then reads the results of the previous request.

7.1.2.2 Software Control.

During the development of this unit the transputer software has had many more processes available than those needed to run the actual flow rate process. The processes include, user control over the multiplexers, H.T.

power supply switching, (this switching process was derived from one of the unused high order multiplexer signals and allowed pulsing of the H.T. supply with software controllable pulse widths), and many other process requests involving the acquisition unit and the distribution of commands to the Processing Array.

What follows is an overview of the process needed for the flow control. The actual way in which the software does the tasks is not included in the description.

The transputer on this board has two main jobs. The first is to detect the onset of injection, if it arrives at all, and then to change its operation to be a data distributor. The whole process is started by the controlling transputer signalling when the H.T. supply is turned on, and giving the number of samples to be taken, all via the Processing Array. Once this message is through, the transputer starts hunting for the injection.

During the hunt for the injection, the transputer integrates the signal I_0. During the search phase, the number of samples allowed in the system cycle is reduced for each sample taken. After the search the transputer computes the number of samples to follow, if any, based on the search time taken. It then passes the value of the number of samples to follow and the partial integration of I_0 to the Controller. If injection is not found or the injection signal is out of the range of the A/D then the process terminates at this point and the transputer goes back to waiting for commands. If, however, injection is found then the transputer starts the Processing Array processing by informing each node the number of samples that will follow.

During the search phase signal I_1 to I_3 are not examined as they should hold no useful information until after injection has been reached. This speeds up the effective time between sample points so that the injection current is checked with a higher time resolution than normal. By not transmitting the injection current information of the Processing Array for examination, all the processors in the array can run the same program.

If injection has been established, then the operation of the transputer changes to one of data formatter and distributor. Now it takes each signal in turn and requests its chosen gain, as determined by the Controller, and transmits the sample value obtained in the form of an 8-bit word along one of the Inmos links to further transputers in the Processing Array which will do all the processing needed between samples.

After a full cycle the Controller will perform any adaptation needed and send the acquisition unit a new request table with updated gains.

7.2 The Processing Array.

7.2.1 The Hardware.

The basis of the Processing Array is four T414 32-bit transputers connected together in a ring network. Each transputer can talk to its neighbour by communicating through link 1, the neighbour receives the transmission on link 2. Reverse network communication can also take place as the Inmos links are bi-directional.

The circuitry of the Processing Array has been optimized to suit this particular application. As all the processing is done in real-time, with efficient programming the T414 processors can be used without the need for any external memory. This gives the system two distinct benefits:

1. A reduction of both the circuitry and the board area.

2. Using only internal memory for programs and data, the whole program will run faster as access to external memory involves at least two extra wait states at its fastest level.

The circuit board has two off–board connection areas. The first area connects the array to the Data Acquisition Unit, previously described in section 7.1, and the second to the Controller described later in section 7.3.

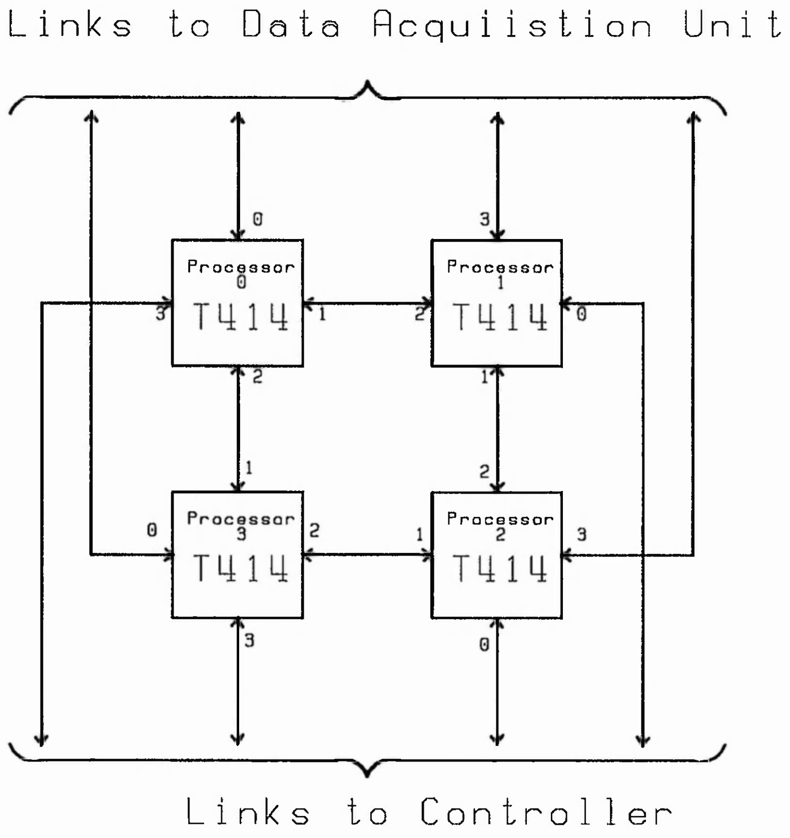

Figure 25 Link Arrangement of the Processing Array

Figure 25 shows the arrangement of the processors and their links within the Processing Array. The links were arranged in that particular order to ease board layout. The board layout was achieved using SMARTWORK.

7.2.2 The Software.

Each transputer within the Processing Array runs the same program, only the data differs. Parameters passed during the power-up phase of the system determine the operation of the array.

Facilities have been incorporated so that the links between the array and the Controller can be broken to allow extra transputer hardware to be inserted; during development of the system one link was broken to enable the IBM PC/AT Transputer Development System to be connected to the entire transputer processing structure via the free link on the Controller, to allow system development and software debugging.

7.3 The Controller.

7.3.1 The Hardware of the Controller.

The final version of the Controller used a multilayer printed circuit board. Four layers were for signal and two for power. The component definitions for BOARD STATION were modified to allow the stacking of components. The board had physically large static RAMs. To reduce board area the space beneath the RAM chip sockets was utilised by typically placing another logic device in it. Components were also laid under the second device, but these were for placement on the underside of the board. The largest example of this "**stacking**" had a RAM chip with its decoupling capacitor underneath it. There was also a buffer chip with its decoupling capacitor under that, plus a single in-line resistor pack in the remaining space beneath the buffer chip; (the decoupling capacitors and resistor pack were targeted for the track-side of the board).

The Controller carried over features from the first design, like the full expansion bus presented on an indirect edge connector. The megabyte of fast static RAM was split into two units which under normal conditions formed a continuous area of RAM. The memory decode and buffer control logic were put into one EPLD. The logic within it allowed for the movement of half of the RAM from the normal RAM map to the ROM area at the upper end of the memory map. Figure 26 shows a block diagram of the Controller. The EPLD is combined in the **Address Decoder and Board Controller** block in the diagram.

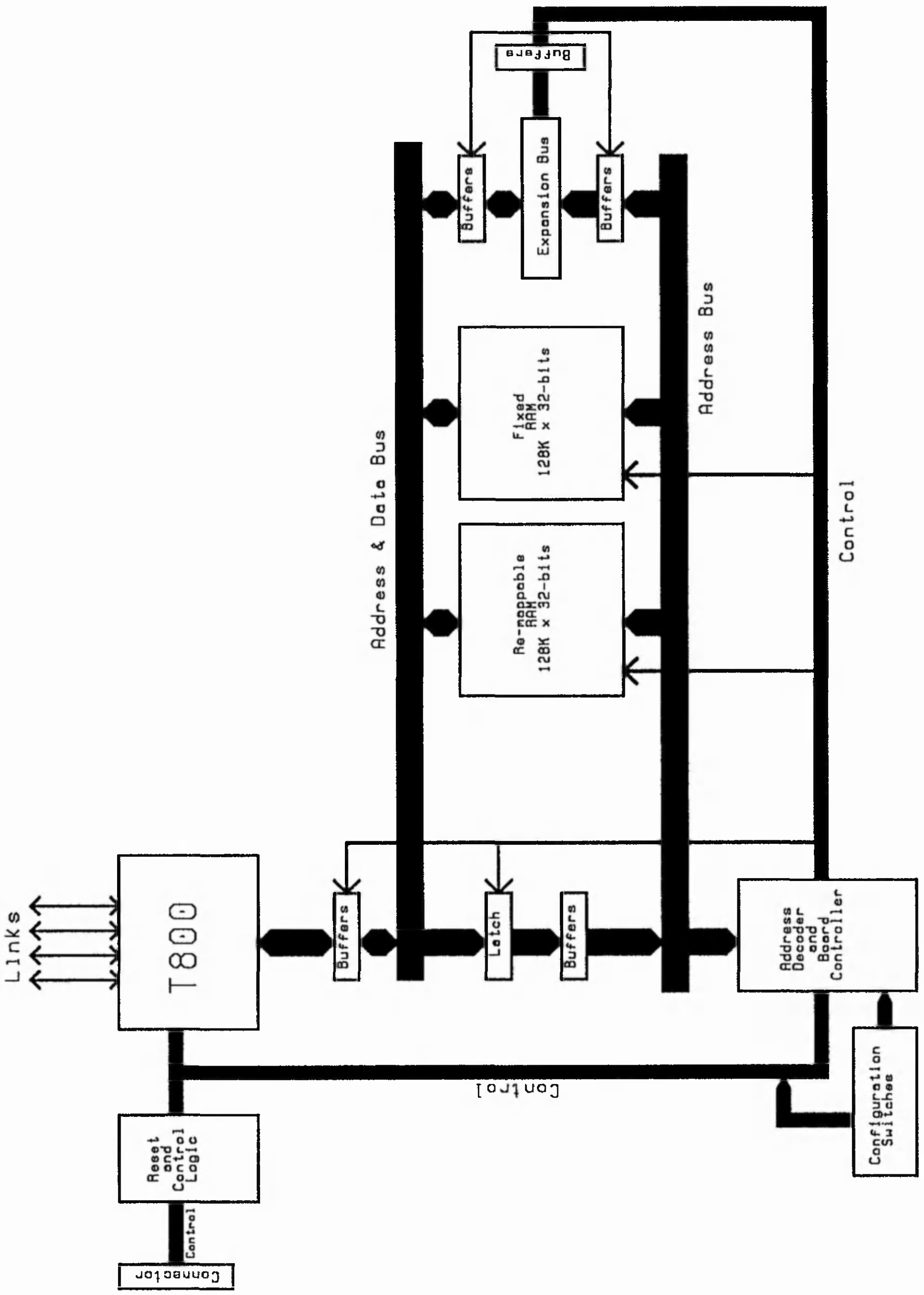

Figure 26 Block Diagram of the Controller.

The configuration of the memory into either all RAM or split RAM/ROM is setup by switches on the pcb. These switches are in a dual in–line package, and also control the speed of the transputer links.

The EPLD also contains logic which controls buffers on the board so that if an external device such as a DMA, took control of the transputer bus, then the buffers would now be under the control of the DMA.

In order to give better understanding of the Controller and its operation, a brief overview of its software follows.

7.3.2 Software Overview for the Controller.

This software is the largest of the external system's programs, some 3,000 lines of Occam code. The Controller forms the front line of the system deciding exactly how to execute the user requests.

Commands come into the Controller from two main directions. The first, in its current arrangement, comes from the BOO4 transputer card inside the IBM PC/AT. Commands from this source are decoded and either cause immediate action by the Controller or will be passed on to the Data Acquisition Unit after any translation necessary. Further details of the operation of the system can be found in section 8.3 dealing with the main system software.

The second command path comes from the upper system. These are normally in the form of structured print commands although other information can be transferred to the Controller in this way.

The Controller has a large memory capacity and only some 17K of this is used by the program code. Some 224K words of RAM of the 256K words, (1 mega byte), is set aside for the execution of the database information gathering command from the IBM. With this command the Controller takes full control of the system instructing the various elements as necessary to compile the required database information while adapting to the conditions it finds.

The Controller also instructs the system how to perform calibration (section 7.5), and here it computes the actual system parameter from its findings. Block data requests from the IBM PC causes data collection by the Controller and after all the data is in, the Controller will release it to the PC. In this manner the Controller has effectively increased the speed of operation of the block gathering commands by removing the bottle neck of the single 10 Mbps link between the PC and the external system and buffering the data from the upper system by the execution of internal parallel processes for link communications.

Another task of the Controller is to perform system adaptation based on the values obtained during processing of the previous data, further details on adaptation can be found in section 8.2. During the execution of either the single shot flow command or the execution of the database gathering, the Controller performs signal gain adaptation depending upon the results obtained from each cycle of the data processing.

7.4 <u>Pulsed High Voltage Power Supply.</u>

For this project a controllable high voltage power supply is needed. The supply must have variable voltage and pulse width. The power supply unit is based around the original power unit used by C.A.Willis. In the original supply unit, the required injection voltage was set-up using a potentiometer and the front panel meter which indicated the output voltage that would be generated. The injection voltage was switched on and off by an external pulse source connected to the trigger input of the supply unit. For a controlled system a new set-up and trigger path had to be generated.

7.4.1 Circuit Description.

For ease of use with transputer based systems, the control link between the computational circuits and H.T. power unit was based on the Inmos link adaptor the CO11 in mode 1. In this mode the link adaptor has a separate 8-bit input and output port and associated control lines. As the link was needed to provide a command path to the supply unit, and no resultant messages were to be passed back from the supply, the input side of the chip was disabled by tying the inputs to their inactive state. On the output side of the chip the received command was decoded. If the command byte was **1111111x** where **x** is either 0 or 1, then bit zero was taken to be the state of the supply, either a 1 for output on or a 0 for output off. Any other command was passed to an eight bit latch which fed the D/A converter used to set the output voltage.

During power-on reset the D/A input is cleared to zero and the on/off Controller is set to the off state.

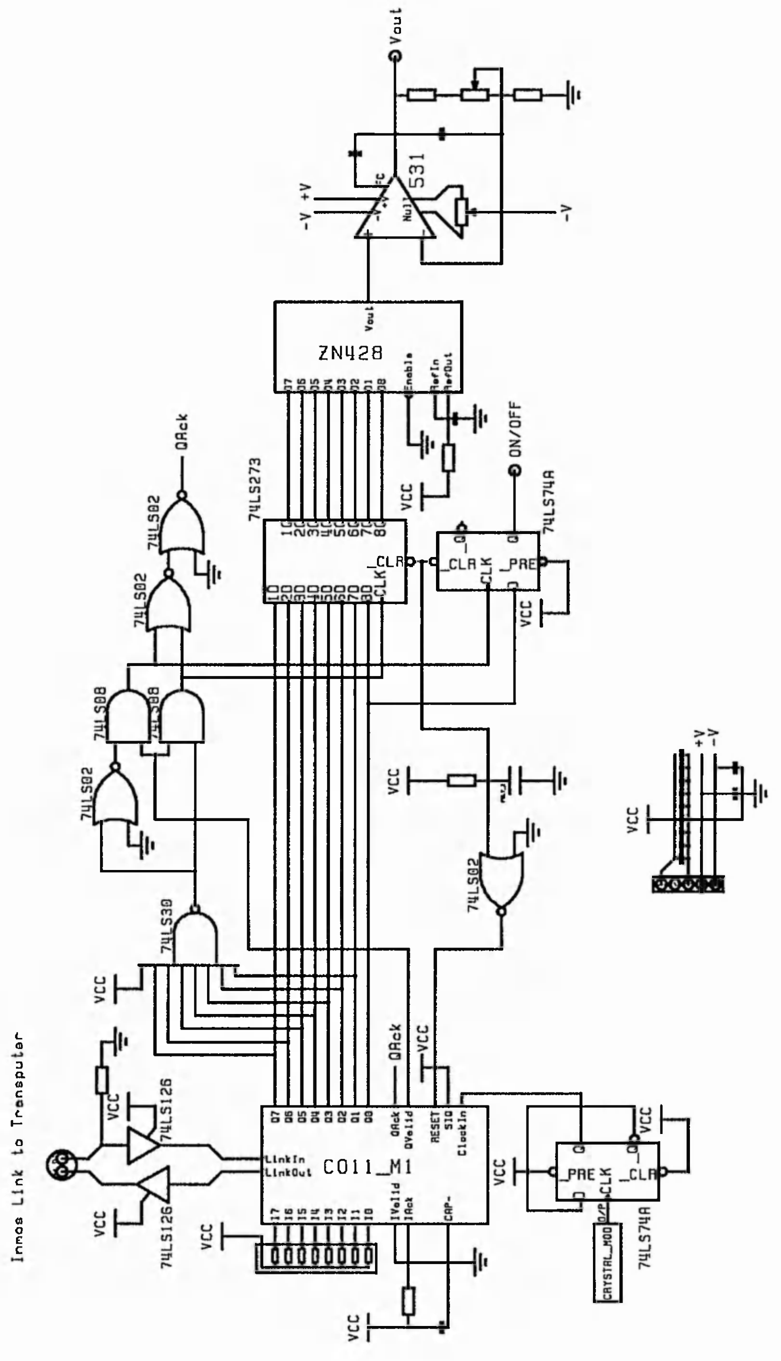

Figure 27 High Voltage Control Unit.

Figure 27 shows the link adaptor, CO11_M1, with its inputs pulled high by a single in-line resistor network. The output of the adaptor goes to the latch and then the digital to analogue converter with added output buffer. In the centre of the diagram the power-on reset circuit can be seen feeding both the link adaptor and the latches for the D/A and the high voltage output switch.

7.5 Automatic System Calibration Unit.

To allow the system to automatically calibrate itself an **Automatic Calibration Unit** was created which is closely linked to the Data Acquisition Unit. This unit gives the system the ability to fully calibrate the analogue input sections on-board the Data Acquisition Unit, working out the true gains and offsets for each channel. To do this function the system was fitted with a turn key switch with three different positions.

The **System Wide Multiplexer** unit, was part of the original design specification of the flow meter system but was not implemented at this stage. Instead the whole unit was operated in a single system mode and so the outputs for the first stage multiplexer, available on the Data Acquisition Unit, were utilised to form part of the automatic calibration system.

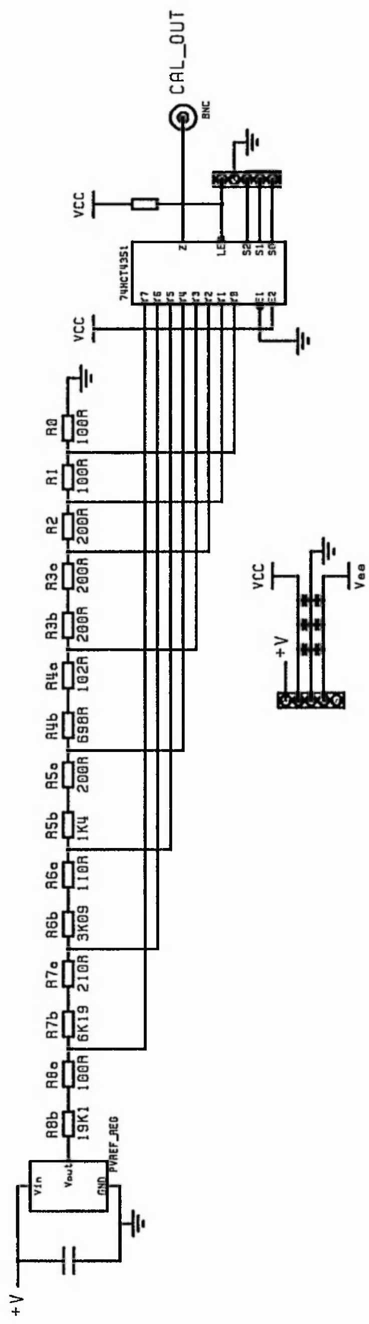

Figure 28 Calibration Unit Voltage Generator.

Figure 28 shows the calibration voltage generation section. The inputs **S0, S1** and **S2** are driven by the unused multiplexer signals. The output of the precision voltage regulator is 10.24 V ±0.03% and the resistor chain has resistors with values of ±0.1% all leading to an accurate voltage within ±0.1% of the selected value. This voltage is fed to an analogue multiplexer, the 74HCT4351 in figure 28.

To avoid loading on the resistor chain causing a drop in requested voltage due to an effective parallel resistance shorting out part of the resistor network, the generator is followed by a voltage follower using an LF351 J-FET op-amp. This has been designed so that if the first stage multiplexers were introduced into the system, then the voltage generated would be compatible with the multiplexers attenuation stage. For this reason the output voltage of the calibration unit is attenuated by 50% to keep compatibility. This voltage is now fed to the common voltage tap on the three position key switch.

Normally the key switch is set to **position 1** where the output of the differential line receivers is connected straight through to the attenuators and the **Signal Selector**. When offset calibration is required the switch is turned to **position 3** where all the inputs are connected to ground potential. The system now tests each channel in turn computing the offset associated with each gain.

For gain calibration the process starts with the offset calibration, as above. The key switch is then turned to **position 2** where all the inputs are now connected to a common input voltage, in this case the calibration voltage. The system now cycles through every channel and gain combination selecting a different calibration voltage for each gain. Using the known input value and the offset value the true gain can be computed from the information gathered:

$$\text{True Gain} = \frac{\text{Integral} - (\text{Number of Processed Samples} \times \text{Offset})}{\text{Number of Processed Samples} \times \text{Calibration Voltage}}$$

The computation of the true gain is performed using real arithmetic with integer to real conversion for all integer parameters.

The **True Gain** and **Offsets** are all kept in two dimensional arrays and are used in the formulation of the true input signal values.

Tests using this unit have shown that the gains within the system are all a factor of two up from each other, when going from unity gain to a gain of 128.

8. System Software.

The following section describes various aspects of the software developed for the system. It covers the development of mathematical functions for use with Proto Occam, 8.1, the theory and application of adaptation techniques used, 8.2, through to optimization of the main system software, 8.3.3, and the software elimination of problems encountered with the actual sensor signals, 8.4.

The software listings are too large to be included in this thesis, but are available in an internal publication within the Department [36]. The contents of this publication is listed in appendix C.

8.1 Proto Occam Maths Functions.

Proto Occam contains a set of 64-bit floating point functions for performing most of the basic floating point operations. However, the function '**x to the power of y**', required for some of the processing, was not supported by Proto Occam. To do '**x to the power of y**', the functions '**Ln x**' and '**e to the power of x**' were created.

In the first stage of creation, series were generated for the functions based on a set Chebyshev's series of polynomials. The actual functions and coefficients were derived from FORTRAN functions by Cody and Waite [29]. The appropriate coefficients were selected from a range of coefficients based on the number of bits in the floating point representation.

After writing the functions in Proto Occam and compiling them, it was found that they did not yield accurate results. After trying to track down the faults in the algorithms it was decided that a change in direction was needed. New algorithms based on Z80 assembly listings, from the Sinclair ZX Spectrum computer, by Logan and O'Hara, [30, 31], were written.

Transformations were created to convert the 40–bit floating point representation of real numbers used in the Spectrum to the 64–bit floating point numbers in Proto Occam. The operation of this transform was carefully checked as without true conversions the functions generated stood no chance of operating.

After completion of the functions it was found that they too did not work. Debugging these functions showed that the fault was in one of the predefined Proto Occam functions. Once this fault was corrected both versions of the mathematical functions worked.

To complete the standard floating point operations the trigonometric function of sine, cosine, and tangent with their inverse functions, were also implemented again based on assembler listings by Logan and O'Hara [31, 32].

The routines developed were introduced into the software simulation of the system in Proto Occam. This simulation was used until an IBM development system from the SERC/DTI transputer initiative arrived running Occam 2.

This new version of Occam actually turned out to be a full implementation of the language, and it contained all the mathematical routines previously developed. The functions were therefore dropped in favour of the predefined mathematical functions of Occam 2.

The SERC/DTI development system was based around a T414 transputer. This was later replaced by a departmental development system based around the T800 transputer.

With this T800 system all the libraries had to be recompiled for T8 processor type as they were all originally compiled by Inmos for T4 type processors. The code of a T4 type processor is mainly compatible with the T8 type except for the floating point operations. With a T4 processor these a supported by software, while with the T8 some are actually hardware. On recompilation if was found that the function which caused the problems in Proto Occam was not supported by the T8 based system although they were available on a T4 type processor. To overcome this and similar difficulties some of the original Proto Occam functions were reintroduced.

8.2 Adaptation Theory.

For this system a range of 16-bits is needed for analogue to digital conversion due to the varying levels of signal, but at any given instance the actual resolution of 8-bits conversion was considered to be adequate. This requirement means that an adaptive quantization unit could be built which could yield fast 16-bit answers with only 8 significant bits. Simulation of the design showed that with adaptation this converter system would rival the 16-bit converters.

The specification of the system is that it should be able to measure both positive and negative voltages of the same magnitude. Since the flash converter used is only capable of converting positive voltages to a digital representation, half the full scale input voltage of the converter is added to the input signal. This produces a conversion result with offset binary.

Offset binary is where all the binary values are offset by some fixed value. In this the values are offset by 128 decimal, so for an input of zero volts the converter will produce an answer of 10000000 binary (128 decimal).

There have been two types of encoding for the adaptation algorithms, hardware and software. Both of these techniques shall be dealt with separately with the hardware being discussed first as this was the method first used and the software was developed from it.

8.2.1 Hardware Encoded Models.

There have been two hardware encoded models both of which were for adaptation on a sample to sample basis, but could equally be used for cycle to cycle adaptation with slight modifications.

The gain computation algorithms are based on the result of the absolute value of the converter's output. Since the converter system uses offset binary, 128, the actual offset, is subtracted from the output to produce the true value that the converter represents. This, however, is not done by some complex subtraction unit, since the subtraction of 128 turns out to be the inversion of the most significant bit.

To reduce the amount of hardware needed to obtain an absolute value, when negative values are encountered they are converted to an absolute value by using one's complement rather than by the true method of two's complement. The difference between the one's complement and two's complement is that the two's complement is a one's complement with one added to the result. The addition of one was considered unnecessary for this application as only an indication of the value is required.

By approximating the absolute value for negative values, the absolute value could be generated by the logical XOR (exclusive-OR) function, with the most significant bit. With this process bit eight in the output will always be a zero as any positive value from the converter bit eight is a zero and, with a negative, it will start as a one but the XOR function will change this to a zero. Since bit eight is always a zero, only the lower 7-bits are used for gain computation.

8.2.1.1 Inter-Sample Adaptation.

The inter-sample adaptation technique involves the comparison of the absolute value of the converter output with two fixed values. If the absolute value is less than the first number then an increase in the sample gain by a factor of two is required for the next sample of that signal. If it is greater than the other value then a reduction by a factor of two is required. The new gain calculated is entered into a special storage which is automatically read when the channel is next used.

8.2.1.2 Intra-Sample Adaptation.

The fast adaptation model used for intra-sample adaptation uses two actual sample passes to create one adapted sample. The first phase of this technique involves the taking of a unity gain sample and examining it to find the range of the converter covered by that particular sample. From this the adaptation circuit decides what gain should be used to boost the signal to cover the full range of the converter.

The bit number of the most significant bit of absolute value of the unity gain sample is found. The power of the gain required is now given by subtracting the bit number from 7. Two to the power of this result is the gain required.

Examples:

10010110 from the converter is converted to
$\underline{00010110}$ The bit positions are defined, to suit computation, as
–7654321 so 5 is the most significant bit giving a gain of $2^{(7-5)}$ (4).

01011xxx should now be yielded as a true sample value.

01110101 from the converter is converted by one's complement to
$\underline{00001010}$ as the signal was negative, (msb = 0). The bit positions are
–7654321 so 4 is the most significant bit giving a gain of $2^{(7-4)}$ (8).

1010xxxx should now be yielded as a true sample value.

10000000 given from the converter goes to
$\underline{00000000}$ as the converter uses offset binary. The bit positions are
–7654321 as there is no msb, 0 is used giving a gain of $2^{(7-0)}$ (128).

0xxxxxxx should now be yielded as a true sample value.

01111111 given from the converter goes to
$\underline{00000000}$ as the converter uses offset binary. The bit positions are
–7654321 as there is no msb, 0 is used giving a gain of $2^{(7-0)}$ (128).

1xxxxxxx should now be yielded as a true sample value.

Where **–** for the bit position indicates the bit is unobtainable using the hardware, and **x** for the result indicates the bit can not be determined as it depends on the signal value.

8.2.2 Software Models.

Three software models are discussed here with the final model outlined in the results section, (section 9.2). The first model written is basically a software version of the inter–sample adaptation technique in section 8.2.1.1. The second model uses full calibration techniques to improve the process while the final model uses knowledge built up about the system to aid the adaptation.

All of these models are for adaptation after a complete set of readings have been taken but could be adapted for other modes of operation. The software versions have evolved from each other due to operating difficulties found whilst in use.

In all cases the underlying process is to check the maximum positive value obtained during a cycle and the maximum negative value against certain criteria and find which region describes the result. The classification of regions and their general action caused is detailed below:

a. Readings outside the converters range producing a 0 or 255 value from the converter. This causes a gain reduction. If the reading was obtained using unity gain then a warning is given to indicate that the converter system has met a value which is outside the operating limits of the system.

b. The readings are approaching the out–of–range values of 0 or 255. This causes gain reduction, but if already at unity gain then no warning is given as the value is not actually out–of–range of the system.

c. If by converting the reading to an equivalent input voltage and by operating upon it the next higher gain the reading that this would produce is still within the limits of acceptability, as defined in **b**, then the next higher gain is used unless the gain is already at the highest in which case the gain remains the same.

d. If none of the above criteria hold then the gain is assumed to be correct as it is within the correct operating limits and the use of a higher gain would produce a reading which is not.

As the selection of points **b** and **c** are the only main differences between the models, each model presented will deal with these points, and present them in equation form. If both equations hold then the point is deemed selected.

8.2.2.1 First Software Model.

In this first model the gains of the system are assumed perfect and so too is the offset associated with that gain.

Point **b** checks:

b1: Most_Positive >= Upper limit

b2: Most_Negative <= Lower limit

Point **c** checks:

c1: Most_Positive − 128 = y
$2y + 128 <$ Upper limit

c2: Most_Negative − 128 = z
$2z + 128 >$ Lower limit

8.2.2.2 Second Software Model.

This model is a slightly more complex model which takes into consideration the information gained during a system calibration. Here there are no assumptions about the gains or offsets.

Point **b** checks:

b1: Most_Positive >= Upper limit

b2: Most_Negative <= Lower limit

Point **c** checks:

c1: (Most_Positive − Offset_{now}) / Gain_{now} = y
Gain_{next} x y + Offset_{next} < Upper limit

c2: (Most_Negative − Offset_{now}) / Gain now = z
Gain_{next} x z > Lower limit

8.2.2.3 Knowledge Based Model.

The knowledge based model tries to predict the outcome of the next cycle based on knowledge gained about previous cycles. The model tries to compensate for the fluctuations in the signals in order to reduce the gain increase which is likely to produce a conversion error.

The definition of this model assumes factors are created from the knowledge called **Posfactor** and **Negfactor** which are compensation factors generated for the changes due to positive and negative values respectively.

Point **b** checks:

b1: (Most_Positive + Posfactor) >= Upper limit

b2: (Most_Negative − NegFactor) <= Lower limit

Point **c** checks:

c1: (Most_Positive + Posfactor − Offset_{now}) / Gain_{now} = y
Gain_{next} x y + Offset_{next} < Upper limit

c2: (Most_Negative − Negfactor − Offset_{now}) / Gain now = z
Gain_{next} x z > Lower limit

The change in gain using this adaptation mode will favour a reduction in gain as values which are out of range of the converter yield no useful information as the true value of the input signal can not be determined.

Checks in a knowledge based system could be added to test the effects of the results generated and create any modification of the model necessary to produce a better result.

8.3 Main System Software.

This section on the main system software is split into three sections. The first outlines the overall system architecture and that of the transputer itself. This is followed by two sections detailing the operation of the software and how it was optimized for performance in terms of data throughput.

8.3.1 System Architecture.

Figure 29 shows the internal architecture of the T414 32-bit transputer. There are four links on most of the transputer processor devices, (M212 disk processor has only two). Each link provides a bi-directional serial data path transmitting at either 10 Mbps or 20 Mbps, selectable by external processor pins.

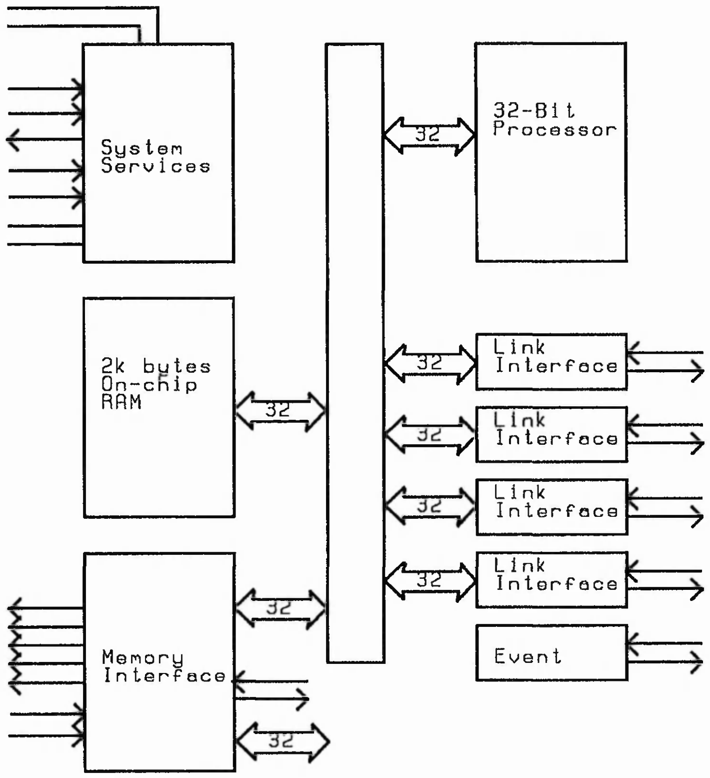

Figure 29 Internal Transputer Architecture.

The basic system configuration, figure 5 page 41, was modified during development of the system to allow the insertion of the IBM PC/AT based Transputer Development System containing a BOO4 transputer card. All the links within the system run at 20 Mbps, except that used for connection to the BOO4 which runs at 10 Mbps. Figure 30 shows the new system configuration used throughout the development phase. The link between processor node 3 and the Controller, node 6, has been broken to facilitate the insertion of the BOO4, (the dashed line in figure 30 shows the broken link).

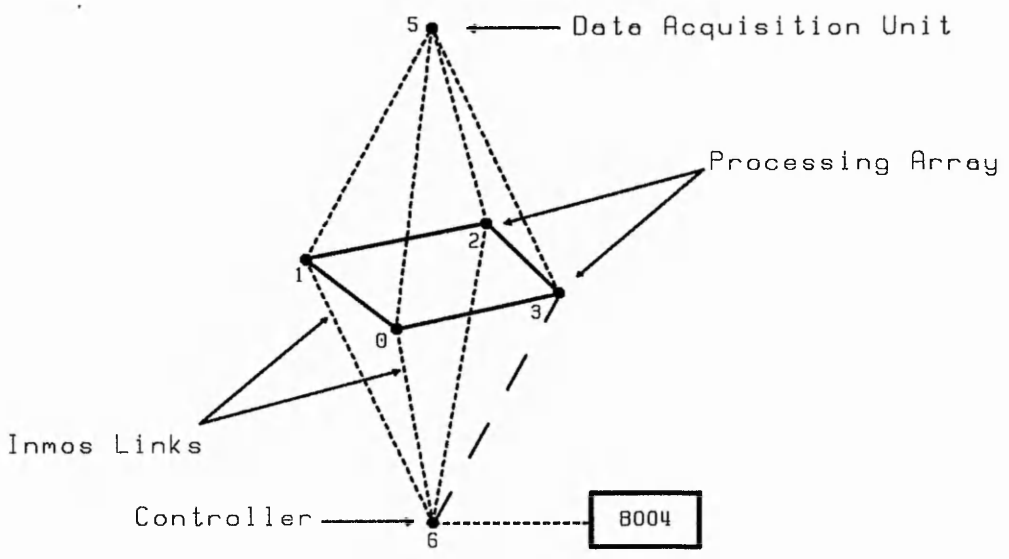

Figure 30 System Configuration.

The heavy lines creating a loop at the Processing Array level, show the ring network within this unit. Parameters passed into the program during the program configuration stage, inform each processor in the array, (now referred to as the *network* or *network processors*), whether or not it is connected to the Controller. When communication is required between a network processor and the Controller, it will either pass the message directly to the Controller, depending upon the configuration data, or via the next network processor on the ring network.

8.3.2 Operational Details.

The software for the system is downloaded from the PC via the driver running on the PC's BOO4 transputer card. Normally external networks of transputers are loaded via the Inmos configurer, but since there is a suite of programs which need to run in parallel with the external system, providing operator interaction, this configurer has been combined with the software needed to drive the external system.

When this driver is loaded and run in the normal manner, it asks which link to download the external system's software. The link to be used is normally link 3 and this is accepted as a default value. If there is any problem with the download then the software will ask if download is to be retired. The operator may choose to try again or return to the top level of the development system.

After a successful download, the external system will automatically configure itself from the information supplied to the program during program configuration at compile time. The Data Acquisition Unit passes back to the Controller the resolution of the flash converter and the network processors will inform the Controller whether or not they have a direct data path to it. This configuration data is stored by the Controller and used for communications with the other processor in the target system.

The system is now menu driven and the operator may drive an interactive session with the external system by pressing key sequences on the PC. At the highest level of the system, the main menu is not normally displayed but the operator may request it by pressing the **SPACE BAR**.

When the user presses a key on the IBM keyboard, the software will decide whether it is an executable command for the PC or whether it should be passed to the external system. The instruction decode path within the external system is that user commands enter the Controller where they are decoded and a decision is made as to whether or not an 'upper array' command is required. Upper array commands are passed to a network processor which in turn passes them to the Data Acquisition Unit. The Data Acquisition Unit further decodes the instruction and determines whether or not to start the network processors.

The program on each network processor is identical. Parameters passed to the program in the configuration stage determine the actual operation of the network node.

The PC also provides the system with a means of visual output as well as access to disk storage. When a transputer in the external system prints any message, the data is routed through the system down to the IBM PC where it is displayed. Each message displayed in this manner has the processor's node number attached to the front of the message so that the operator may easily identify where any particular message has come from.

The operator can abort the programs at any stage, thus providing a good level of fault tolerance to the system. This feature was added so that any interference introduced into the transputer communication links by high voltage switching, would not lock-up the whole network and the user would then have a chance of soft-restart. To abort any command and return to the download stage of the PC program, the operator only needs to press any key whilst there is no response from the external system. If the Controller is ready to decode commands then pressing **Q** will have the same effect. The normal way for the operator to exit the program is by pressing **A**, this has the effect of shutting down each processor cleanly. If any processor does not shutdown then the operator may abort the clean shutdown.

Once the external processing system has been stopped, either by abort or shutdown, the PC software will either ask whether the software should be downloaded again, in the case of abort, or ask the user to press any key and then the system will return to the top level of the development system. In some cases it is possible using the debugging facilities, to find the cause of the fault which needed the abort.

8.3.3 Program Optimization.

Speed is of prime importance in this application, as the faster the data throughput, the better the result of the signal characteristic extraction.

As explained earlier, transputers usually only have 2K of memory and as a result this restricts the length of any program. To add anything to the software the current programs needed to be reduced without any loss in functionality. To do this the programs were optimized for compactness.

What follows is a brief look at the optimization techniques used on the program and their results.

8.3.3.1 Space Optimization.

Space optimization is concerned with the reduction of the current memory requirements without loss of program functionality.

The localization of variables and the use of parameter passing as 'value only', (**VAL** in Occam), can reduce the memory requirements and open the way for buffering. This process was used in the original stages of optimization. Variables were transformed into local variables, wherever possible, and values were passed into procedures rather than using globals.

To reduce space further some communications which use the same format were changed to one procedure, and calls made to it passing the data to be transmitted. After just localization and a reduction in communication definition, a general reduction in program size of 11% was made.

8.3.3.2 Speed Optimization.

Speed optimization usually increases the amount of memory required by the process. On any type of microprocessor, a speed increase can usually be performed by just expanding out loops more and more, until a whole loop has become a large block of repeated instruction. With the transputer, simple loop expansion can cause an overall reduction in speed rather than an increase if array indexing is used. The problem of array indexing is discussed later in this section.

Loop expansion is not the only form of speed increasing technique available on the transputer. Inter–transputer communications can cause overheads if not properly formulated. If a programmer can detach the communications from the main processing then the execution speed will generally increase. Other speed increases associated with link communication can be in the form of data buffering. The larger the amount of data to be transferred on a link, the quicker the overall communication. Small packets of information are transferred slower because the links are all DMA controlled. Direct memory accessing favours the large communication packets as the overhead in DMA set–up becomes a smaller percentage of the overall transmission time per byte. Data buffering puts a large overhead on the memory requirements of the node.

In the general programming example that follows, the speed of the inherently sequential programming section has been increased by detaching the collection of data from the sending. The collect routine **COLLECT** is still a sequential operation, but the actual send routine sends the data to each processor in parallel. The actual communication statements set up internal DMA Controllers, (direct memory access Controllers), with the message and so the loss in processor performance is significantly reduced.

General Structure of Optimized Program.

```
... Define Arrays.   -- Define variables. (Details hidden).
... PROC Collect     -- Define procedure. (Details hidden).
... PROC Send        -- Define procedure. (Details hidden).
SEQ                  -- Start a sequence.
  Collect(A)         -- Call procedure Collect passing A.
  PAR                -- Start parallel processes.
    Collect(B)       -- Call Collect.
    Send(A)          -- Call Send.
  SEQ i = 0 FOR (RequiredBlocks - 2) / 2 -- Repeat.
    SEQ
      PAR
        Collect(A)
        Send(B)
      PAR
        Collect(B)
        Send(A)
  Send(B)
```

With only small arrays being used to buffer the data, the speed increase for four signals is quite significant over the simple sequential approach, 52 ms compared with 79 ms for the same amount of data, 4000 points in each signal.

Part of this speed increase can be attributed to the efficient coding of the COLLECT and SEND routines. These routines use 'sliced' arrays and this reduces the range checking inserted into the program. What follows is a look at the general COLLECT and SEND routines used to create the speed increase mentioned.

Note:–

A 'sliced' array means a partition of that array.

General COLLECT and SEND routines.

```
{{{PROC Collect
PROC Collect([]BYTE Data) -- Define a procedure.
  DataFor0 IS [Data FROM  0 FOR 16]:
  DataFor1 IS [Data FROM 16 FOR 16]:
  DataFor2 IS [Data FROM 32 FOR 16]:
  DataFor3 IS [Data FROM 64 FOR 16]:
  SEQ i = 0 FOR 16        -- Repeat 16 times.
    SEQ
      Process:=Req1
      DataFor0[i]:=AtoD
      Process:=Req2
      DataFor1[i]:=AtoD
      ... Collect data for 2 & 3.
:
}}}

{{{PROC Send
PROC Send(VAL []BYTE Data)
  VAL DataFor0 IS [Data FROM  0 FOR 16]:
  VAL DataFor1 IS [Data FROM 16 FOR 16]:
  VAL DataFor2 IS [Data FROM 32 FOR 16]:
  VAL DataFor3 IS [Data FROM 64 FOR 16]:
  PAR                       -- Execute in parallel.
    AtoDout0 ! DataFor0
    AtoDout1 ! DataFor1
    AtoDout2 ! DataFor2
    AtoDout3 ! DataFor3
:
}}}
```

The COLLECT routine can be further improved by expanding the main loop '**SEQ i = 0 FOR 16**' and replacing it with separate assignment statements. Keeping to 16 for the main block, large arrays can be efficiently handled. Sixteen is an important figure in array indexing because after that a '**prefix**' instruction is needed to expand the subscript range for each multiple of 16. This 'prefix' is one extra byte in the instruction which will cause slowing of the routine.

8.3.4 Conclusion.

General optimization techniques can easily be implemented in real situations by good programming practices. Some program optimization above and beyond the general form is mainly application specific. What follows is a list of general rules which will apply to any program.

i. Keep all variables local.

ii. Pass variables to procedures rather than using globals.

iii. Detach communications from other processing where possible.

iv. Communicate in the largest blocks possible.

v. Reduce repetition of similar communication protocol blocks in favour of placing them into a procedure.

vi. Use array slicing to reduce range checking.

vii. Expand loops in blocks of sixteen to reduce 'prefix' instructions.

viii. Retype, (e.g. transform a 2D array into a 1D to quick access), well-used array elements or use equivalents, (abbreviations).

ix. Use array slicing over looped array indexing, clearing large arrays by the replication of a cleared array slice.

8.4 Software Elimination of Signal Errors.

The main problem with pulse charge flow measurement is one of noise. This has reduced by several stages of signal processing.

8.4.1 Stage 1 Noise Reduction.

The method of reducing noise at the data acquisition level is that the Data Acquisition Unit actually pre–processes the information and intelligently selects only the cycles which will provide useful information. The pre–processing determines two things:

1. Whether injection has taken place.

2. If there are large current spikes due to the charged powder breaking loose from the walls of the pipe and re–entering the flow.

If there is no injection, or large current spikes due to re–entrant particles, then the system has the ability to terminate the cycle and restart it. Repeated failure to obtain injection will cause the Controller to increase the H.T. voltage. If the voltage reaches a predetermined level, then the Controller will shut down the entire system reporting a failure of the H.T./charge injection unit.

Initially this intelligence created a new problem which was seen when capturing waveforms for post processing. This problem was that the system was very quick at recognizing either the failure of injection or current spikes and could quickly stop the present cycle of injection and start another. The problem encountered here was that the '**turn–round**' time was too quick, the system was able to turn the H.T. supply to the needle off, because of the failure, and start a new injection cycle by turning the H.T. on. Since the **turn–round** time was in the order of microseconds, the system could see the air naturally losing the charge due to the supply being turned off. This sharp fall in current was interpreted as powder re–entering the flow and so the cycle was again terminated. This problem was overcome by the addition of a pause to allow the H.T. to return to its normal switch on value.

8.4.2 Stage 2 Noise Reduction.

Stage 2 entails the disregarding of data sets that contain any value which caused the adaptive quantizer to overflow or underflow. This initial elimination stops readings entering the final processing stage which are currently out of range of the system. When this occurs a new gain is applied to the next cycle of data, or if it decided that this cannot rectify the fault, then the operator will be warned. Continual over–range readings will cause the system to report an error and shutdown the H.T. unit until it is reported clear by the operator.

The next phase of the post–processing is to filter the mass flow rate calculated and reduce the effect of random fluctuations in the flow, upon the mass flow reading, (see section 9.2).

8.5 Software for Results Processing.

The software for results processing is split into two main sections.

1. Software based on the transputer inside the IBM PC/AT Transputer Development System.

2. Software based on a standard IBM PC/AT.

The transputer based software was concerned with the development of equations relating the mass flow rate of the system to the actual processed information obtained from the sensor currents. This work, as with the IBM based work, stemmed from the addition into the system's software of the database compiler. The main task of the IBM based software was for the displaying of the database with the ability to create hardcopy results of captured waveforms of interest.

8.5.1 Transputer based Software.

The transputer based results processing software is mainly based around the development of the equations relating mass flow rate to the processed sensor readings. The main system software on the Controller contains a function which creates a database entry comprising a mixture of processed results and raw signal waveforms along with full calibration information and run-time system parameters. This information was stored on floppy discs for post processing, such as an N^{th} order data fitter with equation parser [36].

8.5.2 IBM PC/AT based Software.

There are two main pieces of software used on the IBM PC/AT. The first is mainly concerned with the displaying of results from the transputer system, while the second deals with the processing of the results.

Since the second program is mainly a database processor with little interaction between the user and the system, the main features of this shall be dealt with first.

The second program was created for the processing of the final database generated by the operation of the transputer system. The program allows the selection of data from the database which meets certain criteria defined by the operator. The program has the ability to generate two new databases by the selection criteria. The first new database contains all the selected results from the main database while the second contains the residual data.

The program contains a linear regression algorithm and export facilities to other computers. The main use of this program is concerned with the exporting of graphical data to the Apollo where another program transforms the data into graphs. For more details on this piece of software and the results from it, see section 9.2.

The first program uses the data from database discs or output files, which were created by the transputer system. When activated the program enters into a sub menu:

```
E) Enable Main Results Section.
X) External Waveform Display.
I) Injection Search Rule Setup.
S) Statistical Display.
D) Data Display.
Q) Quit.
```

Option **X** displays a waveform generated by the transputer system under an interactive capture and save session. During a run of the system the user may decide to keep a block of data captured by the system. The results can be saved in one of two forms. The first is a textual file containing a simple graphical trace of the waveform patterns that the data represents. The file contains the same data as can be written to the screen during the transputer session. The next form of block save is as raw data. Both of these data saves record the request made to create the waveform, along with other useful system parameters.

A trace saved as raw data has its information encoded into lines of compact hexadecimal numbers representing the values read from the flash converter. This is the information used by the first graphical display program which produces a high resolution representation of the waveform. This screen picture may be dumped to a printer by a special print routine written, since the standard screen dump provided with the PC did not produce a true representation of the screen. The standard PC screen dump produced only two thirds of the screen, because to produce the high resolution displays the versatile graphics adapter, VGA, was used in its high colour/resolution mode. This mode used more of the screen memory than a normal graphical display which can be dumped successfully with the standard screen dump.

The **Statistical Display**, option **S**, displays a bar chart generated from the distribution of results found whilst gathering one database disc, (see section 9.2 for more details). The chart can be dumped on a printer or can be exported to the Apollo where another program can transform it into a high quality plot.

The **Data Display** option, **D**, provides a simple graphical display of any x,y data.

Option **E** enables the main section of this program and the system then waits for the user to place the database disc in drive **A**. All the header information, appendix B, is read from the disc and the system moves into a mode of displaying processed results. By quickly pressing any key twice, the system changes mode and moves into a menu driven section.

The main menu looks something like this:

```
P) Processed Data Display.
C) Processed Data Continued Scroll.
S) Processed Data Search and Display.
W) Waveform Display.
X) External Waveform Display.
H) Header Display.
B) Brief Header Display.
I) Injection Search Rules Setup.
D) Dump Calibration Data on Printer.
L) Load Another Results Disc.
Q) Quit.

Which one?
```

The program passes all the main printing via a screen formatter so that, for example, the main menu above will actually appear in the centre on the screen, centred in both axes.

When the user presses one of the listed response keys the system will take the appropriate action. Options **B** and **H** will display either the main data from the header, (the main counters and air settings), or the full header data which includes all the calibration data for each signal and gain. Option **C** is to continue the mode of operation that the system first starts in. In this mode the system scrolls through each processed result in turn displaying all the relevant information available from it and warning of any run–time overflows (signal being out of limits of the converter system), which could make the results unreliable. Option **I** allows the operator to change the way the program decides that the injection current has reached the point of injection. Normally the system defaults to the same rules as used by the flow system. This option is provided to allow experiments to be made on real data to see how the changing of certain search parameters will cause either the rejection or acceptance of a particular result. **Load Another Results Disc**, option **L**, causes the reading of a new database disc.

The **Processed Data Display**, option **P**, allows the operator to choose a particular processed data results set and display it. Using the **C** option after this will cause the scrolled results to effectively jump to the result set after that just chosen by **P**.

The **Waveform Display**, option **W**, allows the user to select a waveform by number from within the database. The display is similar to that of the first program except here there are three waveforms packed into one waveform section, so the user is given the opportunity to overlay the waveforms on one graph. The user is also given the opportunity to automatically scroll through the waveforms on the disc from the current position selected.

Once these selections have been made, the user is prompted whether data compression or windowing is required; (only if the data for one waveform exceeds that plottable with one sample per **x** pixel. This is usually the case). If compression is chosen then the entire waveform is scaled to fit onto the screen. Windowing allows the user the ability to select the amount of compression required. Any data which cannot fit onto a windowed screen, is ignored. Each waveform displayed has the equation needed to convert the data to a real value printed near the top of the screen.

Once the waveforms have been displayed the user is prompted whether or not a screen dump is required. Choosing this option will produce a hardcopy of the waves on the printer. If the waveform scroll option has been activated, then at the prompt about printing, typing **Q** will cause the program to abort the scrolling and return to the main menu. Pressing **X** will cause the export of the raw data to three disc files. These files contain raw data values in ASCII decimal representation, (one value per line), which represent the actual A/D values. This data can now be used in other ways. One use of the extracted data is that a routine was created on the Apollo that could reproduce high quality waveforms on the Apollo which can then be annotated for displays.

After each command other than quit, the system returns to the main menu.

9. Results.

9.1 Experimental Procedure.

Figure 31 shows the experimental rig built around the pneumatic transport system. The injector and sensors which are shown in more detail in figure 23 page 90, are coupled to the transputer system via current to voltage converters providing a differential output voltage proportional to the induced sensor current.

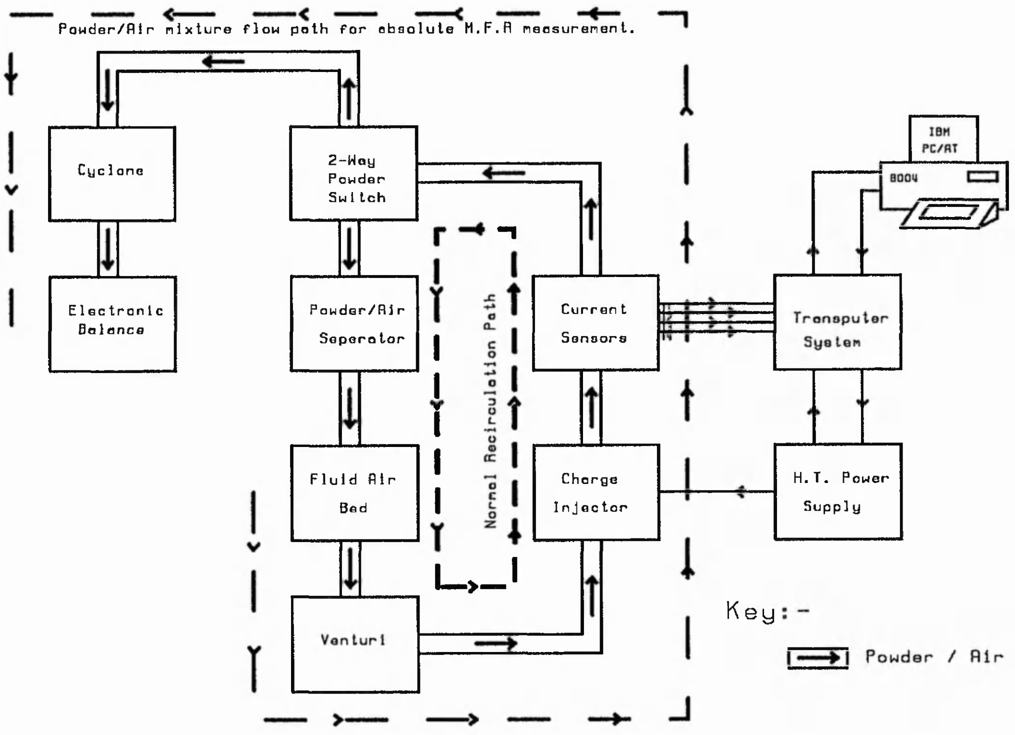

Figure 31 Experimental Apparatus.

The transport system comprises a fluidized powder hopper with primary and secondary air controls to create the desired mass flow rate. The primary air in the system provides the suction on a venturi system and the secondary air provides the powder with a velocity control source. After the powder passes through the sensor unit it is either directed to an electronic balance for absolute mass flow measurement, or it is separated from the air and returned to the hopper.

The initial system calibration was provided by the Automatic Calibration Unit described previously (section 7.5). As the individual gain components of the system have fixed gains, only offset calibration was necessary for further runs. The procedure here, however, shall include the gain calibration stage.

The basic experimental procedure is that the system is switched on and any offset produced by the differential signalling is adjusted out so that when using the maximum gain of 128 the raw A/D reading is equal to the offset value of 128 decimal.

The Calibration Unit was now connected into the system and the auto-calibration checked to prove that the system could accurately determine the level of different input signals when using different gains. This process was found to be successful with the conversion accuracy now at ±1.5 LSB in 8-bits.

After this set-up is complete the transputer system calibrates all the system offsets with the sensor outputs connected into the signal multiplexer. By calibrating the offsets in this manner, any residual offset left after initial set-up is accounted for in the system computations.

Now the system is calibrated, the task of obtaining flow rate measurements can begin. To create a range of different flow rates the primary and secondary air supplies to the fluid bed hopper are varied. Once a setting has been chosen the transputer system is allowed to make the measurements of the sensor outputs by firing the H.T. pulse, searching for injection, and processing the data derived from the analogue signals. For each flow setting, 2000 H.T. pulse cycles were used and the data gathered. The recorded data was then uploaded into the IBM PC/AT and saved onto one disc. The actual mass flow rate for the primary and secondary settings was then found by diverting the flow onto an electronic balance. By taking the difference in balance readings, (before and after), the actual flow rate can be determined based on the time the powder was diverted.

The data gathered during one run with any primary and secondary air setting is 2000 processed results, 62 raw waveform captures and the full system parameters including the calibration information. Each set of processed results yields the following information:

The requested gains of each channel, R_0 to R_3.
Positive Peaks of each sensor, I_0 to I_3.
Negative Peaks of each sensor, M_0 to M_3.
Integral of each sensor current, Q_0 to Q_3.
The time of each positive peak, T_0 to T_3.
The elapsed time of each processor, E_0 to E_3.

From the calibration information and requested gains R_0 to R_3, the true values of the sensor readings can be computed. Velocity information can be generated by using the peak times, T_0 to T_3, and the gap between the sensors. The following velocities can be generated, (where **SensorGap** = gap between one sensor = 21 mm):

V_{13}, the velocity by (T_3-T_1) / (2xSensorGap)
V_{12}, the velocity by (T_2-T_1) / SensorGap
V_{23}, the velocity by (T_3-T_2) / SensorGap

From now the sensor readings shall be referred to by the letter and number system given above, and are assumed to be converted to their true value.

Processing in this manner gave one primary and secondary setting data set on one disc, as the processed data and capture raw waveforms took up over 900K on the floppy disc. After a series of settings had been used the data was then compiled into another database and the waveforms extracted and plotted to record the actual shapes of the induced sensor currents for further analysis.

This process was very time consuming so a more direct solution was adopted to produce the large volume of data covering all the half step primary and secondary combinations. With this new process the system was still set-up and run as previously, but instead of transferring all the data to the PC to be stored on disc, the data was processed by the transputer inside the PC and the compacted results saved on the disc.

The results being extracted from the sensor readings were the mean and standard deviation of each type of reading, plus the number of readings actually used to make up the mean and the percentage of this number to the overall injections seen.

Considering just one single data set. This originally comprised 2000 attempted injections. Out of these 2000 attempts, typically the transputers had detected the occurrence of the injection in some 90–95% of these and so they contained valid information. The information stored in any particular set is the maximum and minimum values found in the signal data, the time of occurrence of the maximum value, the integral of the entire waveform and the elapsed time, along with the actual gains used to obtain the readings.

The maximum and minimum values were stored as direct A/D values and so ranged from 0 to 255. With any 0 or 255 value the actual value of the signal can not be determined as any value slightly more negative or positive would still produce a reading of 0 or 255 respectively. For this reason any sensor reading of either 0 or 255 was deemed invalid and the reading useless.

There is another possibility for obtaining an invalid reading. This is in the peak time. If any sensor further away from the injection unit recorded a peak before any closer sensor to the injector, then this reading was deemed invalid also. Invalid readings within any particular data set caused the disregarding of that individual set, so typically only 70–80% of those injections detected were used in the overall statistics saved by the computer system which made up the first of three database entry types.

The second entry type for any one primary and secondary air combination is composed of the reprocessed results of the first filtering off of any data items which lie outside one standard deviation of the mean. The final entry saved is the data distribution of second entry.

Using this new technique of gathering database entries, the flow system was run with primary and secondary air settings varying between 2.0 to 7.0 and 0.0 to 9.5 in 0.5 units per step, respectively. All the information gathered was compiled into one large database for further processing.

9.2 Main Results.

During the initial phase of this testing program, it has been necessary to modify the way in which the results are to be gathered and the operating conditions of the system prior to final data gathering.

The system's raw waveform data gathering facility was first used to gather several sets of waveforms with a variety of injection waveform shapes. From this data the algorithm used for the determination of injection was created. The simple model used in the original system simulations was found unsuitable for the real system as the injection shape had changed from that seen by Willis.

The first modification in operating conditions came when, although good injections could be seen on a digital storage scope, the transputer system rejected them. By examining the raw waveforms capture in the database discs it was found that there was a problem with the turn-round time of the system. Previously in the original data gathering used for the generation of the algorithm for injection determination, the ability to terminate a cycle and restart it was disabled as there was no way of detecting the injections. What was happening with the algorithm was that if a large negative spike was found, or the search for injection exceeded the time limit, then the Data Acquisition Unit would call a halt to this cycle as no useful data could be derived from it. On receiving the signal to abort a cycle the Controller would note the failure and start a new cycle.

The restart of any new cycle causes the H.T. unit to turn on, thus with this fast turn–round time the system had turned the H.T. off only a few microseconds before turning it on again for the start of this new cycle. Figure 32 illustrates what the system saw.

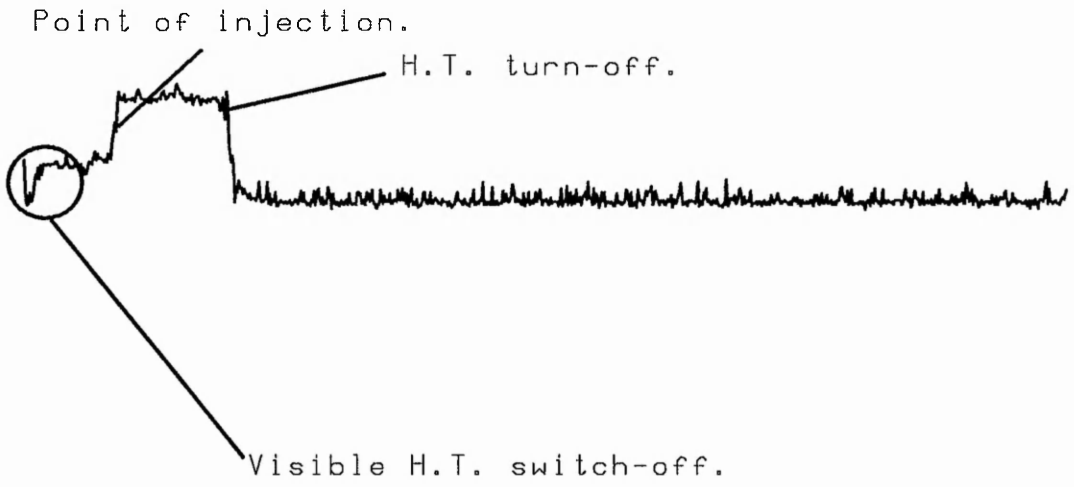

Figure 32 Example of the Fast Cycle Turn–round Timing Problem.

A new 'turn round' algorithm was generated and found successful. It was at this stage that the controlling transputer unit started to seriously malfunction due to poor circuit board construction and so the new multilayer Controller was brought into service.

Next the full operation of the system started to take place. With the powder flowing in the pipes the signals were processed and stored on database discs for later analysis.

After a set of discs had been taken the performance of the capturing process was determined. Here it was found that by using unfiltered sensor currents, there was a high level of noise. This noise, as shown in figure 33, reduces the accuracy of the sensor readings.

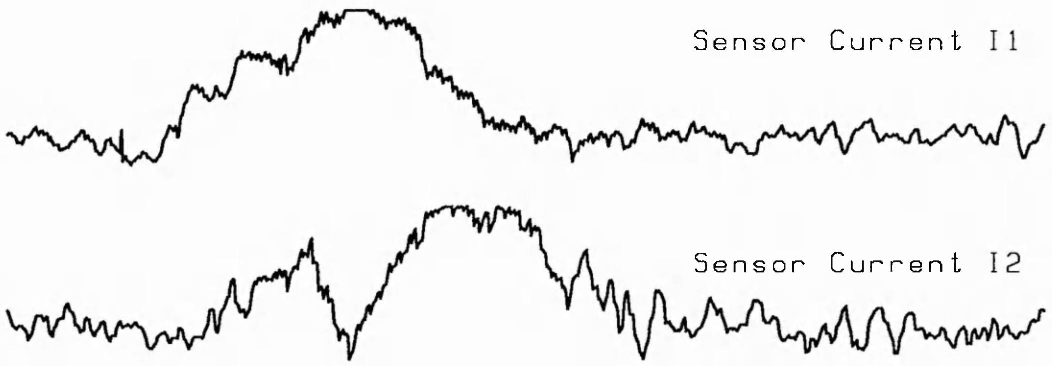

Figure 33 Unfiltered Downstream Sensor Currents.

All the waveform diagrams are shown as an indicator of their shape and are not intended to show detail of exact measurements, however, the entire length of the waveforms represents 57.6 ms of sampled data.

Low pass filters were now added to the sensor interface. Figure 34 shows a recording of an 'ideal' waveform where the injection area of the injection current is a rectangular pulse.

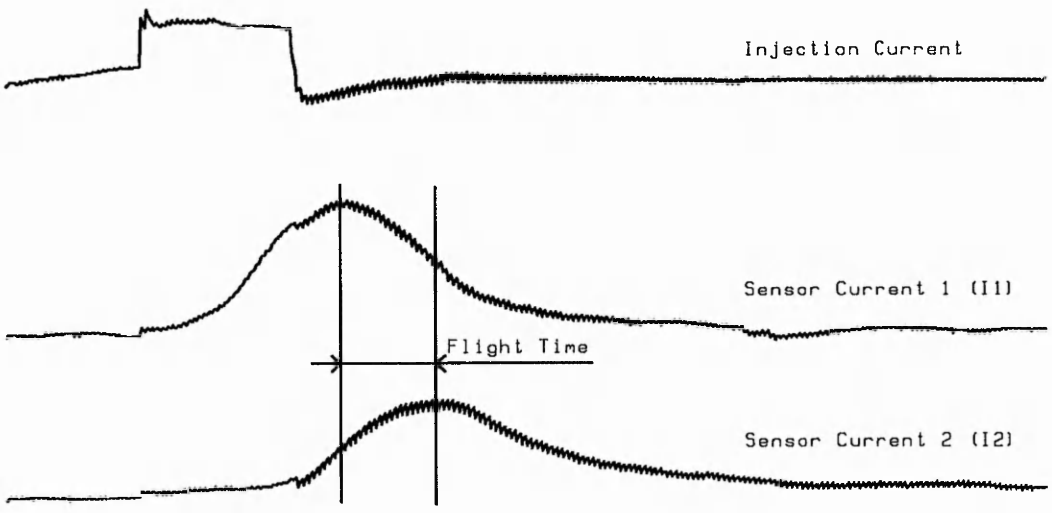

Figure 34 'Ideal' Wave Shapes.

The **Flight Time** in figure 34 is used to generate the velocity parameter V_{12} and represents T_2-T_1.

Many data sets were taken and a more 'typical' waveform is shown in figure 35.

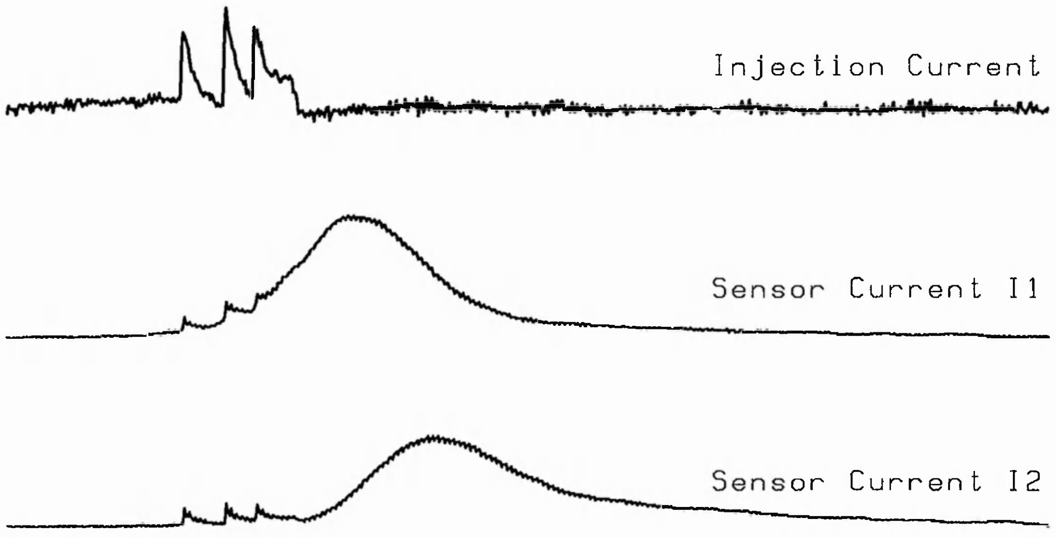

Figure 35 'Typical' Waveforms.

Analysis of the performance of the amount of data extractable from a single database disc which had no 'invalid' readings included showed that typically the system could now see 90–95% of injections, (some of these could be actually injection failures and others just missed due to the algorithm used for determination). Out of this 95% only some 25–30% were accepted, the others being rejected due to invalid readings caused by the adaptation process.

These sets were using an immediate adaptation using the assumption of a high correlation between one injection cycle and the next. Due to the natural fluctuations within the flow this adaptation process was not efficient and so was changed to a voting system.

In the voting system the adaptation required by each cycle causes a vote to be added into a poll. When the poll reaches predetermined levels for unanimous votes, (5 consecutive votes for gain increase or 2 consecutive votes for a decrease in gain), the adaptation for that channel takes place. This new mode of operation increased the acceptance level to 60–70% of those injections found being included in the final database.

The results of the new database discs created from this process were examined to get a feel for the data and what is needed to be recorded for further analysis.

The computation of statistics based on the mean and standard deviation found on the reading for one entire database disc showed a standard deviation approaching the mean. The data distribution of the results included in the computations showed that there were typically one or two results which have a very large deviation from the mean, figure 36 shows one such distribution with the scaling of the graph being generated by the maximum and minimum readings found. The results were now filtered by re–calculating the mean and standard deviation of the results based on the previous mean and standard deviation. Any results outside the mean ± one standard deviation were excluded from the new mean and standard deviation computations. Here tests were made to determine the effect of using only '**clear**' data set or any '**good**' results from all of the data sets.

A '**clear**' data set is defined as any set in which all the readings I_n, (I_0 to I_3), and M_n directly from the sensors are not either 0 or 255, and the peak times increase with sensors further from the injection unit, ($T_1 < T_2 < T_3$).

A '**good**' result is defined as any result in which the readings from which it is composed are valid. So the result I_2/I_1 is good if neither I_2 nor I_1 are either 0 or 255, and Q_3 is good if neither I_3 nor M_3 are either 0 or 255.

Figures 36, 37 and 38 show the distributions obtained by the different forms of processing mentioned.

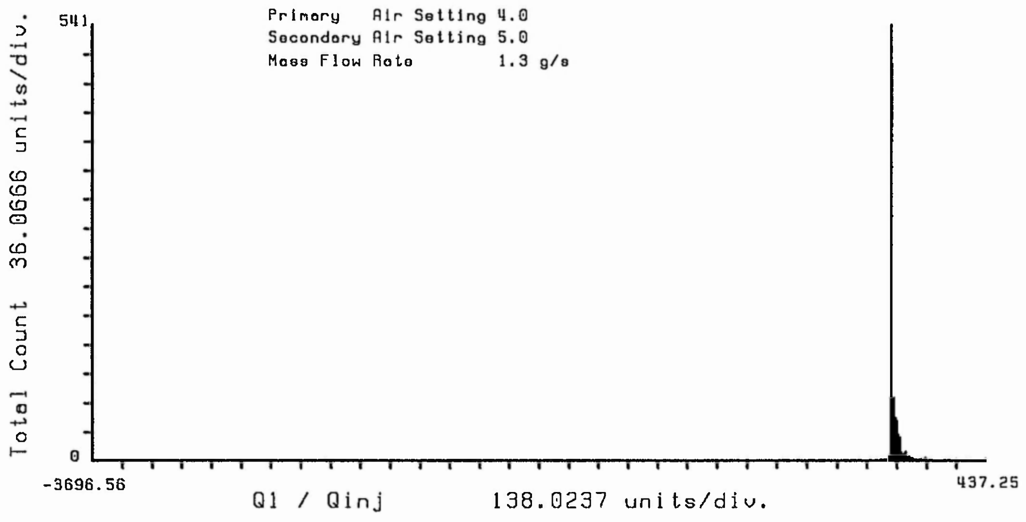

Figure 36 Raw Data Distribution of 'clear' Results.

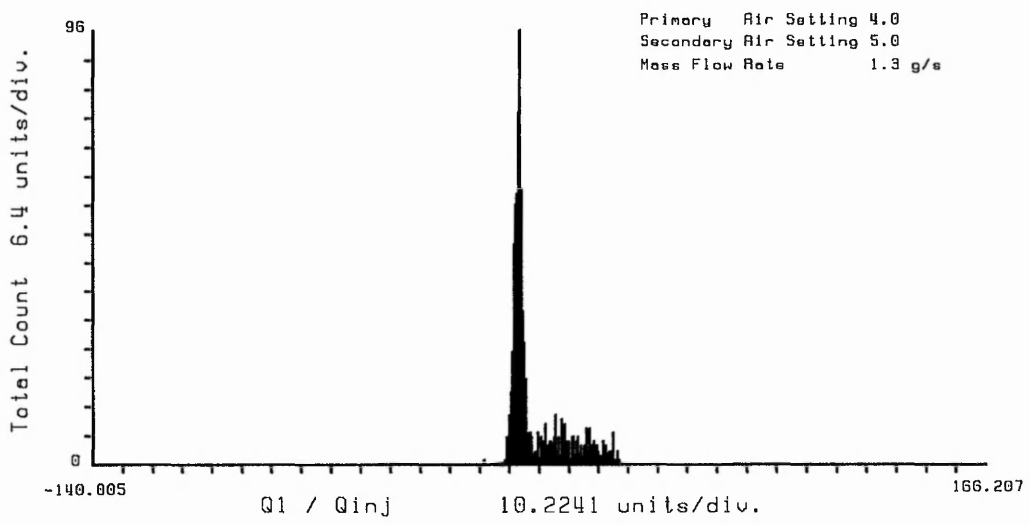

Figure 37 Data Distribution after Second Pass with 'clear' Results.

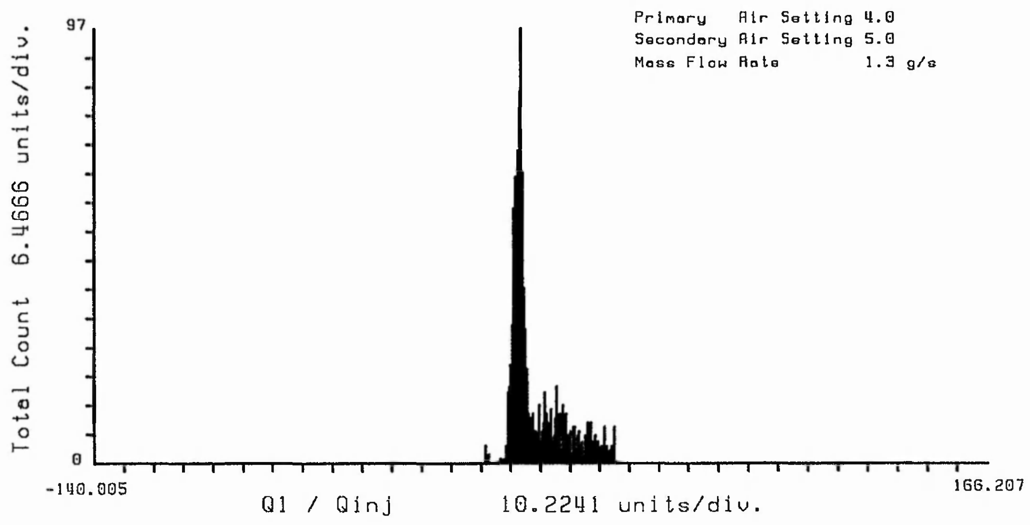

Figure 38 Data Distribution after Second Pass with 'good' Results.

These two methods produce similar results except more results were included in the '**good**' results computation. The '**clear**' results were chosen for further analysis as they still yield a high percentage of readings included in any one final resultant data item. 198 experimental results were obtained for further analysis. Using the first method of one result giving 900K of information, the data gathered would run into the hundreds of megabytes. A decision was made to only return the statistics of the unprocessed data and the statistics and data distribution of the filtered data. The statistics were later compiled into one large database from which all further analysis and results were obtained, using the filtered data from the database. The data distribution information still remains as 198 different distribution files.

More programs were written to cater for this new form of database and to allow extraction of the information by criteria defined by the operator. A graphical package was written for the Apollo which allowed the drawing of the following graphs extracted form the data.

The first stage in analysis of this data was to extract the main readings and plot them against various other readings, mainly against mass flow rate and computed velocity.

Figures 39, 40, 41, 42 and 43, starting on page 154, show sensor currents I_1, I_2 vs mass flow rate, I_2/I_1 vs mass flow rate, I_1 vs velocity and Q_1 vs mass flow rate respectively.

Results.

The characters used on all these graphs represent primary air settings. For each primary air setting the secondary varied from 0.0 to 9.0 in steps of 0.5. Both primary and secondary air settings are in arbitrary units. The characters on the graphs are as follows:

Primary Air Setting (Arbitrary Units)	Letter
2.0	E
2.5	F
3.0	G
3.5	H
4.0	I
4.5	J
5.0	K
5.5	L
6.0	M
6.5	N
7.0	O

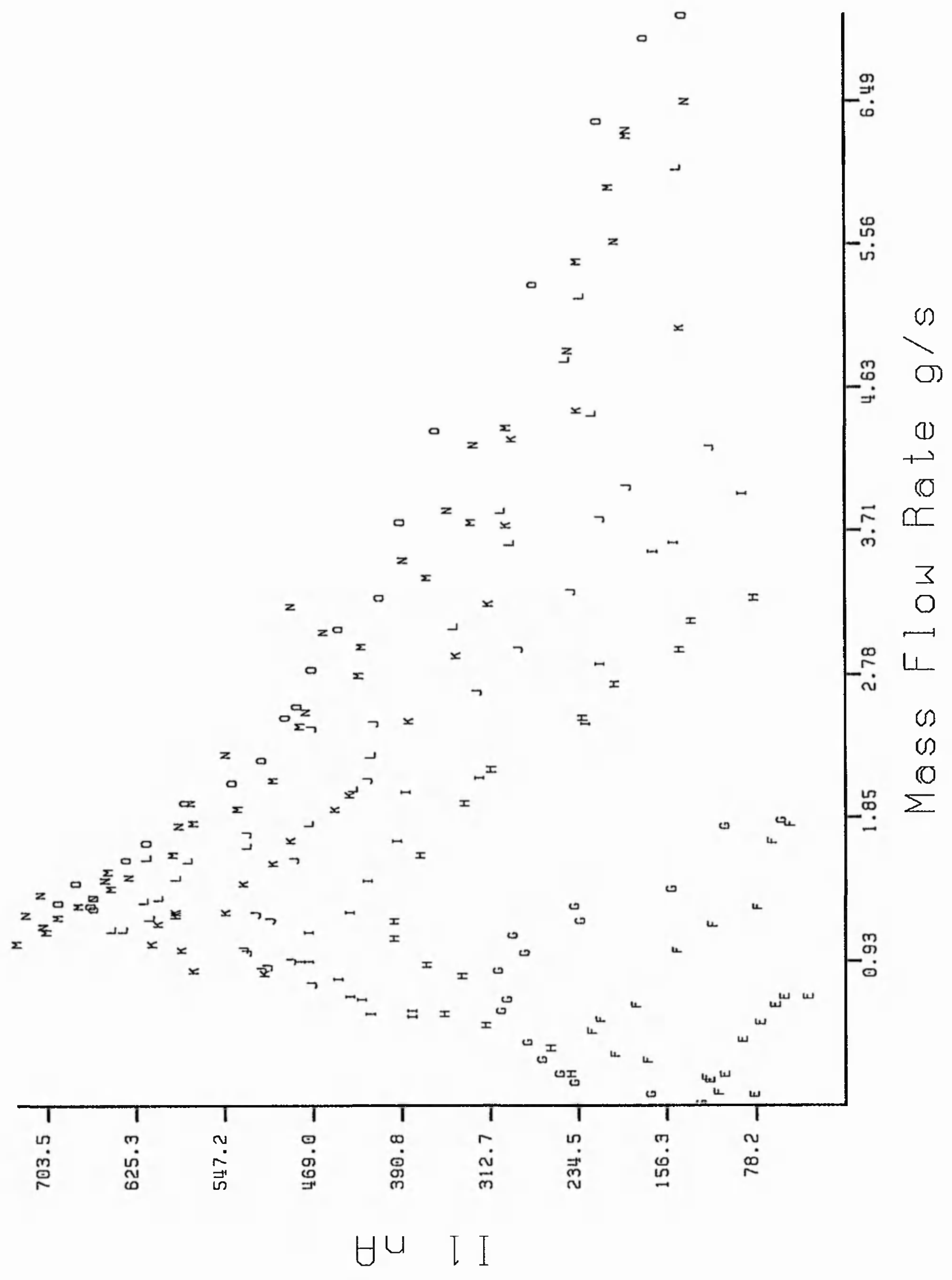

Figure 39 I1 vs M.F.R

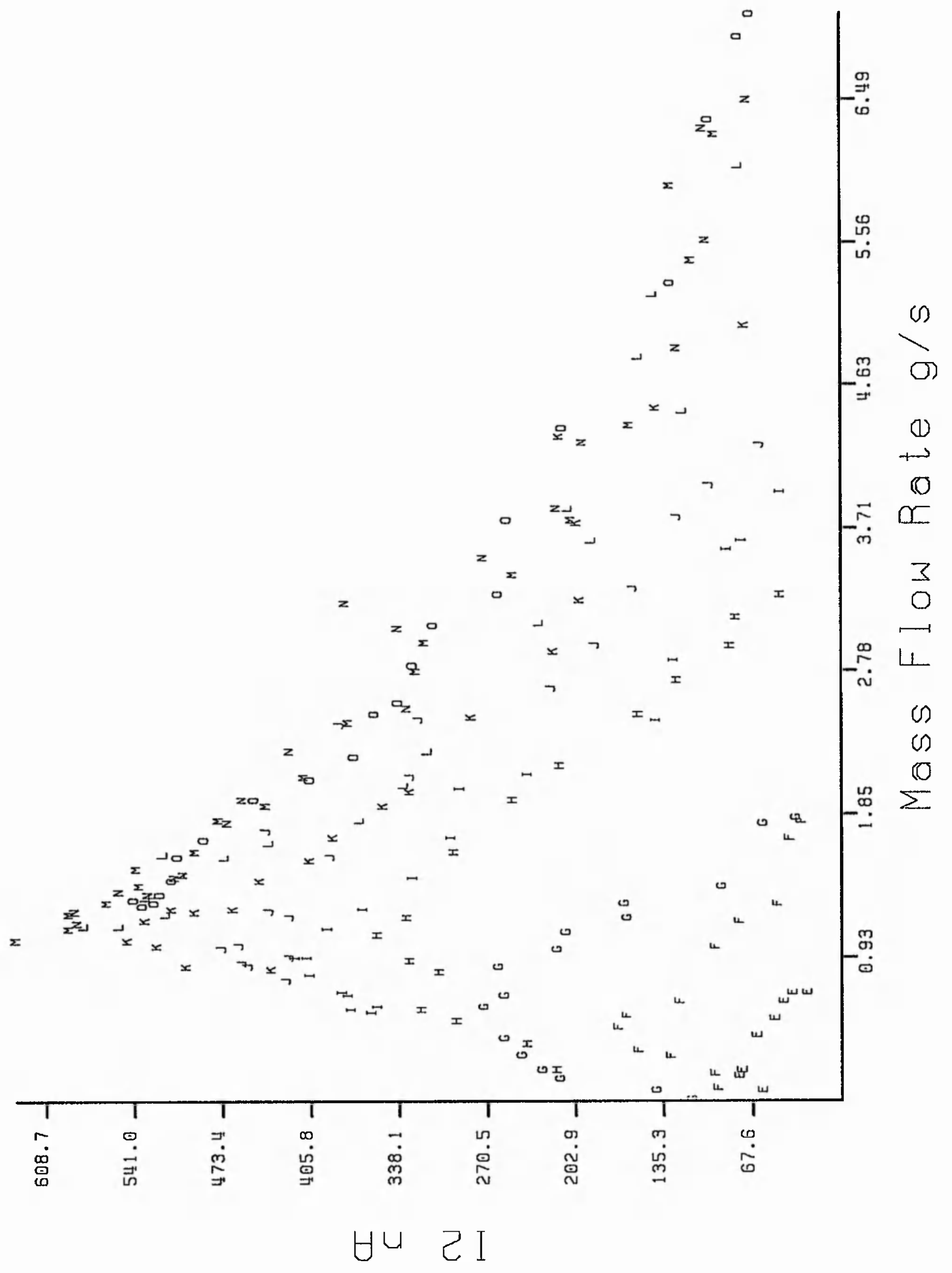

Figure 40 I2 vs M.F.R

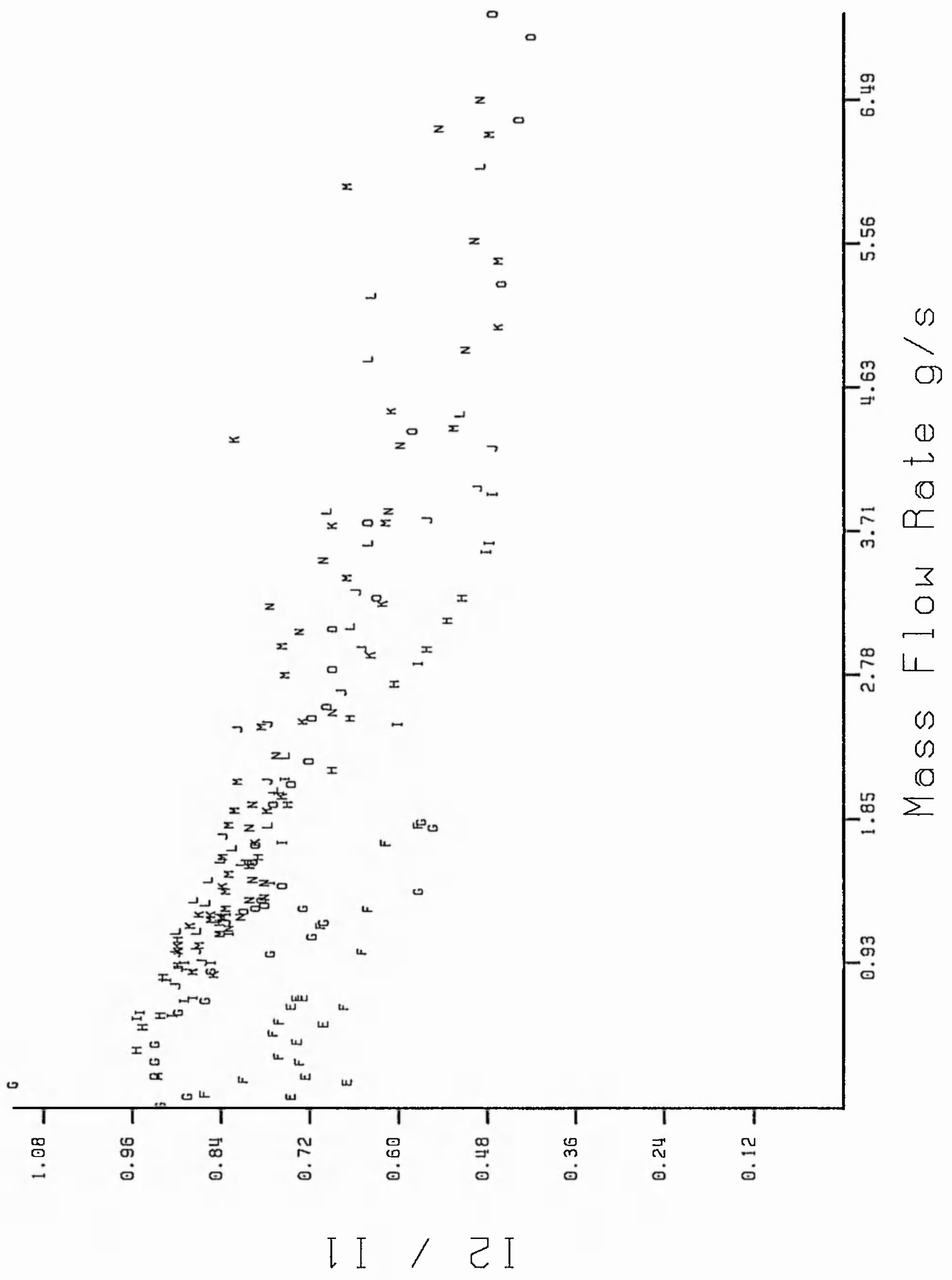

Figure 41 I2/I1 vs M.F.R.

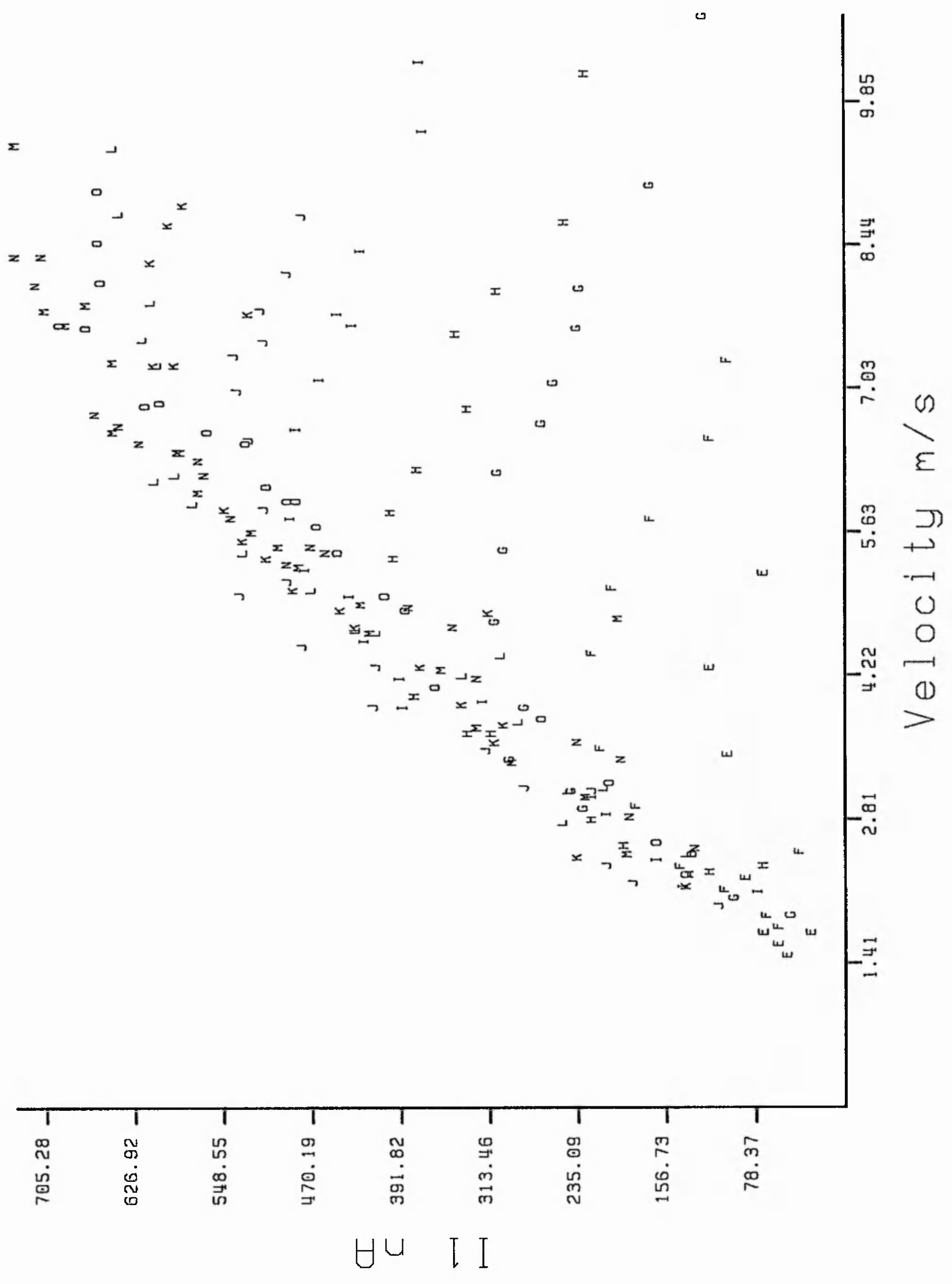

Figure 42 I1 vs Velocity.

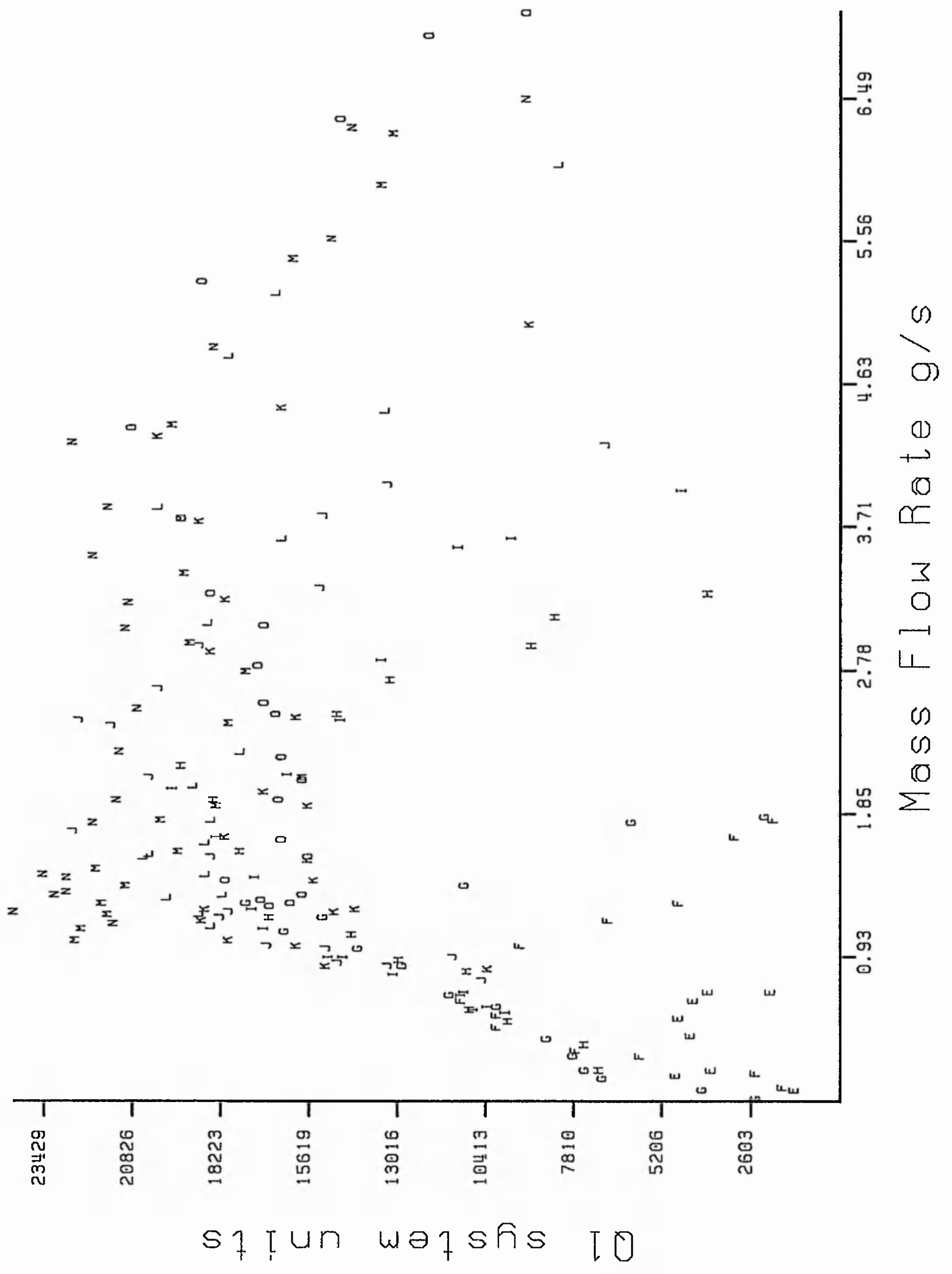

Figure 43 Q1 vs M.F.R.

9.3 Analysis.

Having collected the data and displayed it, analysis took place. It was felt that due to the complex patterns in the waveforms, parameters found within the readings would be used to generate regions of flow for which a single parameter can be computed relating the sensor readings to the flow rate. By the selection of various criteria from the readings it was possible to divide the results into regions. What follows is three such methods in which the data could be divided. It is possible that further analysis at a later date could yield a different set of results, but the first two methods here are sufficient to yield a mass flow rate to within ±15% of full scale within the regions defined in the analysis method.

9.3.1 Analysis I.

Analysis of the plots produced led to the generation of a function for the determination of mass flow rate for high velocities. Figure 44 shows the plot of the parameter calculated against mass flow rate. This was defined as region 1 and to obtain it readings have to meet the following criteria:

$$\text{Velocity} > 7.5 \text{ ms}^{-1}$$
$$I_1 < 650 \text{ nA}$$
$$Q_2 < 26000 \text{ units}$$

and the parameter generated was $MFR = (I_2 - 111.6) / 61.7$ g/s.

Results.

It was found that the velocity was the key factor in the results and consequently all the subsequent graphs are displayed with labelling in lowercase characters created from the velocity information as defined below:

Velocity (ms^{-1})	**Letter**
3.50	a
3.83	b
4.17	c
4.50	d
4.83	e
5.17	f
5.50	g
5.83	h
6.17	i
6.50	j
6.83	k
7.17	l
7.50	m
7.83	n
8.17	o
8.50	p
8.83	q
9.17	r
9.50	s
9.83	t
10.17	u
10.50	v

The figure formats shall be either as above or in uppercase lettering as the first set of figures, e.g. as in figure 39 page 154, unless otherwise stated.

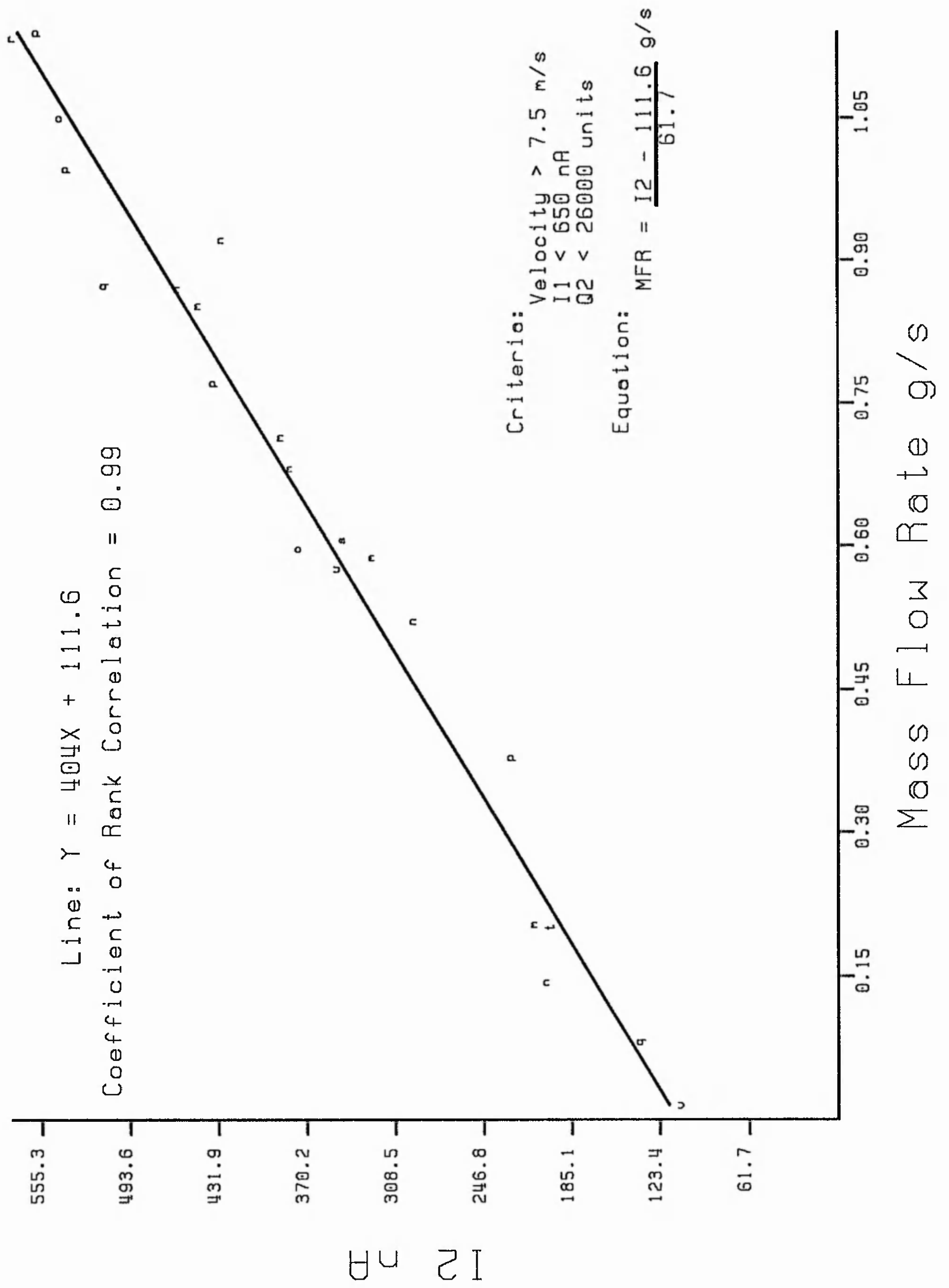

Figure 44 Region 1.

When looking for the next main region of flow only the residual readings from the first region were used. This is so that for computer selection of the regions the computer system can test the criteria for each region in turn and select the required computational equation to give the flow rate.

Looking at the sensor readings I_2/I_1 now left from region 1 after filtering off readings with either a velocity < 3.5 ms^{-1} or $Q_1 < 15800$ there is linear relationship between I_2/I_1 and the mass flow rate. This parameter alone could be used as an indication of mass flow rate. Figure 45 shows the parameter I_2/I_1 after data selection.

Taking the parameter I_2/I_1 and trying to slice it by velocity measurements gave some linear relationships between flow rate and I_2/I_1. Figures 46, 47 and 48 show some of these split regions and equations which can produce an indication of mass flow rate.

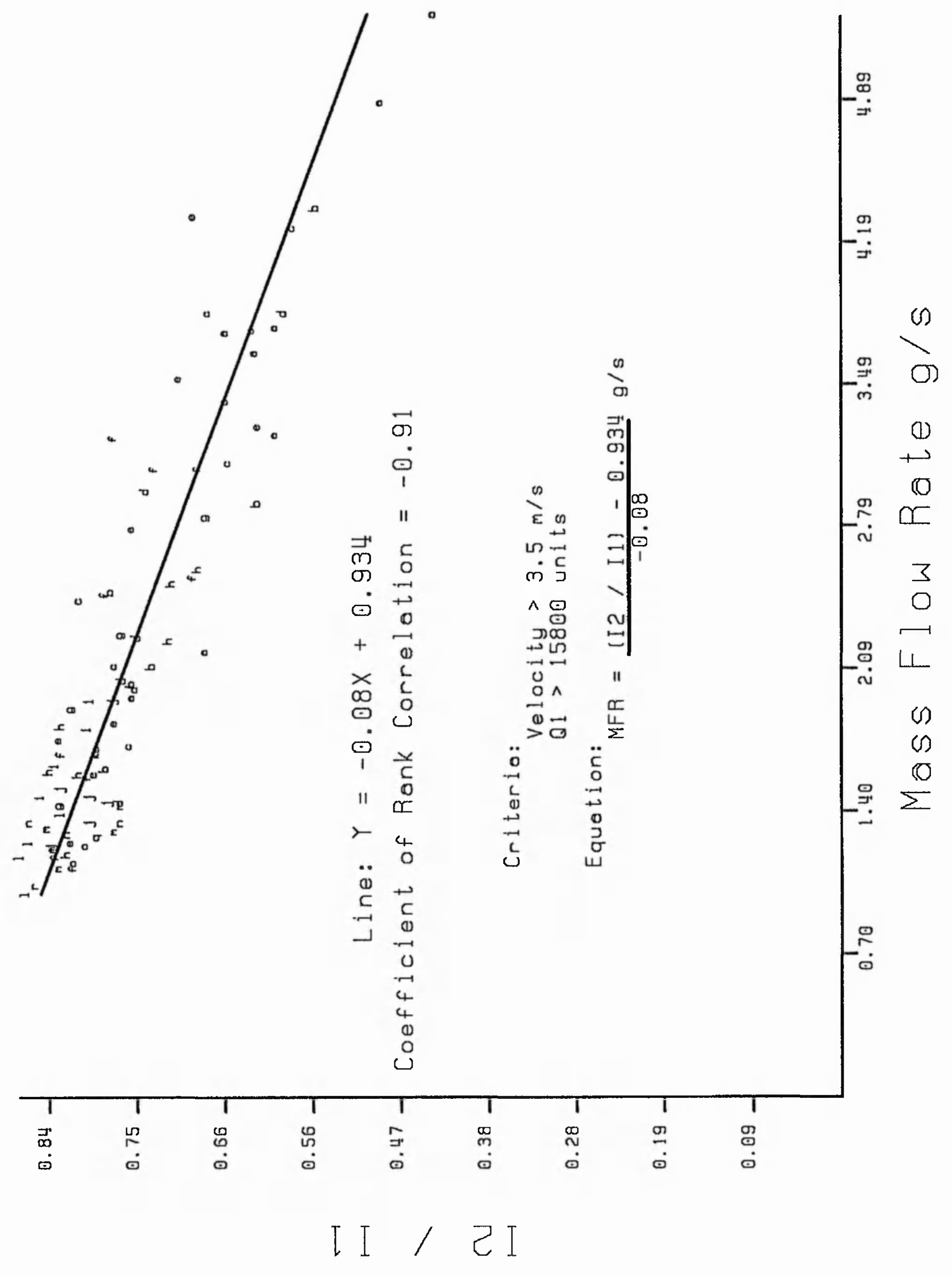

Figure 45 I2/I1 vs Mass Flow Rate after Selection.

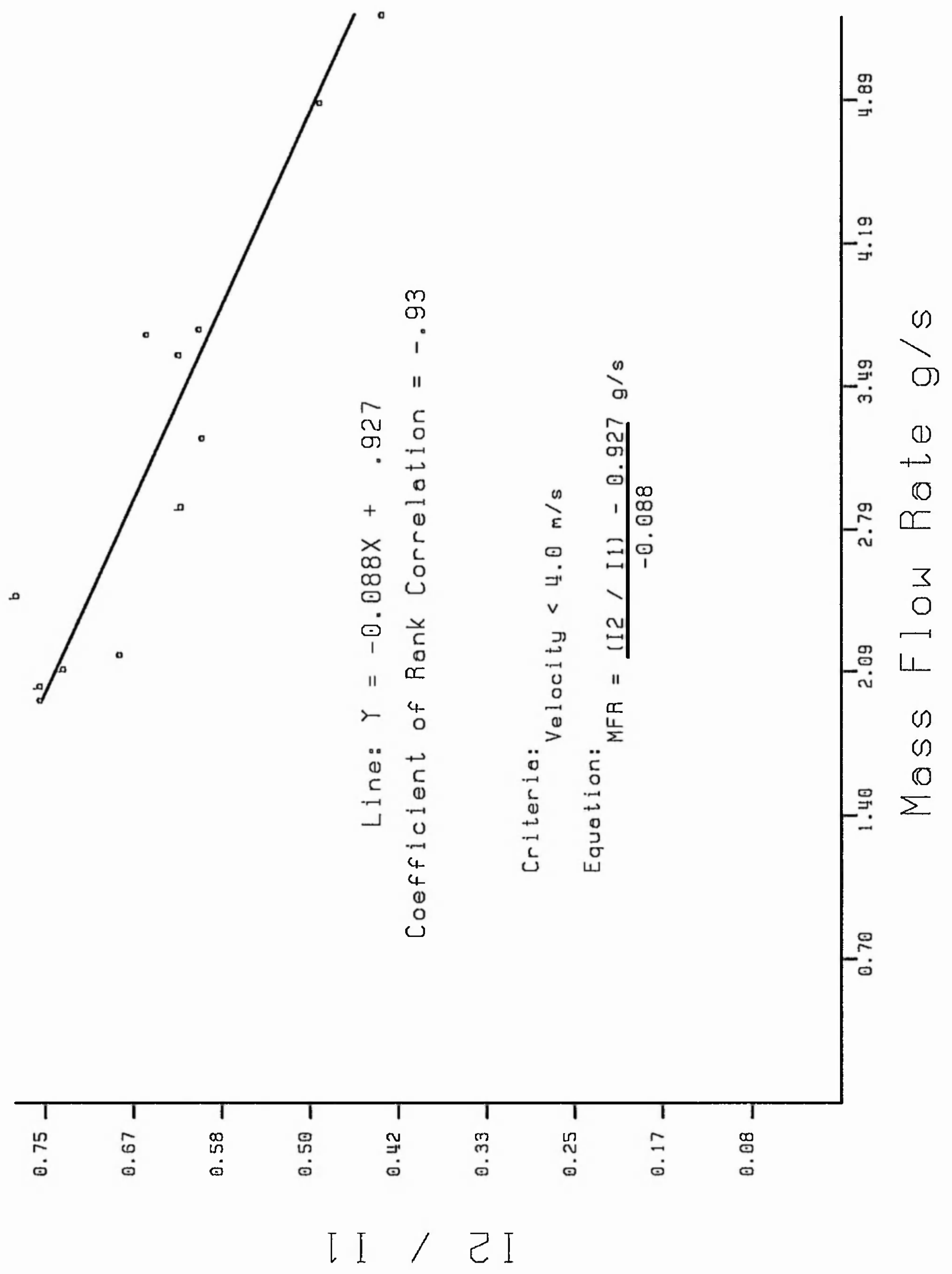

Figure 46 Region 2a.

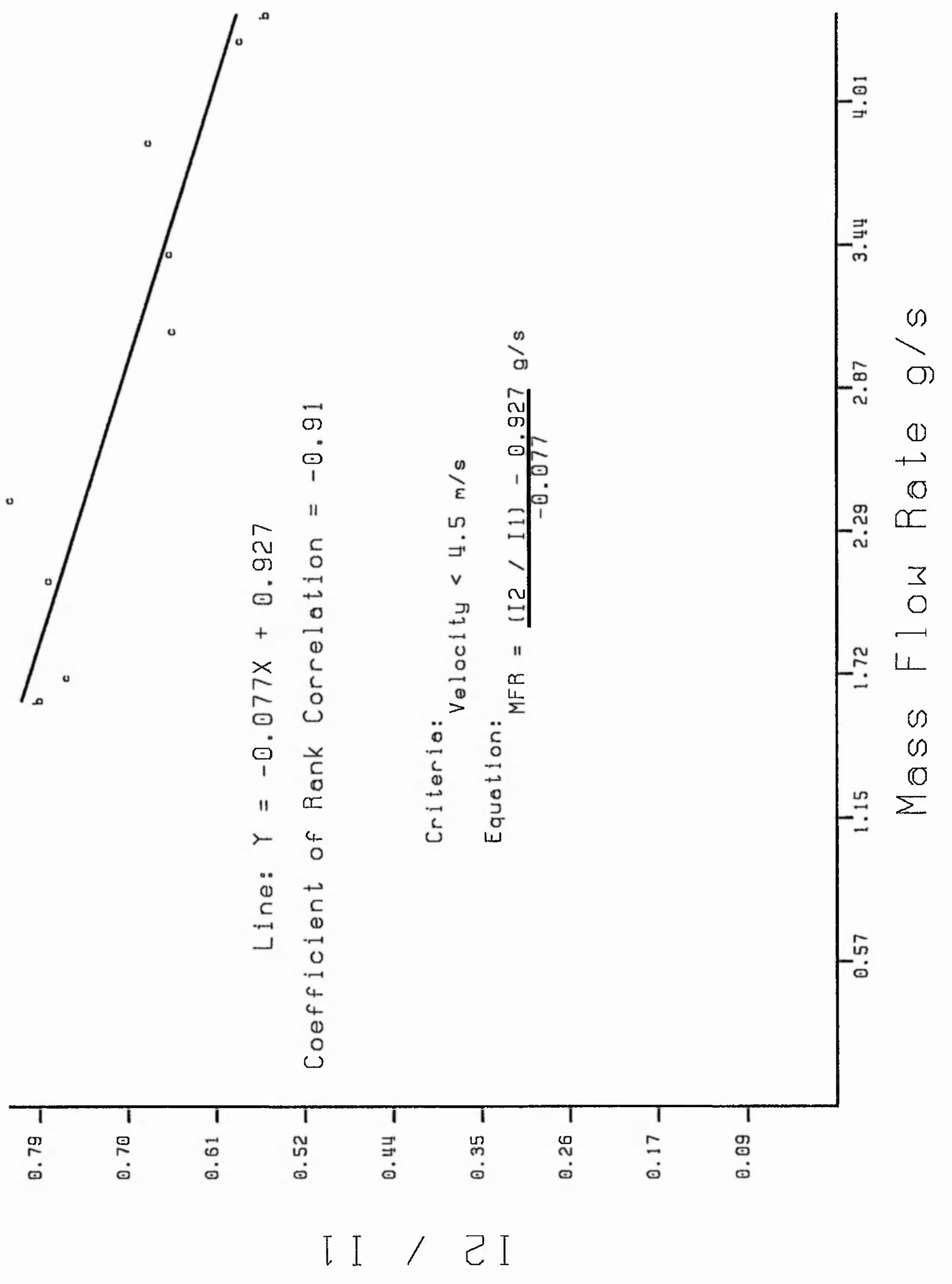

Figure 47 Region 2b.

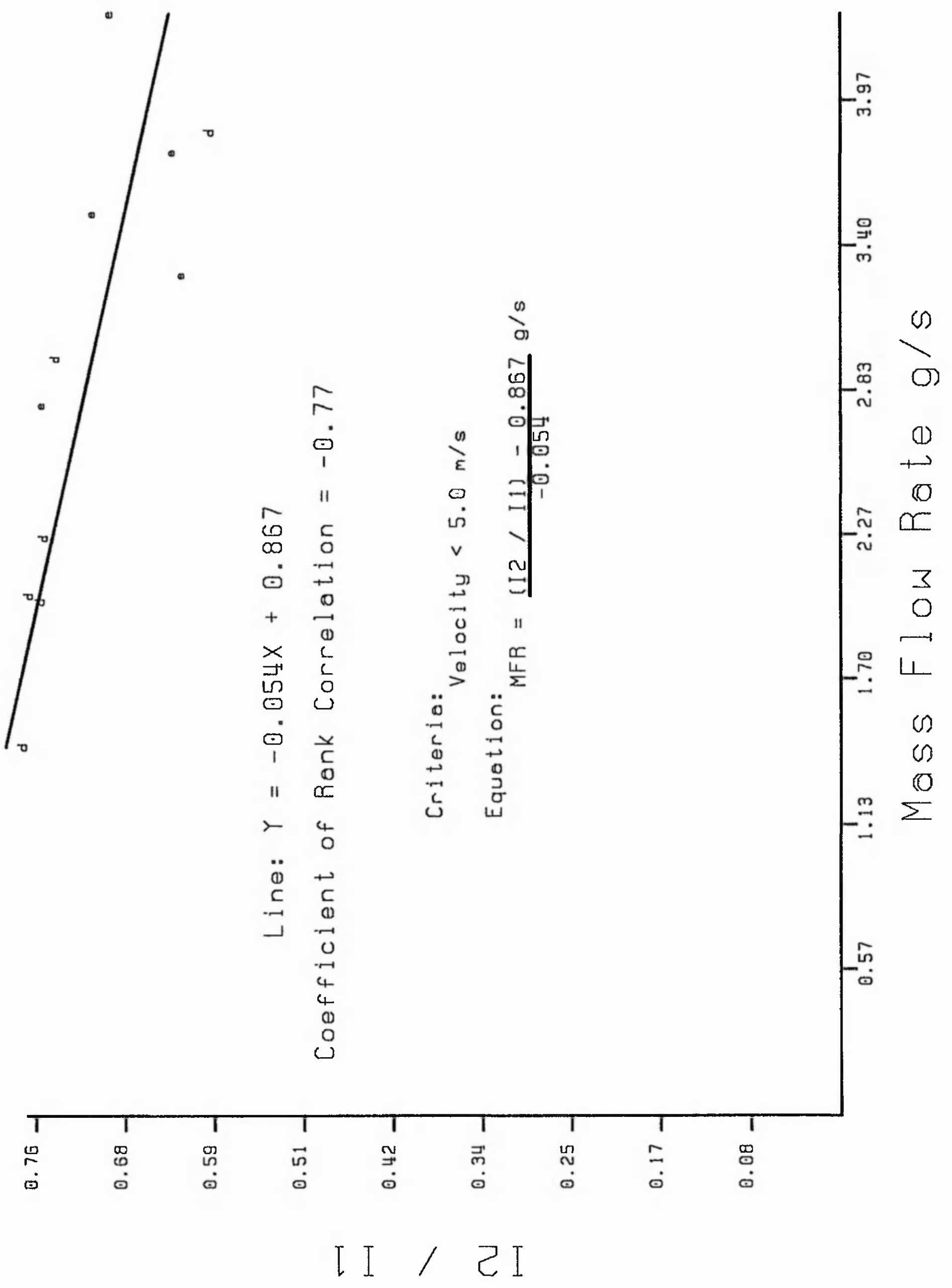

Figure 48 Region 2c.

9.3.2 Analysis II.

Since velocity was being used as a selection parameter in the previous analysis, functions based on velocity were then examined. Figure 49 shows the basic function of I_1/Velocity after all readings with a velocity of less than 3.5 ms^{-1} were rejected.

The rejection of readings with a velocity of less than 3.5 ms^{-1} was successful in removing spurious results where because of low velocity the powder was not properly entrained in the flow. With these low velocities pulsation of the powder occurred.

The next indicator was the ratio I_2/I_1. Under conditions of low flow rate, good entrainment of the powder into the air stream occurs. This yields a high I_2/I_1 indicating good correlation between the signals in the two adjacent sensors.

Figure 50 shows the separated region 1 with its equation and criteria.

Once region 1 is selected the remaining data was subdivided by various criteria as shown in figures 51, 52 and 53. This region is more difficult to explain why the various criteria yield the mass flow rate information due to the more complex flow situations, (i.e. turbulent flow).

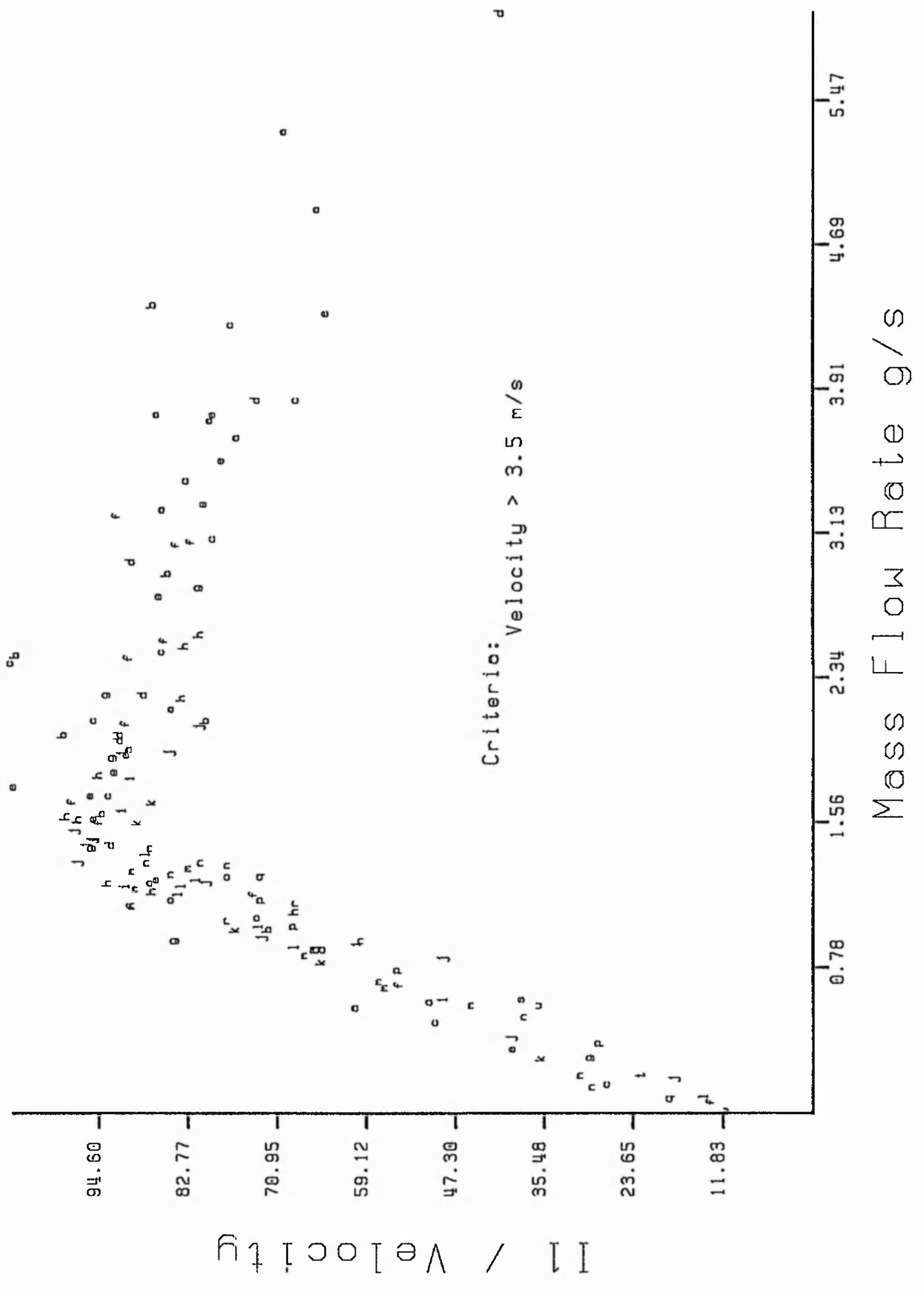

Figure 49 Il/Velocity (Velocity > 3.5 m/s).

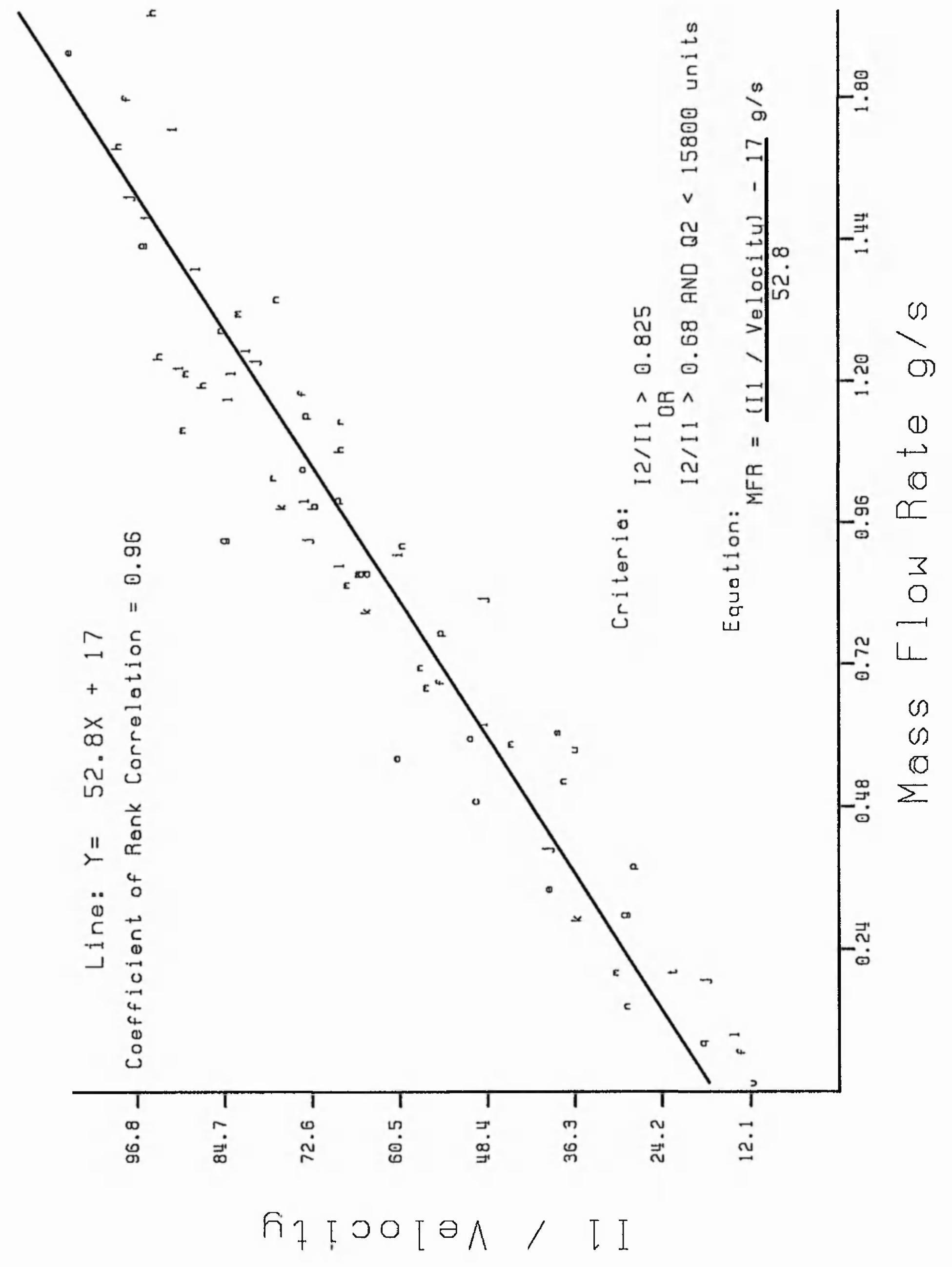

Figure 50 New Region 1.

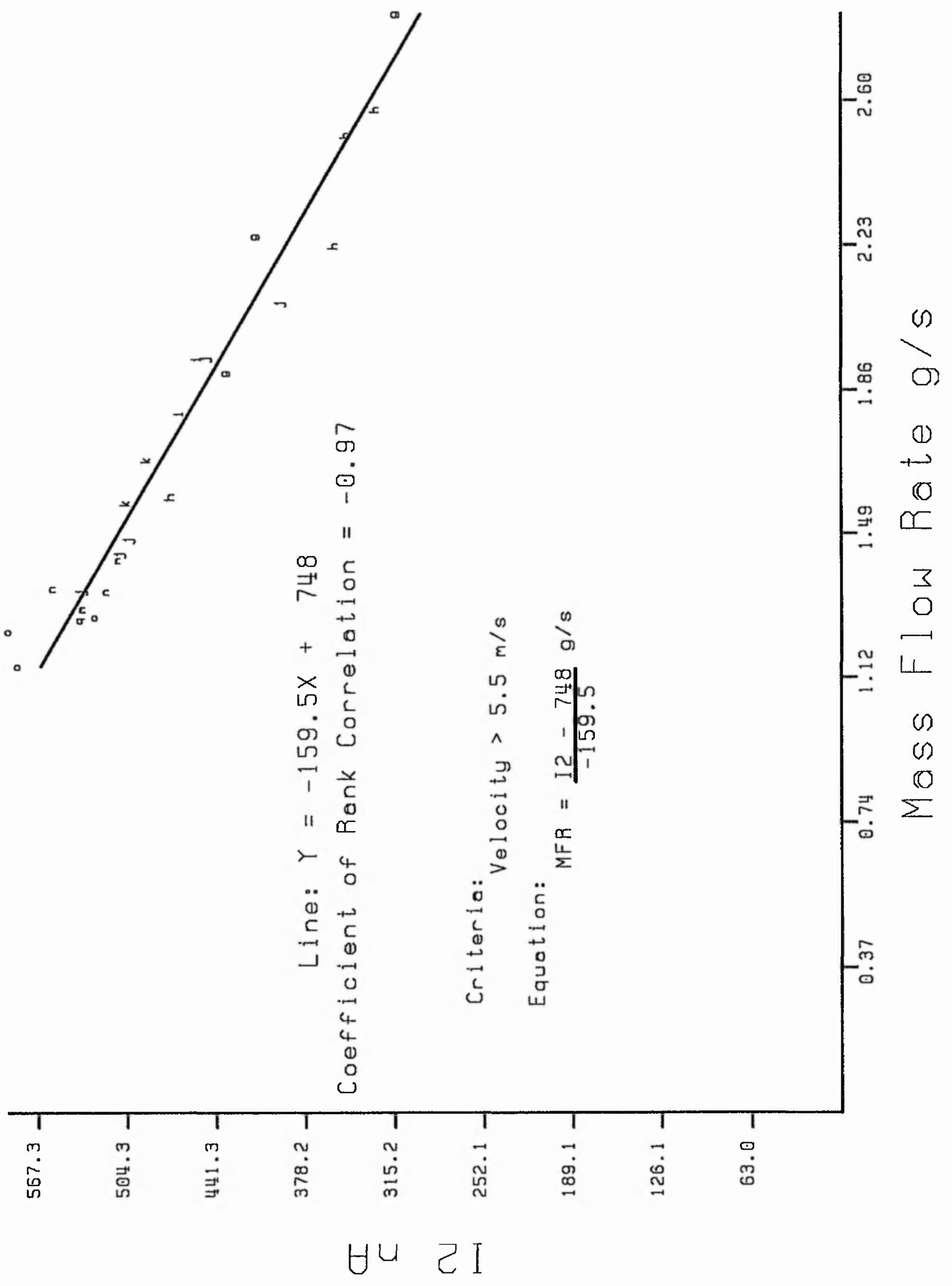

Figure 51 New Region 2a.

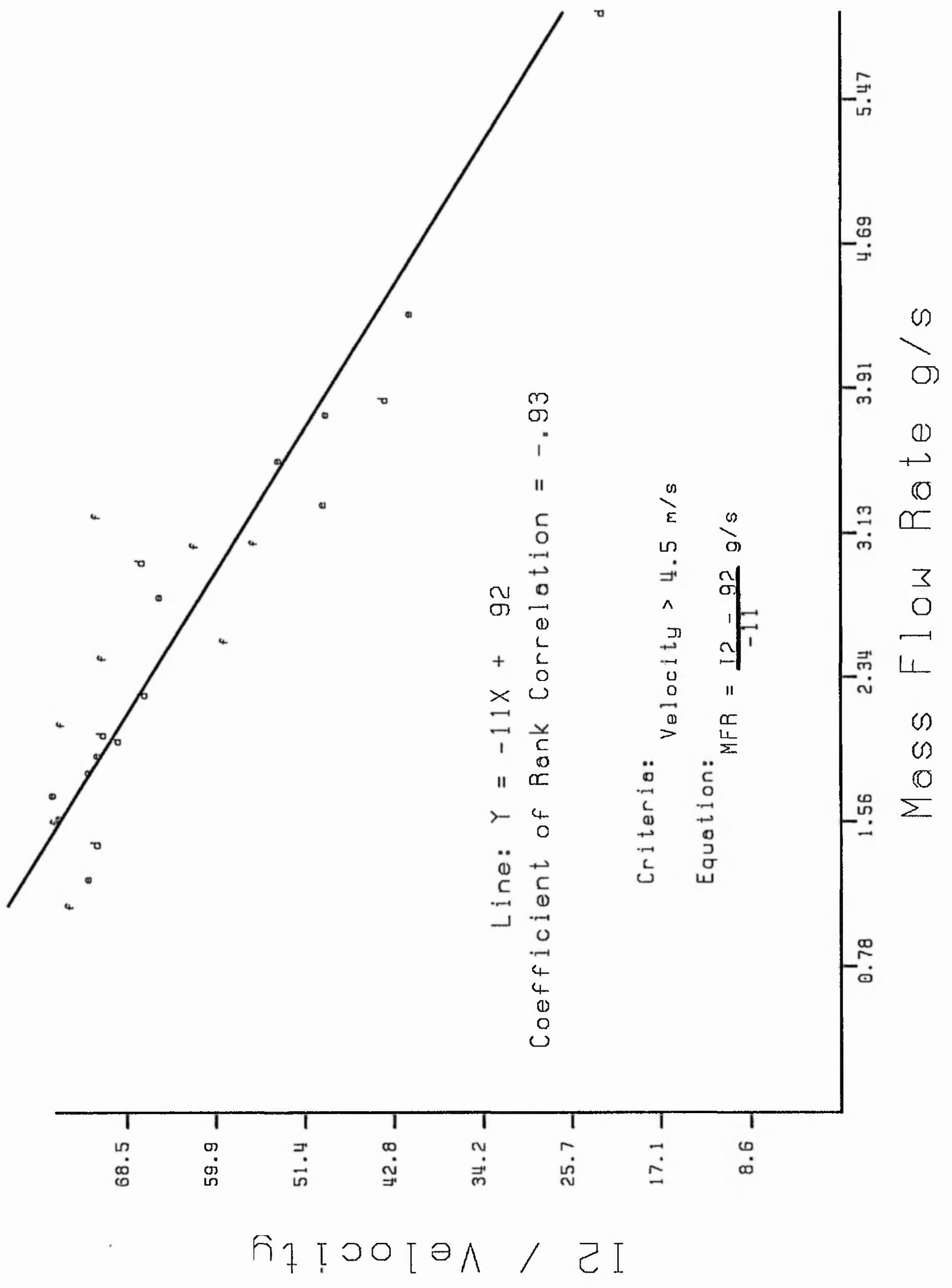

Figure 52 New Region 2b.

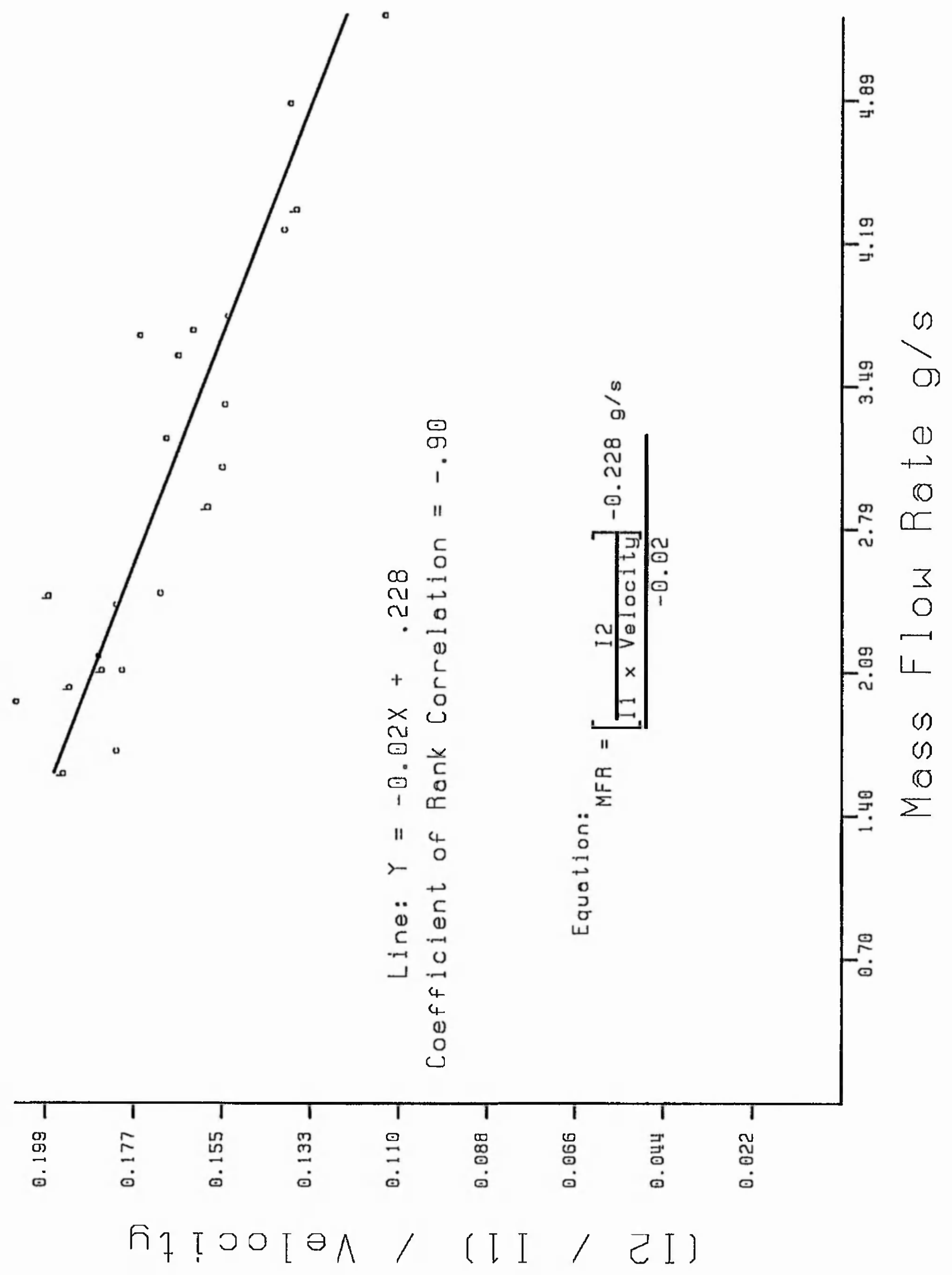

Figure 53 New Region 2c.

9.3.3 Analysis III.

A study of the data, using primary air values as a key to the display, shows a pattern of results for I_1 vs mass flow rate and I_1 vs velocity, figures 54 and 55.

Looking at the plot, for any given reading of I_1 there is a range of mass flow rates. Similarly for the same value of I_1 there is a range of velocities, but since the velocity information was derived by system measurements, the velocity is known and hence the mass flow rate can be determined. This has now given one complex model for the entire range of the system. Further study of these patterns could produce a look–up table of mass flow rates from the values of I_1 and velocity, which yield a greater accuracy than the previous methods.

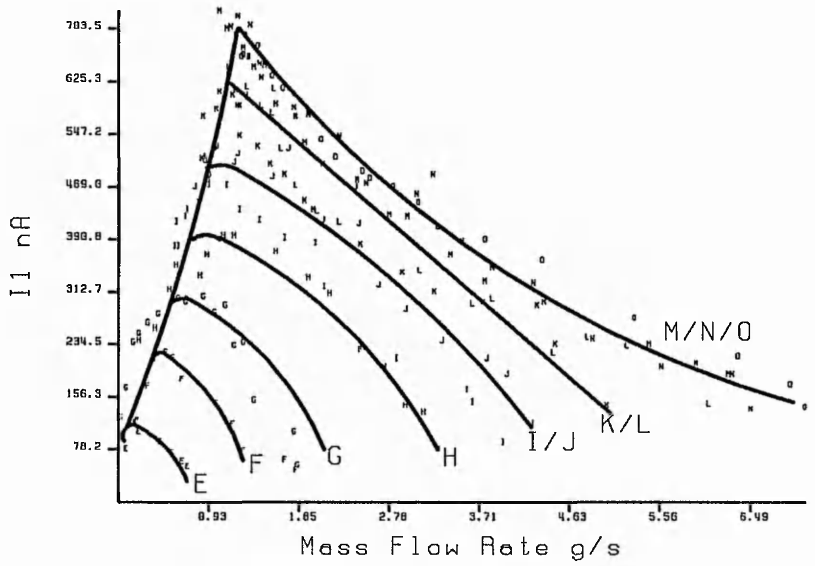

Figure 54 I1 vs M.F.R Patterns.

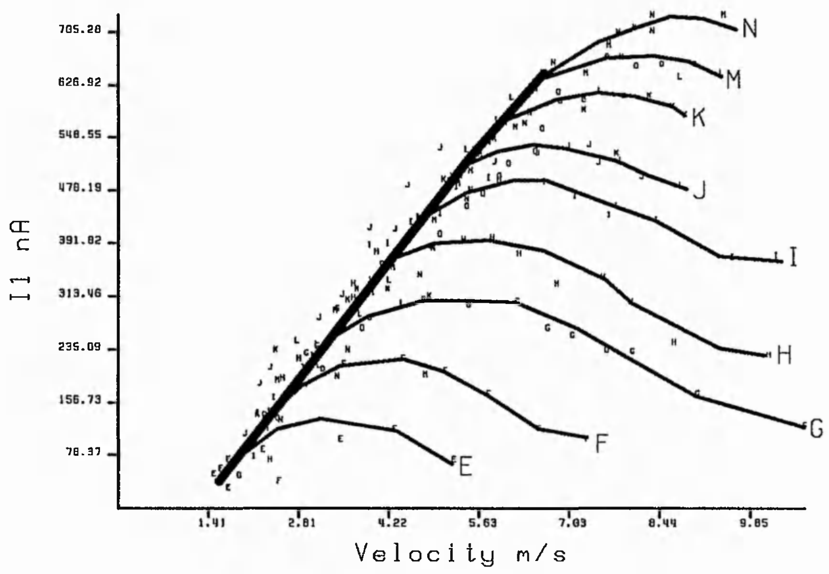

Figure 55 I1 vs Velocity Patterns.

9.4 A Change of Powder.

Another database has been compiled using a fresh white epoxy–polyester powder and one which had not been recirculated many times, as the black powder previously used. Looking at the readings obtained for I_1 against mass flow rate, figure 56, a similarity can be seen between that and those obtained using the black powder. The main difference seen is the lack of values under the main arch of the waveform. Operating with the same conditions as before, we obtain a line in region 1 with the white powder as with the black, figure 57, but of a different slope.

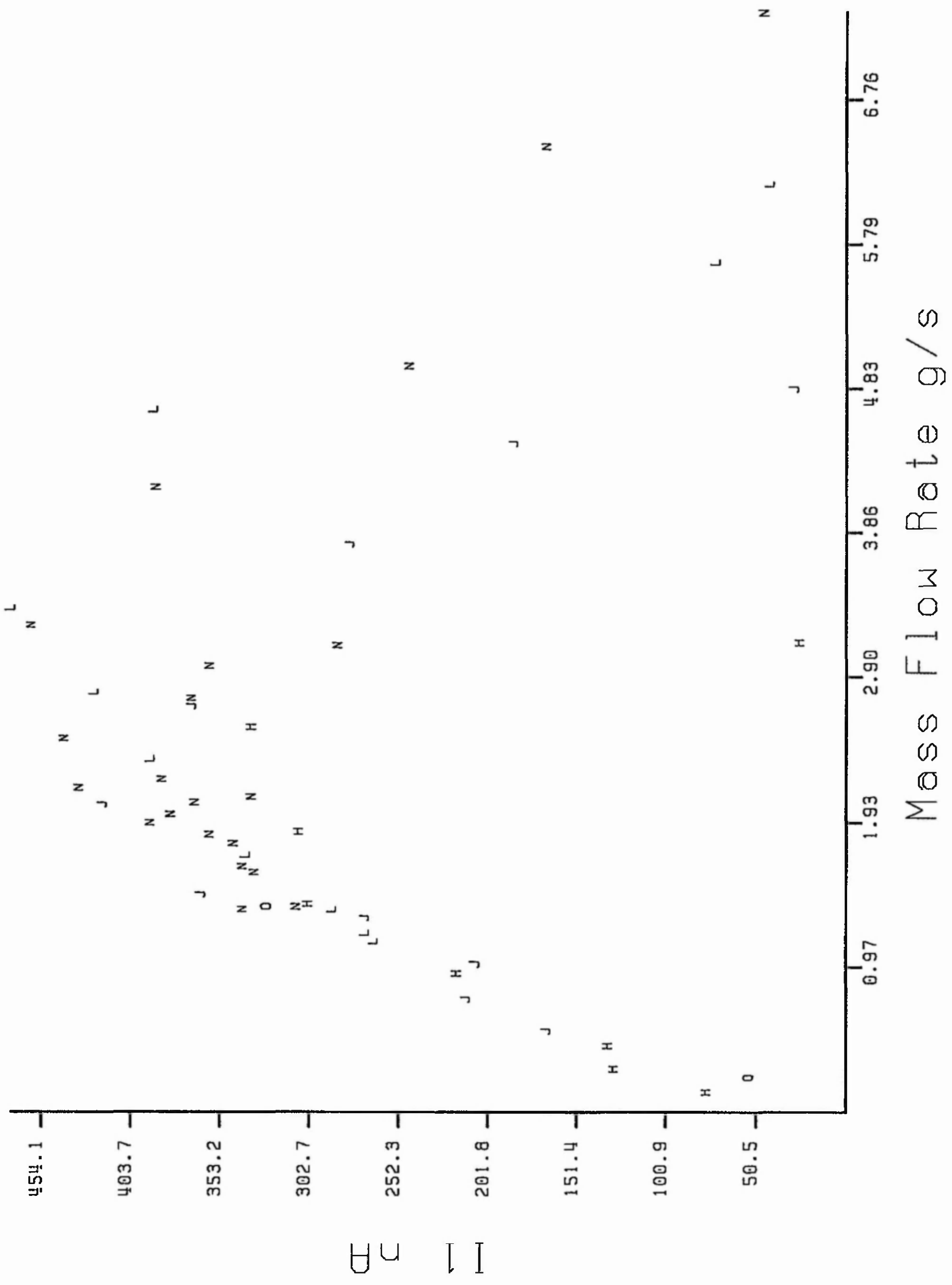

Figure 56 White I1 vs Mass Flow Rate.

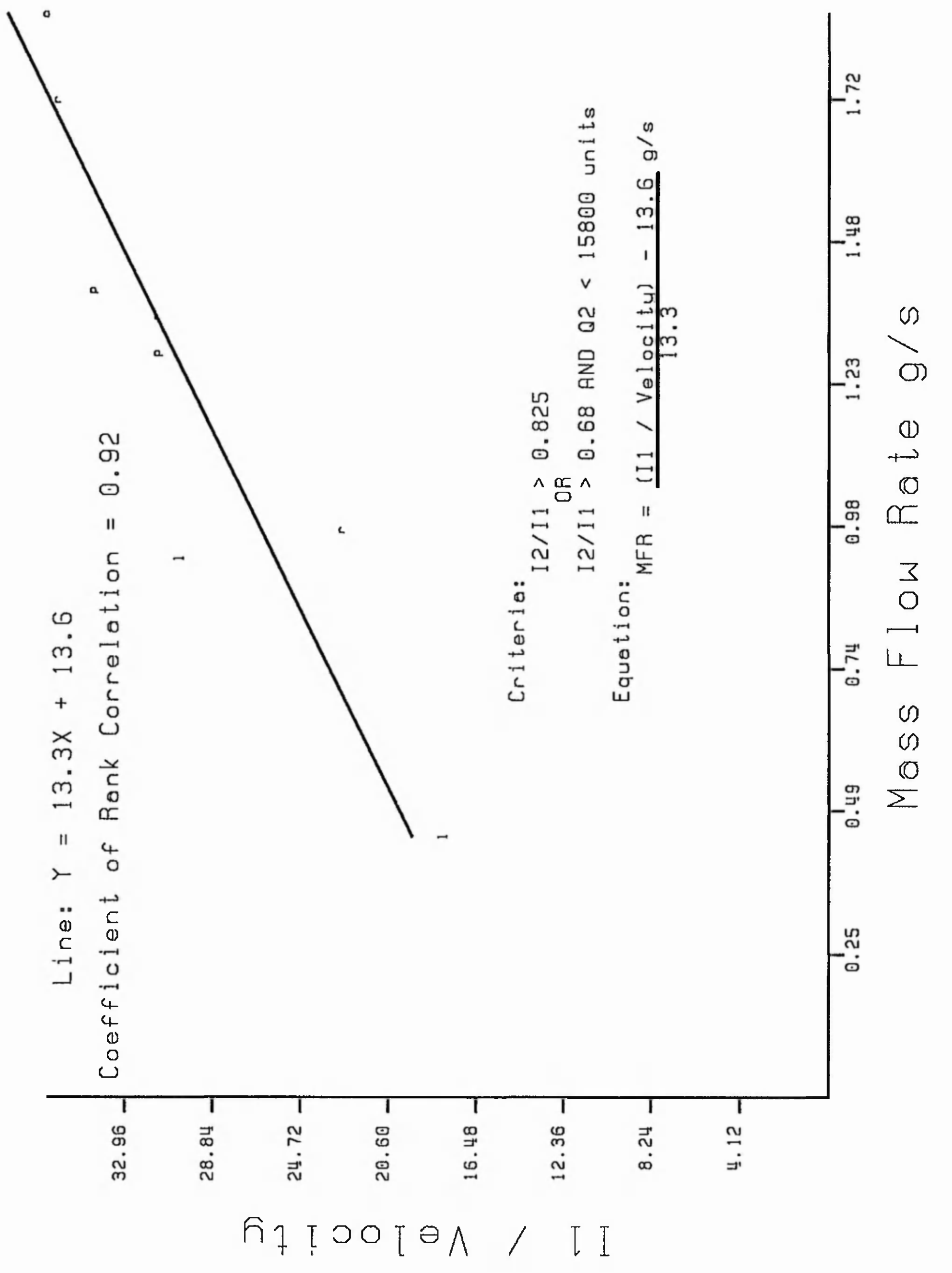

Figure 57 White Region 1.

Looking at the velocity distribution of the new powder it can be seen that all the readings obtained had a higher velocity than their black powder counterparts. This is due to the high concentration of fine particles within the powder which would have escaped from the black powder in the flow system, due to the many recirculations it underwent.

Figure 58 shows a 'typical' set of raw waveforms produced with the white powder.

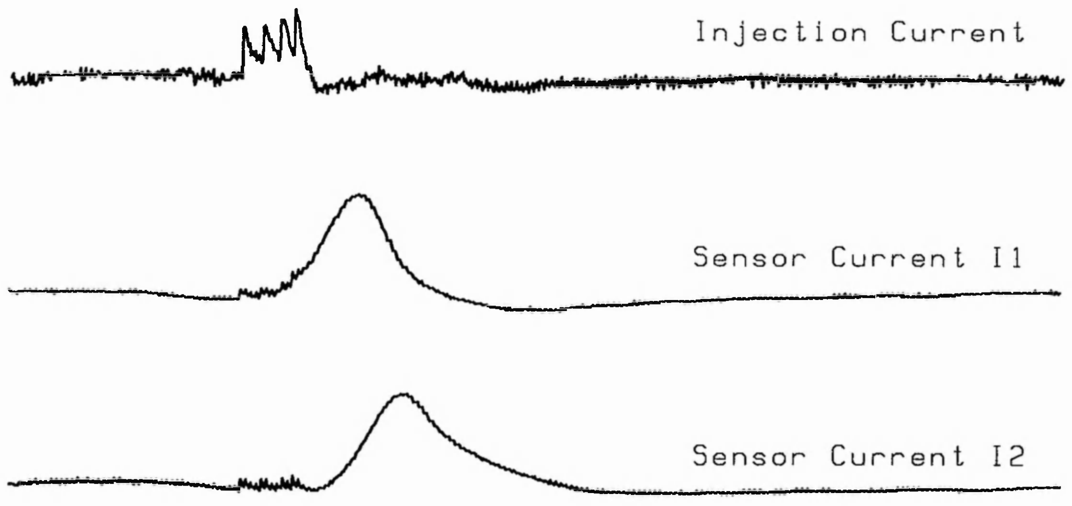

Figure 58 'Typical' White Waveforms.

All the waveform diagrams are shown as an indicator of their shape and therefore the calibrated units are not shown. All diagrams in this section however, have the same scaling factors. The injection current was taken using unity gain, I_1 and I_2 using overall gains of 2 and 4 respectively. The entire length of the waveforms represents 57.6 ms of sampled data.

Taking a closer look at the waveshapes obtained it is found that there are two distinct forms for the downstream sensor currents as in figures 59 and 60. The black powder, however, only has one main shape, and this matches neither of the white shapes, figure 61.

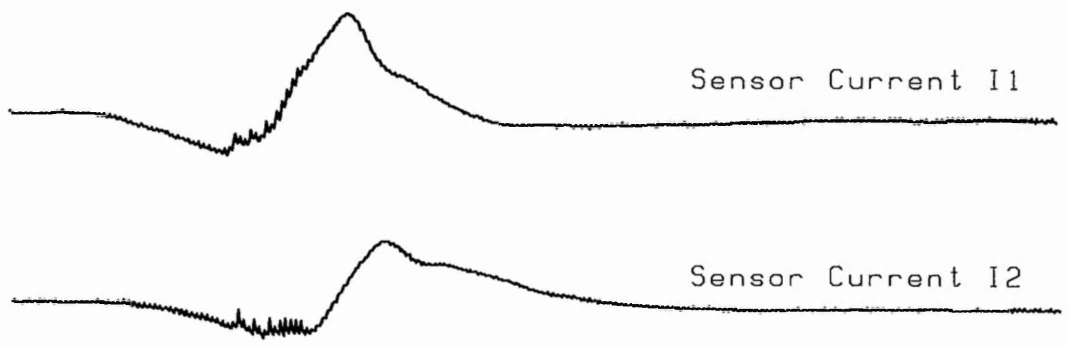

Figure 59 Downstream Sensors Waveforms. (White 1).

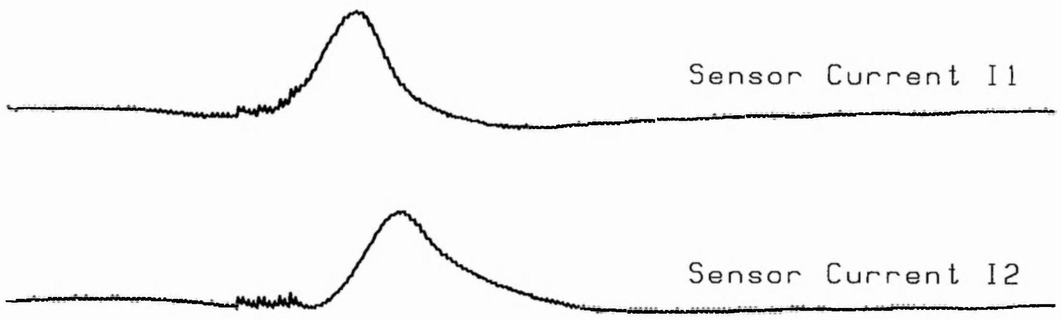

Figure 60 Downstream Sensors Waveforms. (White 2).

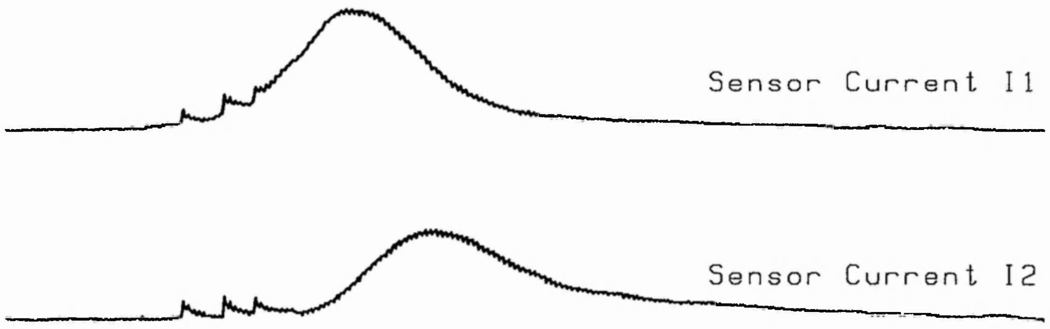

Figure 61 Downstream Sensors Waveforms. (Black).

What appears to be happening with the sensor currents is that with the white powder the finer particles which have been lost from the black due to recycling, are charged by the injector and transported faster than the heavier large particles. This causes a sharper wave edge on the currents and can also create narrower current shape.

Looking at the white waveshapes in figures 59 and 60, it can be seen that the second sensor current has a wider base than that of the first. This can be attributed to the particle distribution; the heavier particles moving slower than the light particles thus causing a stretching of the waveform. The waveform of the white powder can be considered as the current induced by the fine particles superimposed on that produced by the heavy particles as shown in figure 62.

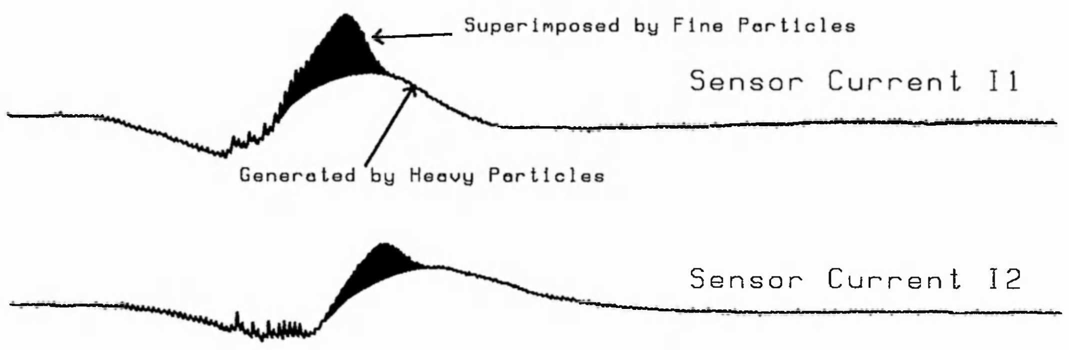

Figure 62 Underlying Trend of White Sensor Currents..

The fine particles have added a slight difficulty into the operation of the system. The rate of change of the injection current is not always great enough to detect injection using the current algorithm and hence some good injection passes can be missed. For further work the algorithm needs modification to allow the detection of injection under these new conditions.

Another feature seen with the white powder is that under conditions of no injection there is a negative trough, figure 63. This phenomenon reduces the overall level of the downstream sensor currents, as when injection does occur it has to cause the sensor currents to rise out of the trough, as in figure 60 page 179. This feature could be a factor in the generation of the different slope in the equations relating the induced sensor current to the mass flow rate, for the two powders studied.

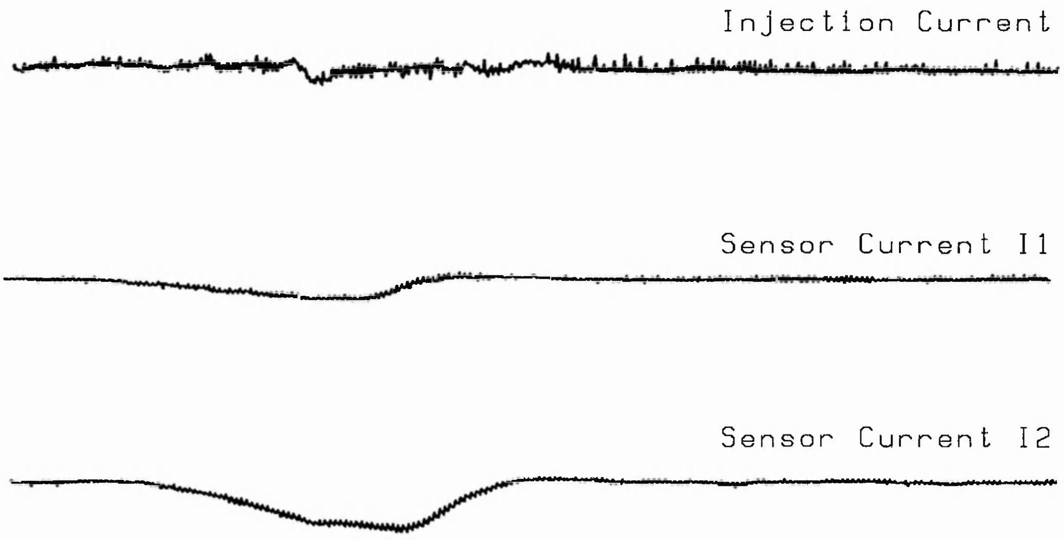

Figure 63 White Waveforms with no Injection.

With further analysis, a more complex solution for the generation of mass flow rates, as suggested in section 9.3.3, could be generated along with a model for the determination of particle size distributions.

10. Conclusions and Future Developments.

Starting from the basic technique of pulse charge injection as outlined by Willis [1], several types of computer system have been designed to tackle the problem of building a real–time instrument to measure the mass flow rate of pneumatically transported powder particles. The work has covered a wide range of topics from computer system design to using modern CAD techniques.

The design chosen for the implementation used the latest INMOS parallel processing devices, transputers. This type of processor has proven to be ideally suited to this application. It is possible that the final multi–transputer design will be used commercially in a closed loop control system for electrostatic paint spraying. The design also has applications in the more general area of particle flow analysis.

The transputer system allowed the gathering of a vast database which was processed by the large amounts of support software generated for that task.

The results have shown that a successful high speed parallel processing system has been developed which can capture large volumes of data at high speed. The processing of these results has led to the generation of an empirical model for the measurement of mass flow rates in the range of 0.2 gs^{-1} to 6 gs^{-1}, with a wide range of velocities, from 3.5 ms^{-1} to 14.0 ms^{-1}. The method involves the separation of the readings so that they lie in a particular 'region' of flow, figures 50 to 53. These regions produce accuracies of ±15% of full scale of that particular region. This separation was performed in a fixed sequence by computer and could easily be performed by the on-line transputer system to produce a real-time mass flow rate measurement.

When changing to a new white epoxy there was a significant change in the sensor readings which has been attributed to the different particle size distributions of the two powders. Processing the patterns produced by the white powder by the techniques developed for the black powder, a similar relationship can be seen in the identified region 1. Although the two actual equations produced are not the same it is considered that with further data analysis a relationship between the two powder types could be obtained.

The problems occurring with a non-recycled powder is that the current method for injection determination finds it harder to detect the onset of injection as the fine particles in the powder have picked up charge and brought it into the sensor area earlier. The passing of this charge produces a smaller rate of change in the injection waveform which is significantly different from that of the black powder. Having a restricted particle size reduces the speed of the front edge of the injection waveform and so when the wavefront occurs there is a greater rate of change in the charge being transported over the sensor and so a greater rate of change in induced sensor current.

10.1 Future Developments.

The future developments is spilt into two sections. The first section deals with hardware spin-offs from the project which could be used generally. The next section is details on specific developments which concern the project instrumentation.

10.1.1 General.

The effects of the change in particle size should be investigated. By using calibrated particle sizes and analyzing the resultant waveforms, a relationship between the induced currents and particle distributions could be determined. The current work has indicated that a relationship could be extracted. As discussed previously.

Further analysis of the white powder without recirculation should enable the production of a new system which is less affected by the waveshapes. Analysis of the waveshapes themselves could also lead to the development of an additional feature of particle size determination.

10.1.2 Modification of the Data Acquisition Unit.

There are two main ways in which the Data Acquisition Unit can be modified. The first is by creating three separate modules for the various sections. The first module would contain the multiplexers for multiplexing one system's signals down to one. This board would be identical to the main system multiplexer unit which should contain eight such boards.

After the multiplexer unit comes the new gain section which should be in a separate screened box to reduce the influence of externally induced signals. This gain section should be of the type detailed in section 6.2.2 with reference to figure 21.

The final module should contain the '**Flash Converter**' and transputer control logic. This module, like all the rest, should have its analogue data paths routed using screened coaxial cable, again to reduce pick–up.

The next phase of modification involves the transputer which interfaces to the A/D. The processor could be replaced with an INMOS 16-bit transputer, the T212 or T222. This will have the effect of not only reducing the cost of the unit, but in fact it will also increase the speed of operation. The speed increase stems from the non-multiplexed address and data bus of the T2 type products. This reduces the cycle time for external memory accesses by one clock cycle, (a reduction of 50 ns for a 20 MHz transputer).

A decision now has to be made on the actual gain computation element. In the original transputer system the gain computation was done by EPLD for intra-sample adaptation. This has the effect of increasing the bandwidth requirement of the serial data paths as it doubles the amount of data being transmitted down the links. The intra-sample adaptation could be provided as one option available on a more general acquisition type board. This board may also include extra memory to add flexibility to other applications.

With the present system the EPLD could be replaced by a simple latch and the inter-cycle adaptation previously used could be kept. Replacing the EPLD will also reduce the cost of the board further.

10.1.3 Modification of the Processing Array.

One major way of modifying the Processing Array is to change the transputers for T2 type transputers, thus reducing the cost of the whole system. The difference in processing performance this will cause however, has not been fully evaluated. Currently large 32-bit summations occur in the array. Changing to a 16-bit processor and using 32-bit arithmetic will create a penalty cost in speed. This speed reduction however, may be slight and could probably by traded in some way by the use of larger data buffers or other optimization techniques.

Another change in the Processing Array involving no new hardware, would be to change the current T414 transputers for T425 transputers which have 4K of RAM not the current 2K. This change would allow the development of more software on the array without increasing the complexity of the circuit board.

A study of the signals required to make up the equations for the mass flow rate could yield a reduction in the number of computations required to produce the same results. The increase in speed made could then be traded for a reduction in the number of processors.

10.1.4 Modification of the Controller.

Once the system has been fully defined with the actual feedback control system, the memory requirements can be evaluated further. The large memory could be tailored to the application and some, if not all, of the control logic could be incorporated into the actual circuit board of the Controller making it more dedicated to the flow control task.

10.1.5 Hardware Spin–offs.

10.1.5.1 High Speed Adaptive Converter System.

A new design of converter system which is a culmination of all of the processes in one could be created and tested. The new design could take on the role of any of the converters previously mentioned.

The basic idea of the converter stems from the original adaptive quantization unit which was designed in discrete digital devices, section 6.1.2. This converter adapted the gain of the converter system by calculating the gain for the next sample on that particular signal based on the current sample. This in fact tracks the signal with a gain. The first version used logic to decide on the new gain. This gain was then stored in a high speed RAM.

In this new design all the logic of the original converter is replaced by high speed static RAMs. The RAMs would be preprogrammed with a look–up table containing all the information needed to generate the gains and their adaptation. The final output stage would also be replaced by high speed RAMs and after the calibration the look–up tables could be modified take into consideration the actual system parameters found.

By using high speed RAMs different types of adaptation technique could be encoded into them.

References.

1. C.A.Willis.

Continuous Mass Flow Rate and Velocity Measurements of Pneumatically Conveyed Powder.

PhD thesis of Trent Polytechnic Nottingham 1984.

2. B.C.O'Neill and C.A.Willis.

Corona Charging of Pneumatically Conveyed Powders.

Journal of Electrostatics June 1985.

3. M.S.Beck and A.Plaskowski.

How Inherent Flow Noise can be used to Measure Mass Flow of Granular Materials.

Instrument Review Nov 1967, pages 458-461.

4. M.S.Beck, J.Drane, A.Plaskowski and N.Wainwrwight.

Particle Velocity and Mass Flow Measurement in Pneumatic Conveyors.

Powder Technology Vol. 2. 1968/1969, pages 267-277.

5. M.S.Beck, J.H.Hobson and P.J.Mendies.

A Mass Flowmeter for Airborne Solids.

Proceedings of Powtech 71; International Powder Technology and Bulk Granular Solids Conference, pages 63-65.

6. R.G.Green.
A Frequency Modulated Transducer for Gas/Solids Flow Measurement.
Proceedings of IMEKO VII Practical Measurement for Improving Efficiency Conference, London 1976, Vol. BFL 251A, pages 387–391.

7. M.A.Cardon, R.G.Green and R.John.
Applications in Monitoring Particulate Mass Flow Rates of Abrasive Materials.
BHRA Conference 1981. Advances in Flow Measurement Techniques.

8. R.G.Green, S.H.Foo and R.Thorn.
Microcomputer–Based Mass Flow Rate Measurement of Solids in a Pneumatic Conveying System.
The Arabian Journal for Science and Engineering, Vol. 7. No. 4, pages 411–418.

9. H.K.Kwan and M.S.Beck.
Mass Flow Measurement of Solids in Gravity Conveyors using Capacitance and Ultrasonic Transducers.
Proceedings of the &th IMEKO Cong., pages 379–385, 1976.

10. P.W.King and P.J.Hewitt.
Monitoring of Electrostatic Charge Transported by Pipe Flowing Dielectrics, using Intrinsic Electrical Noise Signals.
Proceeding of the 142nd Meeting of Electrochemical Soc. Oct 1972.

11. P.W.King.
Mass Flow Measurement of Conveyed Solids, by Monitoring of Intrinsic Electrostatic Noise Levels.
Pneumotransport 2, 2nd International Conf. on the Pneumatic Transport of Solids in Pipes, pages D2–9 to D2–20, Sept 1973

12. M.S.Beck and J.H.Hobson.
Electrostatic Charge Measurement of Particulate Materials being Transported at High Velocities – Explosion Risk Meter for Pneumatic Conveyors.
IEE Conference Proceedings, Aug. 1970, pages 38–41.

13. U.Mann and E.J.Crosby.
Flow Measurement of Coarse Particles in Pneumatic Conveyers.
Ind. Eng. Chem., Process Des. Dev., Vol 16, No. 1, 1977.

14. J.D.King and W.L.Rollwitz.
Magnetic Resonance Measurement of Flowing Coal.
ISA Trans. Vol. 22, pages 69–76. 1983.

15. M.M.Decker.
The Gyroscopic Mass Flowmeter.
The Engineers' Digest. Pages 92–93, July 1960.

16. J.P.Tullis and J.Smith.
Coriolis Mass Flow Meter.
NEL Fluid Mechanics Silver Jubilee Conf. Paper N° 6.3, 1979.

17. P.Kehler.
Two–Phase Flow Measurement by Pulsed Neutron Activation techniques.
Proceedings of ASME (San Francisco California), Dec 1978, pages 11–20.

18. T.Moriyama, S.Fujii, K.Abe, and M.Kobayashi
Mass Flowmeter Using Heat Transfer For Dense Phase Solid Gas Two Phase Flow.
Trans. Soc. Instrumentation and Control Engineering. Vol. 2 2, N°. 1, pages 104–108. Jan 1986.

19. F.R.G.Mitchell, J.M.Proctor and E.Turnbull.
An Optical Method of Measuring Particle Mass Flow Rates.
J. Physics E:Sci. Instrum., Vol. 17, pages 183–184, 1984.

20. Y.Morikawa, Y.Tsuji and T.Tanaka.
Measurements of Horizontal Air–Solid Two–Phase Flow using and Optical Fiber Probe.
Bulletin of JSME, Vol. 29, N°. 249, pages 802–809. March 1986.

21. V.P.Shorin, O.A.Zhuravlev, L.G.Logak, L.N.Medinskaya and A.I.Fedosov.
Holographic Apparatus for Study of Two–Phase Flows.
Translated from Pribory i Tekhnika Eksperimenta, N°. 5, pages 158–161. Sept–Oct 1985.

22. J.D.Trolinger.
Particle and Flow Field Holography. A Critical Survey.
Proc. SPIE Int. Soc. Optical Engineering. Vol. 532, pages 40–62 Jan 1985.

23. M.Robinson and S.C.Sood.
Real–time Depth Measurement in a Stereoscopic Television Display.
Proceedings of the 26th Annual Technical Symposium SPIE, Vol. 367, pages 34–40, 1982.

24. M.Robinson.
Novel Three–Dimensional Imaging Techniques Applied to X–Rays.
Proceedings of the Symposium on X–Ray Real–Time Radiography by British Inst. Non–Destructive Testing, Nov 1988.

25. B.C.O'Neill and C.A.Willis.
B.T.G. Solids Flow Meter Application.
4th July 1983

26. F.Trimaille and G.Bourguignon.
Feedback Control for a Powder Painting System.
Dossier Methodologique. Memoire de fin d'etudes ESTE 1986.

27. F.Trimaille and G.Bourguignon.
Feedback Control for a Powder Painting System.
Dossier Technique. Memoire de fin d'etudes ESTE 1986.

28. D.Lanfranchi and L.Goueta.
Feedback Control for a Powder Painting System.
Internal report Dept. Electrical and Electronic Eng. Trent Polytechnic July 1986.

29. W.J.Cody, Jr and W.Waite.
Software Manual for the Elementary Functions.
Prentice – Hall Inc.

30. Dr I.Logan and Dr F.O'Hara.
The Complete Spectrum ROM Disassembly.
Sections 'Exponential' and 'Natural Logarithm'.
Melbourne House Publications, pages 211–215.

31. Dr I.Logan and Dr F.O'Hara.
The Complete Spectrum ROM Disassembly.
Section 'Series Generator'.
Melbourne House Publications, pages 198–199.

32. Dr I.Logan and Dr F.O'Hara.
The Complete Spectrum ROM Disassembly.
Sections 'Reduce Argument', 'Cosine', 'Sine', 'Tan', 'Arctan', 'Arcsine' and 'Arccos'.
Melbourne House Publications, pages 215–220.

33. Inmos.
Transputer Reference Manual.
Prentice Hall.

34. E.Mills and Dr B.C.O'Neill.
Transputer Instrumentation Applied to Electrostatic Powder Flow Measurement.
Proc. 8th Technical Meeting of the Occam User Group. March 88.

35. E.Mills and Dr B.C.O'Neill.
Transputer Instrumentation Applied to Electrostatic Powder Flow Measurement.
Proc. 23rd Universities Power Engineering Conf. Sept 88.

36. E.Mills and Dr B.C.O'Neill.
Transputer Instrumentation. Software for Particle Flow Measurements.
Internal report Dept. Electrical and Electronic Eng. Trent Polytechnic 1989.

37. D.Lindsey.
The Design and Drafting of Printed Circuits.
Bishop Graphics Inc.

Appendices

A. Printed Circuit Board Production.

Due to the nature of the circuits designed for this project, (large numbers of high speed signals and a high devices count with high density packaging to reduce board area), printed circuit boards have been used in development rather than conventional prototype boards consisting of either wire–wrap or other point to point wiring systems.

Three pcb packages were used during this project, SMARTWORK, RGARPH and Mentor Graphics BOARD STATION. The latter two were purchased in the course of this project and considerable time was spent on familiarization of the packages. Since the packages themselves were new each of them had its own problems.

A.1 PCB Production. The discussion.

The initial development work in creating a 68000 development system from an Atari 1040 ST, used printed circuit for the expansion board. This board was laid out using a basic routing package called **SMARTWORK**. To increase track density on the board, the layout produced by SMARTWORK had half of its final busses routed by hand on the 2x artwork. Large ground plane areas were also added to the board along with ground screens between the signals to reduce inter–signal cross–talk. The final board produced fitted inside the Atari casing with slight modification to some of the Atari's internal RF shielding.

Next a prototype Controller/Development system containing a 10 MHz 68000 processor, a T800 32–bit floating point transputer and an A100 cascadable digital signal processor capable of an equivalent 380 MOPS, (million operations per second). Each processor site had support logic and a full expansion bus brought out to 96–way indirect edge connectors. The T800 could talk directly to the A100 and to the 68000 via a high speed dual port RAM. This was entered into an IBM PC/AT pcb package called **RGRAPH** by Aptos.

RGRAPH enabled schematic entry of the circuit. Since the pan and redraw facilities of the RGRAPH system were very slow it made the system unsuitable for the general use it was intended.

The special components such as the transputers, connectors and the 68000 series of chips were entered into RGRAPH schematic level and pcb level libraries. The schematic was then passed through the next stage of the process. This was **RCAP** which broke down the schematic into a form which could be handled by the **AUTOTOOLS**. RCAP also gave information such as a bill of materials, netlists or wire lists. Next came another session with RGRAPH, this time the results from RCAP were used to manually place components on the pcb. After all the components were placed the '**rats nest**' was called up. The rats nest is direct pin to pin connections representing the desired tracks. You now have the ability to route the tracks by hand or pass the job over to AUTOTOOLS which should route the tracks automatically.

During the pcb level of RGRAPH the actual first signs of problems really occurred. The first slight problem was seen at the schematic entry level. This was when the database limit was attained. However, the RCAP package had the capability of accepting more than one schematic representing one pcb, so this was not considered as a real problem. Now the first real problem was that during movement around the pcb, in RGRAPH, parts actually disappeared. Even when all the components were visible, during screen scales of less than one i.e. zoomed out, the disappearing act was still there. When a plot was done with all components visible the plot usually had bits missing.

Another problem found was at the RCAP stage of the process. When the bill of materials was requested the program crashed leaving an incomplete list of materials. The list should contain the total quantities of components used and their references, but instead after the crash there was a complete list of components, without the total quantities filled in, with the complete list of references.

Overlooking these problems, the pcb level drawing from RGRAPH was passed to the AUTOTOOLS. The program processed the information for about 3 hours, typically, before crashing leaving a unknown error code on the screen. A list of these faults along with the error code generated were telexed to the suppliers. The message that came back from America, where the software originated, was that they had never heard of these problems nor that particular error code. The suppliers assured us that these problems would all vanish with the new versions of RGRAPH and AUTOTOOLS.

With the new version of the software the autorouter stopped with the same error code. The bill of materials also gave the same result. The drawing was reprocessed completely by RCAP and this time the bill of materials came out correctly but the router gave the same message.

A piece of software was written which could cut the rats nests, group together components laid on different layers and then compiled a new rats nest file to cover this group. It could handle four such groups with the components placed on layers one to four. Five new rat nest files were produced, one for each group and a final one which linked up all the groups.

The idea behind this was that the board information would be reduced by reducing the rat information. Once one section of the board was routed another section of rat information would be added and the process repeated. This would finally leave one rat file with the final linkages for each pcb section. This would be added, the board routed and then the job would be done.

This, however, did not work. The autorouter still did not like the information. The next step was to reduce the whole database. The rat cutter was called and a logical section of the pcb was isolated. The other components not associated with this group were then deleted and the board outline reduced. The first group chosen was the area containing the A100 digital signal processor. This was the smallest logical unit and if anything should be able to pass through the autorouter then the smallest group should. Routing the board as a four signal layer board and two power planes, this small section was completed with a little manual routing.

The schematic was entered into the autorouter for the production of a two layer pcb. The router actually worked on this board processing it to some 80%. A full board once routed on the autorouter could not be resubmitted to the autorouter. This was because we were requesting small tracks and small spacing, so the tracks did not fit on the placement grid of the components so next time round the autorouter threw them out for not being on the grid. All pins of the components also had to be on this grid, so the connectors being used in the design had to be changed so that the pins were on the grid. If the routing was just in one area of the pcb and another area was to be routed, then these tracks could be placed on non–active layers thus the autorouter would not see them as components or tracks and so they need not be on the grid. With a full pcb, or continued routing in the same pcb area, then a change of layer could not be used as the tracks already routed will not be recognized by the router and hence new tracks could go over those already there.

It is at this stage where manual routing was called for. With some 100 or so connections still to be made large areas around the transputer were manually ripped up and the tracks hand routed. A similar process took place around the A100 site. Using this method the board was finally routed to 100% which the autorouter was unable to accomplish. Heavy manual routing enabled slight board layout modifications to enable more tracks to be routed through, (manually).

After the completion of the digital signal processing board work started on the Processing Array. This was drawn up using RGRAPH, but rather than using the autorouter for this board, having seen the problems encountered with the system, '**SMARTWORK**' was used instead.

With SMARTWORK the board is laid out by placing component pads on a worksheet, then requesting links between pairs of pads. After each request the router then places a track, covering the minimum board area, between the two pads. The router is unable to change sides of the board. If a side change is needed then the operator needs to place a pad down where the change is to take place. Next a request is made for these two pads to be joined. A side change is now done and a request for this pad and the final pad to be connected is made.

There is however a hidden problem with SMARTWORK. This is that the minimum board area covering for a track is not always the best solution. This sometimes places the track next to pads which need to be joined at a later date. Once tracks are placed right up against a pad it becomes difficult to use that pad as it can soon get surrounded by tracks running very close to it. To overcome this problem the operator must drop '**Fat Cells**' on the board and route to these cells taking the track closer to its destination a bit at a time. SMARTWORK is more difficult to use in that respect, but the operator soon gets used to laying out printed circuit boards and picks up the tricks needed to operate the package efficiently. One feature with SMARTWORK is that the user can control the connections on the board in an individual manner. Other packages for pcb production do not allow you to change the design at board level, thus if you have a single in–line resistor package counting eight resistors and your original schematic contained a signal **X** ending at **resistor 1** and a signal **Y** ending at **resistor 4**, then if during layout you find that **X** is in fact closer to **resistor 4** on the board and **Y** closer to **resistor 1**, then on some higher levels of router you cannot swap them over to ease the routing as it is smart and knows that this now does not match the schematic even though the result is the same. Yet again, this knowledge of the schematic can be quite useful because in SMARTWORK you could connect signal **X** to **Y** causing an error in the circuit and SMARTWORK will not complain, whereas the higher level routers know that this is an error and will either tell the operator this or just disallow the connection altogether with a warning.

Work now restarted on the large routing problem. The large transputer section of the board was cut out using the rat cutter and then passed to the autorouter. Over 7 days of 24 hour autorouting passed when the autorouter finally finished. The result was 74% complete but the database was too big for the package to compile into one database drawing so it crashed without giving the results it had just computed. Looking at the files it left, there was over 174K of pad and via information alone. This far exceeded the 64K database limit. The tracks were well into the megabyte range.

A new Mentor Graphics autorouting package, **BOARD STATION**, was installed on the Apollo system. The Apollo system contained a network of

7 workstations, 4 black and white and 3 colour workstations. The colour nodes are 68020 based units with up to 4 Mbyte of RAM and their own hard discs. The autorouter is only able to run on these high capacity nodes, so there were only three workstations available for autorouting. The work on the boards now switched to the Apollo with the resultant loss of time.

The development system was redesigned to be in separate units. The transputer stage of this was redesigned so that the link between it and the 68000 system was via a single in-line connector where on the 68000 card a dual ported RAM would sit with a matching connector. The A100 signal processing stage was dropped completely from the plans. The transputer section was then entered into the Mentor package. A compact solution using a six layer board was routed to 100%. The Mentor package was considerably faster at routing boards though there were more processes needed to create a schematic, produce a component layout and start autorouting the tracks. Some of these processes were the total redefinition of all the unusual parts to be used as with RGRAPH.

The Mentor package had autoplacement of components as well as autorouting. The autoplacement however, was found unsuitable for this design and yielded a poor component layout. The placement was modified by hand to yield a more suitable arrangement prior to routing.

Due to cost concerns this development card had to be reduced. A new design emerged which dropped large amounts of logic and the dual port RAM interface. This card became quite large but was a two layer board.

The analogue unit was designed on this system. This board was plotted out. It was at this stage that problems with the Mentor software were encountered. The plots were produced on a large Hewlet Packard plotter. The plots were two times artworks but the tracks and pads were only drawn as outlines and not filled in as required. Mentor was contacted and they said they were working on the problem. This problem was not quickly solved so the analogue board, the most important board of the entire system, was plotted out and the two times artwork finished off by hand. All tracks and pads were filled in and ground planes created. This board was then produced by Tate Circuit Industries in Birmingham.

The main pcb was modified slightly and the layout changed. The board was routed and finally ended with a 100% routed solution. Looking closely at the tracks around the indirect edge connectors it was noticed that the autorouter had ignored the keepout areas around the mounting holes for the connectors, even though one of the connector definitions was supplied by Mentor themselves. Hand routing on the Mentor package now took place to rectify this problem. The hand routing with the Mentor package was not as straightforward as expected although it has been claimed that the hand routing feature of the BOARD STATION package is one of its best features.

With the initial hand routing difficulty over, the board was now complete with thin tracks. Time was now needed to enlarge the power lines or re–routed the entire board starting with power tracks of 62 mils thick. The power lines need to be of this order using 1 oz. copper clad board, due to the possible maximum current capacity of about 7 amps. This level could be attained with all the fast TTL buffers, the active RAMs and the transputer all requiring their maximum current ratings. In practice however, the 62 mils would only be required for the initial main stream of the power tree. Reduced track widths could be used for further branches of the power tree which carry less current as long as the normal "**track width vs. current rating**" rules are obeyed, [37].

Photoplotting pcb artwork yields a high quality mask for pcb production and at that time commercial photoplotting was quoted at 60p per square inch. For a double–sided pcb three plots are needed to produce the board. Two plots are needed for the tracks, one for each side, and one for a padmaster for the production of the solder–resist. The padmaster could also be used to obtain the drilling information. With the Controller board being 18x12 sq. inches and 3 plots needed, the total plot area came to 648 sq. inches and about 380 pounds. An offer of producing these plots as a free demonstration was accepted.

The resultant photoplots were sent to Tate Circuit Industries for the circuit board production. The board that came back had no visible breaks in the tracking and this route had saved not only money but also time, for if the board was plotted on the Hewlet Packard plotter then the two times artworks would need filling by hand. A penalty in cost would also be incurred when the artwork reached Tate as they would need to photo–reduce large drawings to produce the masks for the pcb.

The department has since purchased its own small photoplotter based on a flat–bed pen plotter. This machine would still be too small to plot either the main pcb or the analogue board as the largest film the plotter can take yields a board size of 14x10 inches, and both these boards have 18 inches in one dimension by 12 and 10 inches in the other, respectively.

Since the production of the first two layer prototype Controller, the board has been rebuilt as a 6 layer pcb. All the routing was done using the Mentor Package and the photoplots produced in house. The actual cost of a six layer printed circuit board measuring 11 inches by 7 inches and containing 2000 holes, was 420 pounds from Crowthorn Circuits.

B. Format of Results Disc.

Results Header:

Locations	Contents
0	Number of result blocks taken.
1	Number of waveforms captured.
2	Number of injections found.
3	Number of result blocks between waveform captures.
4–5	Primary air setting as REAL64.
5–7	Secondary air setting as REAL64.
8–71	System offsets as an 8x8 INT array.
72–199	System gains as an 8x8 REAL64 array.

Result Block Format:

Locations	Contents
0	Time into full system run.
1	Balance reading and N° of accepted values as INT16s.
2	Packed requests, each request as one BYTE.
3	Number of samples processed by the Processing Array.
4	Partial injection current integral.
5	Packed maximum values found, each as one BYTE.
6	Packed minimum values found, each as one BYTE.
7–10	Signal integrals.
11–14	Time of peaks from start of block.
15–18	Elapsed processor times.

Note:–

5–18 are only valid if 3 is > 0.

Captured Waveform Format:

Locations	Contents
0–999	Injection current, 4000 BYTE values in total.
1000–1999	Current from sensor 1, 4000 BYTE values in total.
2000–2999	Current from sensor 2, 4000 BYTE values in total.

Note:–

To obtain the information about the requested waveforms, extract from the previous result block.

C. Contents of Software Volumes.

D. Simulation of a Discrete Quantization Unit.

What is presented here is a simulation of the discrete adaptive quantizer combining both the original circuitry and using the gain stages used within the actual Data Acquisition Unit. The simulation shows that the circuit should be able to operate with a cycle time of 260 ns. As with the diagrams of the previous discrete quantization elements, sections 6.1.1 and 6.1.2, the diagram has been reduced to only show the logic elements and not the A/D and analogue elements.

Figure 64 shows the circuit used for the simulation.

The signal selector has been expanded to cater for up to 16 signals. The RAM has been wired to enable 16 levels of gain, but the gain calculation stage is still wired to compute using 8 levels. Another section called the '**Gain Preset Unit**' has been added to preset the gains to unity after a reset.

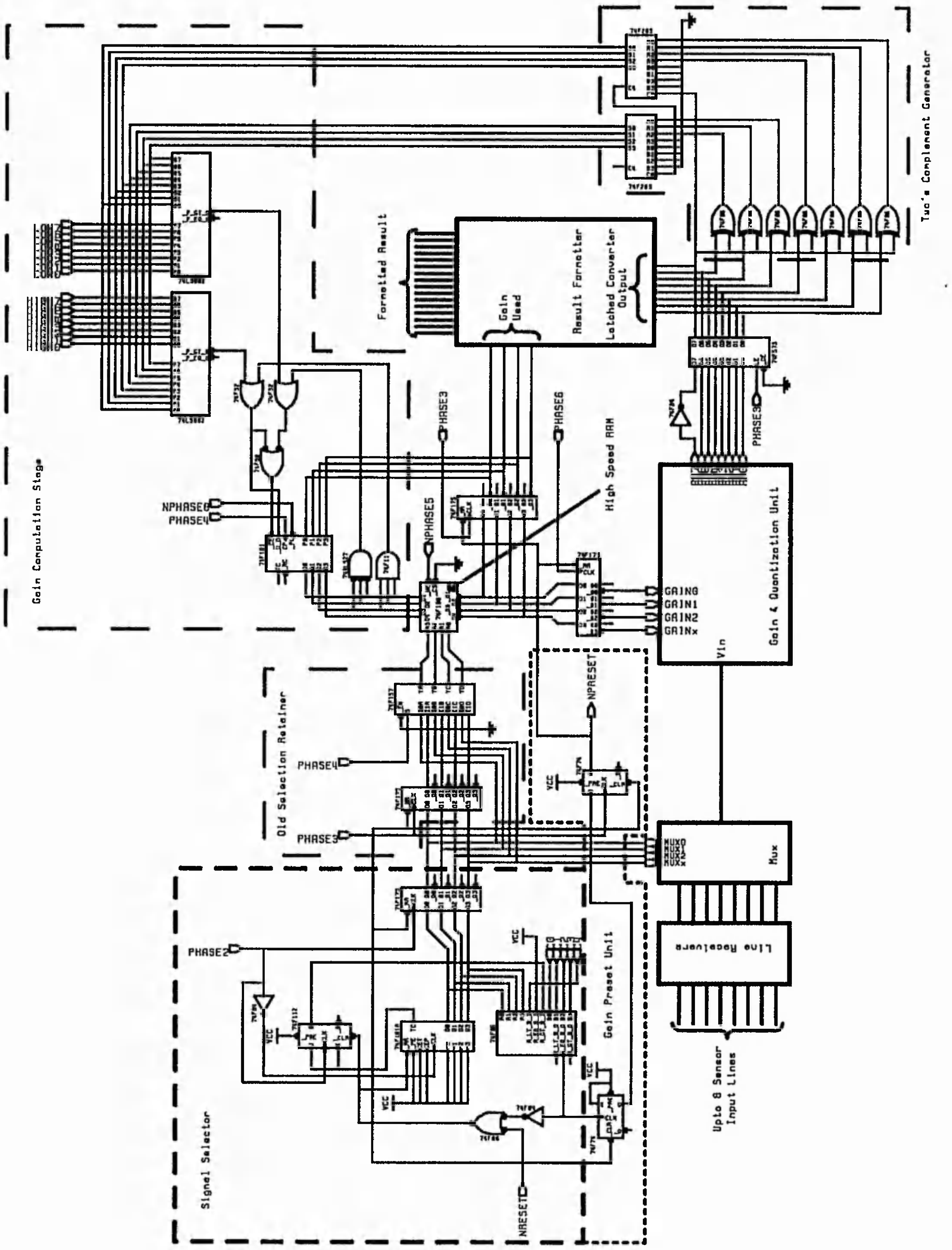

Figure 64 Circuit of the Simulated Discrete Adaptive Quantizer.

Simulation of a Discrete Quantization Unit.

The following list contains the information needed to run the simulator:

```
# Set-up QUICKSIM for simulation of adaptive quantizer.

    RADix Hex
    ASSIgn hi_$list_radix Hex
    ASSIgn hi_$monitor_radix Hex

# Define buses.

    DEFine Bus C C4 C3 C2 C1 C0 -Combine
    DEFine Bus HIGH HIGH7 HIGH6 HIGH5 HIGH4 HIGH3
               HIGH2 HIGH1 HIGH0 -Combine
    DEFine Bus LOW LOW7 LOW6 LOW5 LOW4 LOW3 LOW2
               LOW1 LOW0 -Combine
    DEFine Bus AD AD7 AD6 AD5 AD4 AD3 AD2 AD1 AD0
               -Combine
    DEFine Bus MUX MUXx MUX2 MUX1 MUX0 -Combine
    DEFine Bus GAIN GAINx GAIN2 GAIN1 GAIN0 -Combine

# Set-up desired output format.

# PHASEs.

    TRAce PHASE2
    MONitor Binary PHASE2
    TRAce PHASE3
    MONitor Binary PHASE3
    TRAce PHASE4
    MONitor Binary PHASE4
    TRAce NPHASE5
    MONitor Binary NPHASE5
    TRAce PHASE6
    MONitor Binary PHASE6
    TRAce NPHASE8
    MONitor Binary NPHASE8
    TRAce NRESET

# RESET.

    LISt -Change Binary NRESET
    MONitor Binary NRESET
    TRAce NPRESET
    LISt -Change Binary NPRESET
    MONitor Binary NPRESET
```

```
# Number of signals requested.

    LISt Hex C
    MONitor Hex C

# Multiplexer output.

    TRAce MUXx
    TRAce MUX2
    TRAce MUX1
    TRAce MUX0
    LISt -Change Hex MUX
    MONitor Hex MUX

# Gain used.

    TRAce GAINx
    TRAce GAIN2
    TRAce GAIN1
    TRAce GAIN0
    LISt -Change Hex GAIN
    MONitor Hex GAIN

# A/D converter.

    LISt Hex AD
    MONitor Hex AD

# High value adaptation level.

    LISt Hex HIGH
    MONitor Hex HIGH

# Low value adaptation level.

    LISt Hex LOW
    MONitor Hex LOW

    SCAle USer Time 1
    SCAle TRace Time 10
    INItialize XR
    VIEw Sheet SHEET1 /
    PERiod List 0
    PERiod Trace 0
```

```
# Set-up initial FORCEs.

# Set HIGHn for 230.

   FORCe HIGH E6

# Set LOWn for 26.

   FORCe LOW 1A

# Set Cn for 11 (10 signals.)

   FORCe C B

# Pulse RESET

   FORCe NRESET 0S 0 ,,
   FORCe NRESET 1S 300 ,,

# Set PHASEs to inactive.

   FORCe PHASE2 0S 0 ,,
   FORCe PHASE3 0S 0 ,,
   FORCe PHASE4 0S 0 ,,
   FORCe NPHASE5 1S 0 ,,
   FORCe PHASE6 0S 0 ,,
   FORCe NPHASE8 1S 0 ,,

# Set ADn for 128 (0 volt input signal).

   FORCe AD 80

# Set-up for main clocks.

# A/D process.

   CLOck Period 260
   FORCe PHASE2 1S 1000 -Repeat
   FORCe PHASE2 0S 1150 -Repeat
   CLOck Period 260
   FORCe PHASE6 1S 1060 -Repeat
   FORCe PHASE6 0S 1210 -Repeat
   CLOck Period 260
   FORCe PHASE3 1S 1220 -Repeat
   FORCe PHASE3 0S 1260 -Repeat
```

RAM write sequence.

```
    CLOck Period 260
    FORCe NPHASE8 0S 1020 -Repeat
    FORCe NPHASE8 1S 1060 -Repeat
    CLOck Period 260
    FORCe PHASE4 1S 1080 -Repeat
    FORCe PHASE4 0S 1160 -Repeat
    CLOck Period 260
    FORCe NPHASE5 0S 1100 -Repeat
    FORCe NPHASE5 1S 1140 -Repeat
```

What follows is the simulation traces for the first 10,000 ns of operation of the circuit whilst being fed with inputs approximately at zero volts.

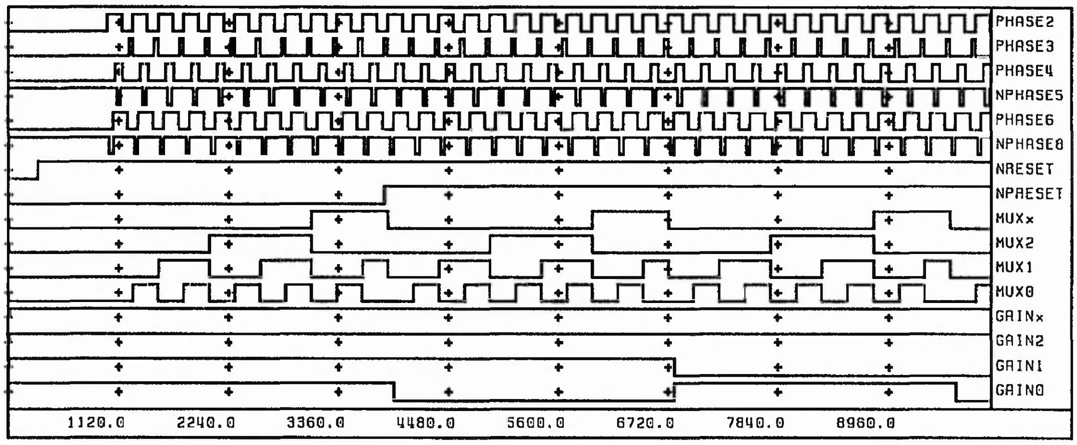

Figure 65 The First 10,000 ns of Operation.

The following list is the changing outputs of the circuit for the duration of the first 30,000 ns of circuit simulation. The outputs show how the gain is increased from unity, (F), to 128, (8). The most significant bit of the gain is ignored as the circuit is intended to be used as the other discrete versions, (only providing gains from 1 to 128 in binary weighted steps).

```
   0.0  0  Xr 0B  Xr X  80  E6  1A
   7.0  0  0  0B  Xr X  80  E6  1A
   9.0  0  0  0B  0  X  80  E6  1A
  13.5  0  0  0B  0  F  80  E6  1A
 300.0  1  0  0B  0  F  80  E6  1A
1266.5  1  0  0B  1  F  80  E6  1A
```

```
1525.0  1  0  0B  0  F  80  E6  1A
1526.5  1  0  0B  2  F  80  E6  1A
1786.5  1  0  0B  3  F  80  E6  1A
2045.0  1  0  0B  0  F  80  E6  1A
2046.5  1  0  0B  4  F  80  E6  1A
2306.5  1  0  0B  5  F  80  E6  1A
2565.0  1  0  0B  4  F  80  E6  1A
2566.5  1  0  0B  6  F  80  E6  1A
2826.5  1  0  0B  7  F  80  E6  1A
3085.0  1  0  0B  0  F  80  E6  1A
3086.5  1  0  0B  8  F  80  E6  1A
3346.5  1  0  0B  9  F  80  E6  1A
3605.0  1  0  0B  8  F  80  E6  1A
3606.5  1  0  0B  A  F  80  E6  1A
3825.3  1  1  0B  A  F  80  E6  1A
3865.0  1  1  0B  0  F  80  E6  1A
3925.0  1  1  0B  0  E  80  E6  1A
4126.5  1  1  0B  1  E  80  E6  1A
4385.0  1  1  0B  0  E  80  E6  1A
4386.5  1  1  0B  2  E  80  E6  1A
4646.5  1  1  0B  3  E  80  E6  1A
4905.0  1  1  0B  0  E  80  E6  1A
4906.5  1  1  0B  4  E  80  E6  1A
5166.5  1  1  0B  5  E  80  E6  1A
5425.0  1  1  0B  4  E  80  E6  1A
5426.5  1  1  0B  6  E  80  E6  1A
5686.5  1  1  0B  7  E  80  E6  1A
5945.0  1  1  0B  0  E  80  E6  1A
5946.5  1  1  0B  8  E  80  E6  1A
6206.5  1  1  0B  9  E  80  E6  1A
6465.0  1  1  0B  8  E  80  E6  1A
6466.5  1  1  0B  A  E  80  E6  1A
6725.0  1  1  0B  0  E  80  E6  1A
6785.0  1  1  0B  0  C  80  E6  1A
6786.5  1  1  0B  0  D  80  E6  1A
6986.5  1  1  0B  1  D  80  E6  1A
7245.0  1  1  0B  0  D  80  E6  1A
7246.5  1  1  0B  2  D  80  E6  1A
7506.5  1  1  0B  3  D  80  E6  1A
7765.0  1  1  0B  0  D  80  E6  1A
7766.5  1  1  0B  4  D  80  E6  1A
8026.5  1  1  0B  5  D  80  E6  1A
8285.0  1  1  0B  4  D  80  E6  1A
8286.5  1  1  0B  6  D  80  E6  1A
8546.5  1  1  0B  7  D  80  E6  1A
8805.0  1  1  0B  0  D  80  E6  1A
8806.5  1  1  0B  8  D  80  E6  1A
```

```
 9066.5  1  1  0B  9  D  80  E6  1A
 9325.0  1  1  0B  8  D  80  E6  1A
 9326.5  1  1  0B  A  D  80  E6  1A
 9585.0  1  1  0B  0  D  80  E6  1A
 9645.0  1  1  0B  0  C  80  E6  1A
 9846.5  1  1  0B  1  C  80  E6  1A
10105.0  1  1  0B  0  C  80  E6  1A
10106.5  1  1  0B  2  C  80  E6  1A
10366.5  1  1  0B  3  C  80  E6  1A
10625.0  1  1  0B  0  C  80  E6  1A
10626.5  1  1  0B  4  C  80  E6  1A
10886.5  1  1  0B  5  C  80  E6  1A
11145.0  1  1  0B  4  C  80  E6  1A
11146.5  1  1  0B  6  C  80  E6  1A
11406.5  1  1  0B  7  C  80  E6  1A
11665.0  1  1  0B  0  C  80  E6  1A
11666.5  1  1  0B  8  C  80  E6  1A
11926.5  1  1  0B  9  C  80  E6  1A
12185.0  1  1  0B  8  C  80  E6  1A
12186.5  1  1  0B  A  C  80  E6  1A
12445.0  1  1  0B  0  C  80  E6  1A
12505.0  1  1  0B  0  8  80  E6  1A
12506.5  1  1  0B  0  B  80  E6  1A
12706.5  1  1  0B  1  B  80  E6  1A
12965.0  1  1  0B  0  B  80  E6  1A
12966.5  1  1  0B  2  B  80  E6  1A
13226.5  1  1  0B  3  B  80  E6  1A
13485.0  1  1  0B  0  B  80  E6  1A
13486.5  1  1  0B  4  B  80  E6  1A
13746.5  1  1  0B  5  B  80  E6  1A
14005.0  1  1  0B  4  B  80  E6  1A
14006.5  1  1  0B  6  B  80  E6  1A
14266.5  1  1  0B  7  B  80  E6  1A
14525.0  1  1  0B  0  B  80  E6  1A
14526.5  1  1  0B  8  B  80  E6  1A
14786.5  1  1  0B  9  B  80  E6  1A
15045.0  1  1  0B  8  B  80  E6  1A
15046.5  1  1  0B  A  B  80  E6  1A
15305.0  1  1  0B  0  B  80  E6  1A
15365.0  1  1  0B  0  A  80  E6  1A
15566.5  1  1  0B  1  A  80  E6  1A
15825.0  1  1  0B  0  A  80  E6  1A
15826.5  1  1  0B  2  A  80  E6  1A
16086.5  1  1  0B  3  A  80  E6  1A
16345.0  1  1  0B  0  A  80  E6  1A
16346.5  1  1  0B  4  A  80  E6  1A
16606.5  1  1  0B  5  A  80  E6  1A
```

```
16865.0  1  1  0B  4  A  80  E6  1A
16866.5  1  1  0B  6  A  80  E6  1A
17126.5  1  1  0B  7  A  80  E6  1A
17385.0  1  1  0B  0  A  80  E6  1A
17386.5  1  1  0B  8  A  80  E6  1A
17646.5  1  1  0B  9  A  80  E6  1A
17905.0  1  1  0B  8  A  80  E6  1A
17906.5  1  1  0B  A  A  80  E6  1A
18165.0  1  1  0B  0  A  80  E6  1A
18225.0  1  1  0B  0  8  80  E6  1A
18226.5  1  1  0B  0  9  80  E6  1A
18426.5  1  1  0B  1  9  80  E6  1A
18685.0  1  1  0B  0  9  80  E6  1A
18686.5  1  1  0B  2  9  80  E6  1A
18946.5  1  1  0B  3  9  80  E6  1A
19205.0  1  1  0B  0  9  80  E6  1A
19206.5  1  1  0B  4  9  80  E6  1A
19466.5  1  1  0B  5  9  80  E6  1A
19725.0  1  1  0B  4  9  80  E6  1A
19726.5  1  1  0B  6  9  80  E6  1A
19986.5  1  1  0B  7  9  80  E6  1A
20245.0  1  1  0B  0  9  80  E6  1A
20246.5  1  1  0B  8  9  80  E6  1A
20506.5  1  1  0B  9  9  80  E6  1A
20765.0  1  1  0B  8  9  80  E6  1A
20766.5  1  1  0B  A  9  80  E6  1A
21025.0  1  1  0B  0  9  80  E6  1A
21085.0  1  1  0B  0  8  80  E6  1A
21286.5  1  1  0B  1  8  80  E6  1A
21545.0  1  1  0B  0  8  80  E6  1A
21546.5  1  1  0B  2  8  80  E6  1A
21806.5  1  1  0B  3  8  80  E6  1A
22065.0  1  1  0B  0  8  80  E6  1A
22066.5  1  1  0B  4  8  80  E6  1A
22326.5  1  1  0B  5  8  80  E6  1A
22585.0  1  1  0B  4  8  80  E6  1A
22586.5  1  1  0B  6  8  80  E6  1A
22846.5  1  1  0B  7  8  80  E6  1A
23105.0  1  1  0B  0  8  80  E6  1A
23106.5  1  1  0B  8  8  80  E6  1A
23366.5  1  1  0B  9  8  80  E6  1A
23625.0  1  1  0B  8  8  80  E6  1A
23626.5  1  1  0B  A  8  80  E6  1A
23885.0  1  1  0B  0  8  80  E6  1A
24146.5  1  1  0B  1  8  80  E6  1A
24405.0  1  1  0B  0  8  80  E6  1A
24406.5  1  1  0B  2  8  80  E6  1A
```

```
24666.5  1  1  0B  3  8  80  E6  1A
24925.0  1  1  0B  0  8  80  E6  1A
24926.5  1  1  0B  4  8  80  E6  1A
25186.5  1  1  0B  5  8  80  E6  1A
25445.0  1  1  0B  4  8  80  E6  1A
25446.5  1  1  0B  6  8  80  E6  1A
25706.5  1  1  0B  7  8  80  E6  1A
25965.0  1  1  0B  0  8  80  E6  1A
25966.5  1  1  0B  8  8  80  E6  1A
26226.5  1  1  0B  9  8  80  E6  1A
26485.0  1  1  0B  8  8  80  E6  1A
26486.5  1  1  0B  A  8  80  E6  1A
26745.0  1  1  0B  0  8  80  E6  1A
27006.5  1  1  0B  1  8  80  E6  1A
27265.0  1  1  0B  0  8  80  E6  1A
27266.5  1  1  0B  2  8  80  E6  1A
27526.5  1  1  0B  3  8  80  E6  1A
27785.0  1  1  0B  0  8  80  E6  1A
27786.5  1  1  0B  4  8  80  E6  1A
28046.5  1  1  0B  5  8  80  E6  1A
28305.0  1  1  0B  4  8  80  E6  1A
28306.5  1  1  0B  6  8  80  E6  1A
28566.5  1  1  0B  7  8  80  E6  1A
28825.0  1  1  0B  0  8  80  E6  1A
28826.5  1  1  0B  8  8  80  E6  1A
29086.5  1  1  0B  9  8  80  E6  1A
29345.0  1  1  0B  8  8  80  E6  1A
29346.5  1  1  0B  A  8  80  E6  1A
29605.0  1  1  0B  0  8  80  E6  1A
29866.5  1  1  0B  1  8  80  E6  1A

  TIME    ^NRESET    ^MUX        ^HIGH
                ^C       ^GAIN        ^LOW
             ^NPRESET      ^AD
```

Looking at the above output, what could be described as errors in the output can be seen. These are the non-continuous counts for the **GAIN** and the **MUX** outputs. It should be noted, however, that the invalid count lasts only 1.5 ns. This '**error**' is in fact due to the now symmetrical propagation delay times of the latch for transitions from 0 to 1. What happens in this case is that the outputs, when changing from 7 to 8, pass through a zero condition. The propagation from 1 to 0 is less than the propagation of the 0 to 1, hence 7 goes momentarily to 0 whilst changing to 8. An example of this can clearly be seen highlighted at **TIME = 28566.5** near the very end of the list.

E. Debugging Examples.

In the following section there will be several examples of software debugging. This starts with the simple debugging using the Atari as a Transputer Development Workstation, mentioned in section 5.3. Next follows a debug example using the IBM based Transputer Development System. This method allows detailed analysis of all the processors in the system, providing the operator has experience debugging transputers and detailed knowledge of what the tasks should have been doing.

Below is a typical example of a peek session on a reset transputer using the Atari based TDW written specially for the task.

The actual program, written in Occam 1, was:

```
CHAN Output AT 3:
DEF Text = "This is some text from the transputer.":
SEQ
  SEQ i = 1 FOR Text[0]
    Output ! Text[i]
  SEQ i = 0 FOR 20
    Output ! i
```

When **DEF Text = "This is some text from the transputer.":** is compiled, a constant of 26, ($38_{DECIMAL}$), representing the size of the string, is placed in Text[0]. This is then followed by the ASCII representation for each character from Text[1] to Text[Text[0]], (Text[38]).

The next page shows the type of data returned from the transputer with the transputer instructions by the side of it. The following that is an example of disassembled code.

Example of TDW Peek:

ADDRESS	FOUND	CODE	ASSEMBLY
80000048	**53** 00 80 **C1**	53 00	?
8000004C	10 26 **54 68**	80	?
80000050	**69 73 20 69**	C1	EQC
80000054	**73 20 73 6F**	10	?
80000058	**6D 65 20 74**	26	LEN
8000005C	**65 78 74 20**	54.21	[TEXT]
80000060	**66 72 6F 6D**	60 BD	AJW -3
80000064	**20 74 68 65**	41	LDC 1
80000068	**20 74 72 61**	D1	STL 1
8000006C	**6E 73 70 75**	22 46	LDC 26
80000070	**74 65 72 21**	D2	STL 2
80000074	60 BD **41** D1	24 F2	MINT
80000078	**22 46** D2 **24**	53	LDNLP 3
8000007C	**F2** 53 **71** 63	71	LDL 1
80000080	4A **21 FB** F2	63 4A	LDC -54$_{DECIMAL}$
80000084	**F1** FF **11** 21	21 FB	LDPI
80000088	40 **22 F1** 40	F2	BSUB
8000008C	**D1** 21 44 **D2**	F1	LB
80000090	24 F2 **53** 71	FF	OUTWORD
80000094	**FF** 11 **49** 22	11	LDLP 1
80000098	F1 **B3** 22 F0	21 40	LDC 16$_{DECIMAL}$
		22 F1	LEND
		40	LDC 0
		D1	STL 1
		21 44	LDC 20$_{DECIMAL}$
		D2	STL 2
		24 F2	MINT
		53	LDNLP 3
		71	LDL 1
		FF	OUTWORD
		11	LDLP 1
		49	LDC 9
		22 F1	LEND
		B3	AJW 3
		22 F0	RET

Note:–

1. The data has been tabulated to clarify the example.
2. Bold characters in the **FOUND** column represent single commands.

SEQ i = 1 FOR Text[0]

LDC 1	Load a constant of 1.
STL 1	Store it in the first local, (i).
LDC 26	Load a constant of $38_{DECIMAL}$.
STL 2	Store loop count in the second local.

Output ! Text[i]

MINT	Stack the minimum integer.
LDNLP 3	Form the address of the required channel.
LDL 1	Get the value of i.
LDC $-54_{DECIMAL}$	Load offset to start of text.
LDPI	Create a pointer for the next instruction.
BSUB	Create a byte subscript.
LB	Load byte.
OUTWORD	Output it.

Perform loop.

LDLP 1	Point to i.
LDC 16	Stack 16 for loop back pointer.
LEND	Loop back depending upon i and increase i.

SEQ i = 0 FOR 20

LDC 0	Load a constant of 0.
STL 1	Store it in the first local, (i).
LDC $20_{DECIMAL}$	Load a constant of $20_{DECIMAL}$.
STL 2	Store loop count in the second local.

Output ! i

MINT	Stack the minimum integer.
LDNLP 3	Form the address of the required channel.
LDL 1	Get the value of i.
OUTWORD	Output it.

Perform loop.

LDLP 1	Point to i.
LDC 9	Stack 9 for loop back offset.
LEND	Loop back depending upon i and increase i.

F. EPLD Data.

The EPLDs were created using an IBM PC running Altera's erasable programmable logic device software, **Aplus**. This section contains the input data files and the pin diagrams of the EPLDs used. The diagrams were extracted from the Altera report files produced by the software.

Each section containing an EPLD for a new task is started on a new page for easy access to the data. The presentation format is the input data file, **.ADF**, followed by a cut–down version of the report file, **.RPT**. The report file normally contains data relating to the utilization of macrocells for the input and output pads and buried registers within the device, (this had been removed to produce a more condensed form).

F.A Atari Control EPLDs.

The Atari control EPLD has been modified when changing fabrication implementation. The first version presented here, was used in the original printed circuit board which sat underneath the Atari's keyboard inside the actual casing of the computer. When changing from the pcb version to the wire–wrapped version the EPLD was modified again to contain some extra features. This EPLD is identical to that used in the EA2 board except that one of the buffer control output pins needed inversion within the EPLD to provide easier buffer layout on the pcb. This modified version is contained within the descriptions within the main body to the thesis, (section 5.1, page 48).

F.A.A Original Atari Control EPLD.

Input file (.ADF):

```
E.Mills.
Research
November 12, 1986

2
EP1210
Modification due to new board.
NETMAP Version 3.0, Baseline 15, 8/3/1985
PART: EP1210
INPUTS: NBR1/38, RW/35, NBGACK/36, CLK, NBR2/37, ERRMASK,
        NBERR1, NRESET/19, NB2/5, NA2/2, NA1/3, NA0/4,
        NB1/6, NB0/7, FC2/23, FC1/22, FC0/21, NAS/24,
        BM1EN, BM2EN
OUTPUTS: NBR, DIR2, BM1, BM2, NBM1, NBM2, NBERR, NILP0/10,
         NIPL1/9, NILP2/8, NENA, NENB, STATUS, BG1D, BG2D,
         BREN
NETWORK:
NBR = CONF (NBR,VCC)
DIR2 = CONF (DIR2,VCC)
BM1 = CONF (BM1,VCC)
BM2 = CONF (BM2,VCC)
NBM1 = CONF (NBM1,VCC)
NBM2 = CONF (NBM2,VCC)
NBERR = CONF (NBERR,VCC)
NILP0 = CONF (NIPL0,VCC)
NIPL1 = CONF (NIPL1,VCC)
```

```
NILP2 = CONF (NIPL2,VCC)
NENA = CONF (NENA,VCC)
NENB = CONF (NENB,VCC)
STATUS = CONF (AGTB,VCC)
BG1D = CONF (BG1D,VCC)
BG2D = CONF (BG2D,VCC)
BREN = CONF (BREN,VCC)
NBR = OR (NBREN,BGACK)
DIR2 = XOR (RW,BM2)
BM1 = AND (BM1EN,BGACK)
BM2 = AND (BM2EN,BGACK)
NBM1 = NOT (BM1)
NBM2 = NOT (BM2)
NBERR = OR (NBERR1,ERRMASK)
NIPL0 = OR (SA0,SB0)
NIPL1 = OR (SA1,SB1)
NIPL2 = OR (SA2,SB2)
NENA = AND (INTACK,ANGTB,NBM2)
NENB = AND (INTACK,AGTB,NBM2)
AGTB = OR (A2GB2,L2,L3)
BG1D = AND (BR1EN,NOTCPU)
BG2D = AND (BR2EN,NOTCPU)
BREN = NOT (NBREN)
NBREN = AND (NBR1,NBR2)
BGACK = NOT (NBGACK)
RW = INP (RW)
BM1EN = INP (BM1EN)
BM2EN = INP (BM2EN)
NBERR1 = INP (NBERR1)
ERRMASK = INP (ERRMASK)
SA0 = AND (NA0,AGTB)
SB0 = AND (NB0,ANGTB)
SA1 = AND (NA1,AGTB)
SB1 = AND (NB1,ANGTB)
SA2 = AND (NA2,AGTB)
SB2 = AND (NB2,ANGTB)
INTACK = AND (FC2,FC1,FC0,AS)
ANGTB = NOT (AGTB)
A2GB2 = AND (A2,NB2)
L2 = AND (A2EB2,A1GB1)
L3 = AND (A2EB2,A1EB1,A0GB0)
BR1EN = OR (BRL1,BROK)
NOTCPU = NOR (BM1,BM2)
BR2EN = OR (BROK,BRL2)
NBR1 = INP (NBR1)
NBR2 = INP (NBR2)
NBGACK = INP (NBGACK)
```

```
NA0 = INP (NA0)
NB0 = INP (NB0)
NA1 = INP (NA1)
NB1 = INP (NB1)
NA2 = INP (NA2)
NB2 = INP (NB2)
FC2 = INP (FC2)
FC1 = INP (FC1)
FC0 = INP (FC0)
AS = NOT (NAS)
A2 = NOT (NA2)
A2EB2 = NOT (NA2EB2)
A1GB1 = AND (A1,NB1)
A1EB1 = NOT (NA1EB1)
A0GB0 = AND (A0,NB0)
BRL1 = NORF (M1,CLK,RESET,GND)
BROK = NOR (BRL1,BRL2)
BRL2 = NORF (M2,CLK,RESET,GND)
NAS = INP (NAS)
NA2EB2 = XOR (NA2,NB2)
A1 = NOT (NA1)
NA1EB1 = XOR (NA1,NA2)
A0 = NOT (NA0)
M1 = OR (SBR1,SFB1)
CLK = INP (CLK)
RESET = NOT (NRESET)
M2 = OR (SBR2,SFB2)
SBR1 = AND (NBR1,NBOTH)
SFB1 = AND (BR2EN,BOTH)
NRESET = INP (NRESET)
SBR2 = AND (NBR2,NBOTH)
SFB2 = AND (NBR2EN,BOTH)
NBOTH = NOT (BOTH)
BOTH = NOR (NBR1,NBR2)
NBR2EN = NOT (BR2EN)
END$
```

EPLD Data.

Part of report file, (.RPT):

```
ALTERA Design Processor Utilization Report

 ***** Design implemented successfully

E.Mills.
Research
November 12, 1986

2
EP1210
Modification due to new board.
NETMAP Version 3.0, Baseline 15, 8/3/1985

              EP1210
            - - - - -
       CLK -| 1   40|- Vcc
       NA2 -| 2   39|- Vcc
       NA1 -| 3   38|- NBR1
       NA0 -| 4   37|- NBR2
       NB2 -| 5   36|- NBGACK
       NB1 -| 6   35|- RW
       NB0 -| 7   34|- ERRMASK
     NILP2 -| 8   33|- NBERR1
     NIPL1 -| 9   32|- BG2D
     NILP0 -|10   31|- NENB
      NBM1 -|11   30|- BM2
      NBM2 -|12   29|- STATUS
      DIR2 -|13   28|- NBERR
       NBR -|14   27|- BM1
      BREN -|15   26|- NENA
       GND -|16   25|- BG1D
     BM2EN -|17   24|- NAS
     BM1EN -|18   23|- FC2
    NRESET -|19   22|- FC1
       GND -|20   21|- FC0
            - - - - -


**UNUSED RESOURCES**

      Name  Pin  Resource     MCell    PTerms

         -   16         -        20        12
         -   NA         -        15         8
         -   NA         -        16         8
```

```
**PART UTILIZATION**

97% Pins
89% MacroCells
28% Pterms
```

F.A.B Atari Control EPLD. EA2 version.

Input file (.ADF):

```
E.Mills.
Research
July 8, 1987

3
EP1210
Modification due to new EA2 board.
NETMAP Version 3.0, Baseline 15, 8/3/1985
PART: EP1210
INPUTS: NBR1/38, RW/35, NBGACK/36, 8MHz/1, NBR2/37,
        ERRMASK/34, NBERR1/33, NRESET/19, NB2/5, NA2/2,
        NA1/3, NA0/4, NB1/6, NB0/7, FC2/23, FC1/22, FC0/21,
        NAS/24, BM1EN/18, BM2EN/17,NMON
OUTPUTS: NBR/14, DIR2/26, NBM2/27, BM1/30, NBERR/29,
         NILP0/10, NIPL1/9, NILP2/8, NENA/11, NENB/12,
         STATUS/13, BG1D/15, BG2D/25, DIR1, FIX2
NETWORK:
NBR = CONF (NBR,VCC)
DIR2 = CONF (DIR2,VCC)
NBM2 = CONF (NBM2,VCC)
BM1 = CONF (BM1,VCC)
NBERR = CONF (NBERR,VCC)
NILP0 = CONF (NIPL0,VCC)
NIPL1 = CONF (NIPL1,VCC)
NILP2 = CONF (NIPL2,VCC)
NENA = CONF (NENA,VCC)
NENB = CONF (NENB,VCC)
STATUS,AGTB = COIF (STATUS,VCC)
BG1D = CONF (BG1D,VCC)
BG2D = CONF (BG2D,VCC)
DIR1 = CONF (DIR1,VCC)
FIX2 = CONF (FIX2,VCC)
NBR = OR (NBREN,BGACK)
DIR2 = AND (DIR2E,NMON)
BM2 = AND (BM2EN,BGACK)
```

```
NBERR = OR (NBERR1,ERRMASK)
NIPL0 = OR (SA0,SB0)
NIPL1 = OR (SA1,SB1)
NIPL2 = OR (SA2,SB2)
NENA = AND (INTACK,ANGTB,NBM2)
NENB = AND (INTACK,AGTB,NBM2)
STATUS = OR (A2GB2,L2,L3)
BG1D = AND (BR1EN,NOTCPU)
BG2D = AND (BR2EN,NOTCPU)
DIR1 = NOT (NDIR1)
FIX2 = AND (BM2,NMON)
NBREN = AND (NBR1,NBR2)
BGACK = NOT (NBGACK)
DIR2E = XOR (RW,BM2)
NMON = INP (NMON)
BM2EN = INP (BM2EN)
BM1 = AND (BM1EN,BGACK)
NBERR1 = INP (NBERR1)
ERRMASK = INP (ERRMASK)
SA0 = AND (NA0,AGTB)
SB0 = AND (NB0,ANGTB)
SA1 = AND (NA1,AGTB)
SB1 = AND (NB1,ANGTB)
SA2 = AND (NA2,AGTB)
SB2 = AND (NB2,ANGTB)
INTACK = AND (FC2,FC1,FC0,AS)
ANGTB = NOT (AGTB)
NBM2 = NOT (BM2)
A2GB2 = AND (A2,NB2)
L2 = AND (A2EB2,A1GB1)
L3 = AND (A2EB2,A1EB1,A0GB0)
BR1EN = NOR (BRL1,BROK)
NOTCPU = NOR (BM1,BM2)
BR2EN = NOR (BROK,BRL2)
NDIR1 = XOR (RW,BM1)
NBR1 = INP (NBR1)
NBR2 = INP (NBR2)
NBGACK = INP (NBGACK)
RW = INP (RW)
BM1EN = INP (BM1EN)
NA0 = INP (NA0)
NB0 = INP (NB0)
NA1 = INP (NA1)
NB1 = INP (NB1)
NA2 = INP (NA2)
NB2 = INP (NB2)
FC2 = INP (FC2)
```

```
FC1 = INP (FC1)
FC0 = INP (FC0)
AS = NOT (NAS)
A2 = NOT (NA2)
A2EB2 = NOT (NA2EB2)
A1GB1 = AND (A1,NB1)
A1EB1 = NOT (NA1EB1)
A0GB0 = AND (A0,NB0)
BRL1 = NORF (M1,CLK,RESET,GND)
BROK = NOR (BRL1,BRL2)
BRL2 = NORF (M2,CLK,RESET,GND)
NAS = INP (NAS)
NA2EB2 = XOR (NA2,NB2)
A1 = NOT (NA1)
NA1EB1 = XOR (NA1,NB1)
A0 = NOT (NA0)
M1 = OR (SBR1,SFB1)
CLK = INP (8MHz)
RESET = NOT (NRESET)
M2 = OR (SBR2,SFB2)
SBR1 = AND (NBR1,NBOTH)
SFB1 = AND (BR2EN,BOTH)
NRESET = INP (NRESET)
SBR2 = AND (NBR2,NBOTH)
SFB2 = AND (NBR2EN,BOTH)
NBOTH = NOT (BOTH)
BOTH = NOR (NBR1,NBR2)
NBR2EN = NOT (BR2EN)
END$
```

Part of report file, (.RPT):

```
ALTERA Design Processor Utilization Report

 ***** Design implemented successfully

E.Mills.
Research
July 8, 1987

3
EP1210
Modification due to new EA2 board.
NETMAP Version 3.0, Baseline 15, 8/3/1985

              EP1210
            - - - - -
      8MHz -| 1    40|- Vcc
       NA2 -| 2    39|- Vcc
       NA1 -| 3    38|- NBR1
       NA0 -| 4    37|- NBR2
       NB2 -| 5    36|- NBGACK
       NB1 -| 6    35|- RW
       NB0 -| 7    34|- ERRMASK
     NILP2 -| 8    33|- NBERR1
     NIPL1 -| 9    32|- FIX2
     NILP0 -|10    31|- RESERVED
      NENA -|11    30|- BM1
      NENB -|12    29|- NBERR
    STATUS -|13    28|- DIR1
       NBR -|14    27|- NBM2
      BG1D -|15    26|- DIR2
      NMON -|16    25|- BG2D
     BM2EN -|17    24|- NAS
     BM1EN -|18    23|- FC2
    NRESET -|19    22|- FC1
       GND -|20    21|- FC0
            - - - - -


**UNUSED RESOURCES**

      Name  Pin  Resource     MCell    PTerms

        -    31      -            2        10
        -    NA      -           15         8
```

```
          -    NA             -          16          8

**PART UTILIZATION**

97% Pins
89% MacroCells
20% Pterms
```

F.B G96 68010 to Transputer Interface EPLD.

The '**G96 68010 to Transputer Interface EPLD**' was used to create the decoding for the **Gespac GESMPU14** 68010 based G96 development card. The interface card that this was on, was designed to interface the 68010 card to a dual–port RAM area, which in turn interfaced to a transputer card, (mainly the Controller). Although the design of this unit was completed and the circuit entered into a pcb board routing package, the board was never produced. The pcb package was **RGRAPH** and was decommissioned after failing to produce useful results.

The design included this dual–port RAM interface along with 4 Inmos link Adaptors, an IEEE–488 instrument interface and 64K of static RAM. The EPLD provided the necessary decode logic for the address bus of the 68010 along with interrupt request logic for the Link Adaptors and the dual–port RAM.

Input file (.ADF):

```
E.Mills.
Research.
October 9, 1987

2
EP1210
68000 address decoder.
NETMAP Version 3.0, Baseline 15, 8/3/1985
PART: EP1210
INPUTS: A17, A18, A19, A20, A21, A22, NAS, NBUSY, NIACK,
        NLDS, A0, A1, A2, A8, A7, A9, NVPA, 10MHZ
OUTPUTS: NSRAM, NDPRAM, NIEEE, NVEN, NBUFEN, NLINK0,
         NLINK1, NLINK2, NLINK3, 5MHZ, DTACK
NETWORK:
NSRAM = CONF (NSRAM,VCC)
NDPRAM = CONF (NDPRAM,VCC)
NIEEE = CONF (NIEEE,VCC)
NVEN = CONF (NVEN,VCC)
NBUFEN = CONF (NBUFEN,VCC)
NLINK0 = CONF (NLINK0,VCC)
NLINK1 = CONF (NLINK1,VCC)
NLINK2 = CONF (NLINK2,VCC)
NLINK3 = CONF (NLINK3,VCC)
5MHZ,5MHZ = RORF (N5MHZ,10MHZ,GND,GND,VCC)
DTACK = CONF (DTACK,VCC)
```

```
NSRAM = NAND (RAM,A18,NA17)
NDPRAM = NAND (RAM,A18,A17)
NIEEE = NAND (PERIF,A7,NA8)
NVEN = NAND (IACK,NA2,A1,A0)
NVDTACK = NAND (LDS,IACK,NA2,A1,A0)
NLINKD = NAND (PERIF,A7,A8)
NBUFEN = AND (NRESTBE,NRAMBE)
NDPRMD = NAND (RAM,A18,A17,NBUSY)
NLINK0 = NAND (PERIF,A7,A8,NA1,NA0)
NLINK1 = NAND (PERIF,A7,A8,NA1,A0)
NLINK2 = NAND (PERIF,A7,A8,A1,NA0)
NLINK3 = NAND (PERIF,A7,A8,A1,A0)
N5MHZ = NOT (5MHZ)
10MHZ = INP (10MHZ)
RAM = NOR (NAS,A22,A21,A20,A19)
A18 = INP (A18)
NA17 = NOT (A17)
A17 = INP (A17)
PERIF = NOR (NVPA,A9)
A7 = INP (A7)
NA8 = NOT (A8)
IACK = NOT (NIACK)
NA2 = NOT (A2)
A1 = INP (A1)
A0 = INP (A0)
LDS = NOT (NLDS)
A8 = INP (A8)
NRESTBE = NAND (PERIF,A7)
NRAMBE = NAND (RAM,A18)
NBUSY = INP (NBUSY)
NA1 = NOT (A1)
NA0 = NOT (A0)
NAS = INP (NAS)
A22 = INP (A22)
A21 = INP (A21)
A20 = INP (A20)
A19 = INP (A19)
NVPA = INP (NVPA)
A9 = INP (A9)
NIACK = INP (NIACK)
A2 = INP (A2)
NLDS = INP (NLDS)
DTACK = NAND (NDPRMD,NLINKD,NSRAM,NVDTACK)
END$
```

Part of report file, (.RPT):

```
ALTERA Design Processor Utilization Report

 ***** Design implemented successfully

E.Mills.
Research.
October 9, 1987

2
EP1210
68000 address decoder.
NETMAP Version 3.0, Baseline 15, 8/3/1985

              EP1210
            - - - - -
    10MHZ -| 1    40|- Vcc
       A1 -| 2    39|- Vcc
       A0 -| 3    38|- A17
     NLDS -| 4    37|- A18
    NIACK -| 5    36|- A19
    NBUSY -| 6    35|- A20
      NAS -| 7    34|- A21
   NLINK2 -| 8    33|- A22
 RESERVED -| 9    32|- DTACK
   NDPRAM -|10    31|- NSRAM
   NBUFEN -|11    30|- NVEN
   NLINK0 -|12    29|- NLINK1
    NIEEE -|13    28|- RESERVED
 RESERVED -|14    27|- RESERVED
   NLINK3 -|15    26|- RESERVED
      GND -|16    25|- 5MHZ
      GND -|17    24|- A2
      GND -|18    23|- A8
     NVPA -|19    22|- A7
      GND -|20    21|- A9
            - - - - -


**UNUSED RESOURCES**

      Name  Pin  Resource     MCell    PTerms

         -    9         -        27        10
         -   14         -        22        10
         -    16            -         20         12
         -    17            -         19          4
         -    18            -         18          8
         -    26            -          7         10
         -    27            -          6          8
         -    28            -          5          6
         -    NA            -         13          8
         -    NA            -         14          8
         -    NA            -         15          8
         -    NA            -         16          8

**PART UTILIZATION**

78% Pins
57% MacroCells
7% Pterms
```

F.C Auto Gain EPLD.

The EPLD presented here is detailed in section 6.2.1. This EPLD calculates the gain required for optimum conversion by an 8-bit flash converter for intra-sample adaptation. Another task of this device is to control the operation of the signal multiplexer which is just before the gain stage.

Input file (.ADF):

```
E.Mills.
Research
July 19, 1988
AUTO_GAIN_PLA
4
EP1210
Auto Gain Logic. Gain latch section change.
NETMAP Version 3.0, Baseline 15, 8/3/1985
PART: EP1210
INPUTS: D7/38, D6/37, D5/36, D4/35, D3/34, D2/33, D1/7,
        NAtoD/1, ClrGain/6, NRdGain/5, READ/4, D0/3
OUTPUTS: Gain1/29, Gain0/31, Mux2/23, Mux1/30, Mux0/32,
         AD7/13, AD6/27, AD5/11, AD4/12, AD3/8, AD2/15,
         AD1/28, AD0/25, Gain2/17
NETWORK:
Gain1 = CONF (Gain1,VCC)
Gain0 = CONF (Gain0,VCC)
Mux2,MuxQ2 = ROIF (MuxD2,NAtoD,ClrGain,GND,VCC)
Mux1,MuxQ1 = ROIF (MuxD1,NAtoD,ClrGain,GND,VCC)
Mux0,MuxQ0 = ROIF (MuxD0,NAtoD,ClrGain,GND,VCC)
AD7 = CONF (MAD7,Rd)
AD6 = CONF (MAD6,Rd)
AD5 = CONF (MAD5,Rd)
AD4 = CONF (MAD4,Rd)
AD3 = CONF (MAD3,Rd)
AD2 = CONF (MAD2,Rd)
AD1 = CONF (MAD1,Rd)
AD0 = CONF (MAD0,Rd)
Gain2 = CONF (Gain2,VCC)
Gain1 = NAND (MuxQ,IGain1)
Gain0 = NAND (MuxQ,IGain0)
MuxD2 = XOR (MuxC2,MuxQ2)
NAtoD = INP (NAtoD)
ClrGain = INP (ClrGain)
```

```
MuxD1 = XOR (MuxC1,MuxQ1)
MuxD0 = XOR (MuxQ,MuxQ0)
MAD7 = AND (NRdGain,d7)
Rd = INP (READ)
MAD6 = AND (NRdGain,d6)
MAD5 = AND (NRdGain,d5)
MAD4 = AND (NRdGain,d4)
MAD3 = AND (NRdGain,d3)
MAD2 = OR (MAD21,MAD22)
MAD1 = OR (MAD11,MAD12)
MAD0 = OR (MAD01,MAD02)
Gain2 = NAND (MuxQ,IGain2)
MuxQ = NORF (MuxD,NAtoD,ClrGain,GND)
IGain1 = NORF (B,NAtoD,ClrGain,GND)
MuxC2 = AND (MuxQ,MuxQ0,MuxQ1)
MuxC1 = AND (MuxQ,MuxQ0)
NRdGain = INP (NRdGain)
d7 = INP (D7)
d6 = INP (D6)
d5 = INP (D5)
d4 = INP (D4)
d3 = INP (D3)
MAD21 = AND (NRdGain,d2)
MAD22 = AND (RdGain,IGain2)
MAD11 = AND (NRdGain,d1)
MAD12 = AND (RdGain,IGain1)
MAD01 = AND (NRdGain,d0)
MAD02 = AND (RdGain,IGain0)
IGain2 = NORF (C,NAtoD,ClrGain,GND)
C = NOR (B6,B5,B4,B3)
B = NOR (B6,B5,BC)
B6 = XOR (Nd7,d6)
B5 = XOR (Nd7,d5)
BC = AND (BA,BB)
MuxD = NOT (MuxQ)
A = NOR (B6,AB,AD,AF)
AB = AND (B4,AA)
AD = AND (B2,AC)
AF = AND (B0,AE)
d2 = INP (D2)
RdGain = NOT (NRdGain)
d1 = INP (D1)
d0 = INP (D0)
IGain0 = NORF (A,NAtoD,ClrGain,GND)
B4 = XOR (Nd7,d4)
B3 = XOR (Nd7,d3)
Nd7 = NOT (d7)
```

```
BA = OR (B2,B1)
BB = NOR (B4,B3)
AA = NOT (B5)
B2 = XOR (Nd7,d2)
AC = NOR (B5,B3)
B0 = XOR (Nd7,d0)
AE = NOR (B5,B3,B1)
B1 = XOR (Nd7,d1)
END$
```

Part of report file, (.RPT):

```
ALTERA Design Processor Utilization Report

 ***** Design implemented successfully

E.Mills.
Research
July 19, 1988
AUTO_GAIN_PLA
4
EP1210
Auto Gain Logic. Gain latch section change.
NETMAP Version 3.0, Baseline 15, 8/3/1985

              EP1210
             - - - - -
     NAtoD -| 1    40|- Vcc
       GND -| 2    39|- Vcc
        D0 -| 3    38|- D7
      READ -| 4    37|- D6
   NRdGain -| 5    36|- D5
   ClrGain -| 6    35|- D4
        D1 -| 7    34|- D3
       AD2 -| 8    33|- D2
  RESERVED -| 9    32|- Gain2
  RESERVED -|10    31|- RESERVED
       AD5 -|11    30|- Gain1
       AD4 -|12    29|- Gain0
       AD7 -|13    28|- AD3
  RESERVED -|14    27|- AD6
       AD1 -|15    26|- RESERVED
  RESERVED -|16    25|- AD0
      Mux1 -|17    24|- RESERVED
  RESERVED -|18    23|- Mux0
  RESERVED -|19    22|- Mux2
       GND -|20    21|- RESERVED
             - - - - -


**UNUSED RESOURCES**

      Name  Pin  Resource    MCell    PTerms

        -    2       -          -         -
        -    9       -         27        10
```

```
          -   10          -         26          8
          -   14          -         22         10
          -   16          -         20         12
          -   18          -         18          8
          -   19          -         17          8
          -   21          -         12          8
          -   24          -          9         12
          -   26          -          7         10
          -   31          -          2         10

**PART UTILIZATION**

70% Pins
64% MacroCells
19% Pterms
```

F.D Controller Board Decoder Unit EPLD.

This EPLD performs the task of decoding the transputer's address bus onboard the Controller Unit and generates the chip selects for the static RAMs. The other task of this device is to control the buffers on the board. When the transputer bus is requested, and the request granted, the EPLD forces certain buffers into their tri-state mode and reverses the operation of some of the other buffers. This has the effect of releasing the busses of the transputer and allows external bus masters to use the bus logic and RAM devices onboard the Controller itself.

Input file (.ADF):

```
E.Mills
Flowmeter Research.
March 13, 1989

2
EP1210
Main PCB address decoder.
NETMAP Version 3.0, Baseline 15, 8/3/1985
PART: EP1210
INPUTS: A31, NA30, NA29, NA28, NA27, NA26, NA25, NA24, NA23, NA22,
        NA21, NA20, ROM, NA19, NA18, NA17, NS0, 10MHz, NRD, MEMGNT
OUTPUTS: EXT, NSRAM7/27, NSRAM6/30, NSRAM5/8, NSRAM4/15, NSRAM3/11,
         NSRAM2/12, NSRAM1/28, NSRAM0/29, 5MHz, DBDIR
NETWORK:
EXT = CONF (EXT,VCC)
NSRAM7 = CONF (NSRAM7,VCC)
NSRAM6 = CONF (NSRAM6,VCC)
NSRAM5 = CONF (NSRAM5,VCC)
NSRAM4 = CONF (NSRAM4,VCC)
NSRAM3 = CONF (NSRAM3,VCC)
NSRAM2 = CONF (NSRAM2,VCC)
NSRAM1 = CONF (NSRAM1,VCC)
NSRAM0 = CONF (NSRAM0,VCC)
5MHz,5MHz = CORF (N5MHz,10MHz,GND,GND,VCC)
DBDIR = CONF (DBDIR,VCC)
EXT = NOR (MNEG,NHEN)
NSRAM7 = OR (E7,NHEN)
NSRAM6 = OR (E6,NHEN)
NSRAM5 = OR (E5,NHEN)
NSRAM4 = OR (E4,NHEN)
NSRAM3 = OR (E3,NLEN)
```

```
NSRAM2 = OR (E2,NLEN)
NSRAM1 = OR (E1,NLEN)
NSRAM0 = OR (E0,NLEN)
N5MHz = NOT (5MHz)
10MHz = INP (10MHz)
DBDIR = XOR (BGACK,NRD)
MNEG = NAND(A31,NA30,NA29,NA28,NA27,NA26,NA25,NA24,NA23,NA22,NA21,NA20)
NHEN = OR (HMEM,NS0)
E7 = NAND (C,B,A)
E6 = NAND (C,B,NA)
E5 = NAND (C,NB,A)
E4 = NAND (C,NB,NA)
E3 = NAND (NC,B,A)
NLEN = OR (MNEG,NS0)
E2 = NAND (NC,B,NA)
E1 = NAND (NC,NB,A)
E0 = NAND (NC,NB,NA)
BGACK = INP (MEMGNT)
NRD = INP (NRD)
A31 = INP (A31)
NA30 = INP (NA30)
NA29 = INP (NA29)
NA28 = INP (NA28)
NA27 = INP (NA27)
NA26 = INP (NA26)
NA25 = INP (NA25)
NA24 = INP (NA24)
NA23 = INP (NA23)
NA22 = INP (NA22)
NA21 = INP (NA21)
NA20 = INP (NA20)
HMEM = AND (HMNEG,HMPOS)
NS0 = INP (NS0)
NC = INP (NA19)
NB = INP (NA18)
NA = INP (NA17)
A = NOT (NA)
B = NOT (NB)
C = NOT (NC)
HMNEG = OR (MNEG,ROM)
HMPOS = NAND (ROM,MPOS)
ROM = INP (ROM)
MPOS = NOR (A31,NA30,NA29,NA28,NA27,NA26,NA25,NA24,NA23,NA22,NA21,NA20)
END$
```

Part of report file, (.RPT):

```
ALTERA Design Processor Utilization Report

 ***** Design implemented successfully

E.Mills
Flowmeter Research.
March 13, 1989

2
EP1210
Main PCB address decoder.
NETMAP Version 3.0, Baseline 15, 8/3/1985

              EP1210
            - - - - -
     10MHz -| 1    40|- Vcc
      NA20 -| 2    39|- Vcc
      NA21 -| 3    38|- A31
      NA22 -| 4    37|- NA30
      NA23 -| 5    36|- NA29
      NA24 -| 6    35|- NA28
      NA25 -| 7    34|- NA27
    NSRAM5 -| 8    33|- NA26
  RESERVED -| 9    32|- DBDIR
  RESERVED -|10    31|- RESERVED
    NSRAM3 -|11    30|- NSRAM6
    NSRAM2 -|12    29|- NSRAM0
       EXT -|13    28|- NSRAM1
  RESERVED -|14    27|- NSRAM7
    NSRAM4 -|15    26|- RESERVED
       GND -|16    25|- 5MHz
    MEMGNT -|17    24|- ROM
       NRD -|18    23|- NA19
       NS0 -|19    22|- NA18
       GND -|20    21|- NA17
            - - - - -

**UNUSED RESOURCES**

      Name  Pin  Resource     MCell    PTerms

        -     9        -         27        10
        -    10        -         26         8
        -    14        -         22        10
        -    16        -         20        12
```

```
         -   26         -          7        10
         -   31         -          2        10
         -   NA         -         13         8
         -   NA         -         14         8
         -   NA         -         15         8
         -   NA         -         16         8

**PART UTILIZATION**

83% Pins
64% MacroCells
7% Pterms
```

G. Partial Software Listings.

This section contains some of the 68000 based software developed for the Atari and the final host 68000 system. It starts with the peak detection and integration software for the flow system.

Each software section starts on a new page.

G.A Main 68000 Flow Processing.

```
* This is an N signal peak detector using registers only to store
* the peaks, and memory to store the address of the peaks.

* D4.W must hold N where 0<=N<=7 and is one less than the number of
*      signals.
* D5.W must hold x where x is one less than the number of data points.
* A0.L must contain the start address of the data block.

* The signal data must be interleaved such that the memory for 3
* signals is in the following format:

*           I0
*           I1
*           I2
*           I0
*           I1       etc

* The routine will return the peaks in the following format:

*    31   16|15   0
*    --------------
*  D0:   I7 :  I0 :
*  D1:   I6 :  I1 :
*  D2:   I5 :  I2 :
*  D3:   I4 :  I3 :
*    --------------

* D4-D7    unaffected.
* A0/A2-A7 unaffected.

* A1.L will point to the address of the stored peak addresses.

PEAKDET  MOVEM.L D4-D6/A0,-(SP)      Save registers.
         LEA PEAKADR(PC),A1          Point to peak address table.
         BSR PKPRESET                Preset peaks.
         SWAP D5
         MOVE.W D4,D5                Set D5 top to N.
         SWAP D5
PEAKL1   MOVE.W D4,D6                Copy N to a working register.
         SWAP D4                     Save counter in top of D4
         MOVE.W (A0)+,D4             Read data.
         BSR PKCHK                   Check for peak.
         BNE NOTPEAK                 If NE then not a peak.
         MOVE.W D4,D0              Update with new peak value.
         LSL.L #2,D6               D6=N*4
         MOVE.L A0,0(A1,D6.W)      Save address in peak address table.
NOTPEAK  BSR SHUFFLE               Shuffle register round.
         SWAP D4                   Get at counter.
         DBRA D4,PEAKL1            Loop until count finished.
         SWAP D5                   D5.W = master copy of N.
         MOVE.W D5,D4              Reset D4 back to N.
         MOVEQ #7,D6               Set for calculation.
         SUB.W D5,D6               Determine who to reorder peaks.
         SWAP D5                   D5.W now back to number of items.
PEAKROL  DBRA D6,PEAKRO            If count <> -1 then reorder.
         DBRA D5,PEAKL1            Repeat process until items = -1.
         MOVEM.L (SP)+,D4-D6/A0    Restore registers.
         RTS
PEAKRO   BSR SHUFFLE               Shuffle peaks once.
         BRA PEAKROL               Go back and check again.

PEAKADR  DCB.L 8,0

* This routine will preset the peaks for the three different types of
* peak detection.

* If D7.B = 0 Then it sets for largest absolute value peak detection.
* If D7.B = + Then it sets for largest positive value peak detection.
* If D7.B = - Then it sets for largest negative value peak detection.

PKPRESET MOVEQ #0,D0               Clear as for absolute.
         TST.B D7                  Check control.
         BEQ PKSET                 Set for absolute.
         BPL PKSET                 Set for positive.
PKSET    MOVE.L D0,D1
         MOVE.L D0,D2
         MOVE.L D0,D3              Preset them all.
         RTS


* This routine will detect one of three types of peaks.

* If D7.B = 0 Then it marks the largest absolute value.
* If D7.B = + Then it marks the largest positive value.
* If D7.B = - Then it marks the largest negative value.

* D0.W contains the old peak value and D4.W contains the new data item.

* If D4.W should become then new peak then the routine returns the EQ
* flag, otherwise it returns NE.

* The old peak in D0 is not affected neither is D4.

PKCHK     MOVEM.L D0/D4,-(SP)          Save values.
          TST.B D7                     Test control.
          BEQ PKCHKABS                 Absolute detection required.
          BPL PKCHKMAX                 Positive detection required.
PKCHKMIN  BSR PKMIN                    Go for negative peak detection.
          BRA PKCHKEND                 Ok end routine.
PKCHKABS  BSR PKABS                    Go for absolute peak detection.
          BRA PKCHKEND                 Ok end routine.
PKCHKMAX  BSR PKMAX                    Go for positive peak detection.
PKCHKEND  MOVEM.L (SP)+,D0/D4          Restore values. Flags unaffected.
          RTS

* Negative peak detection.

PKMIN     TST.W D4                     Check data read.
          BPL PKMINNO                  Not negative so now minimum.
          CMP.W D0,D4                  Check against current peak.
          BGE PKMINNO                  Data GE to peak so no new min.
          CMP.B D0,D0                  Set EQ to show new peak.
          RTS
PKMINNO   ANDI #$FB,CCR                Set NE to show no new peak.
          RTS

* Positive peak detection.

PKMAX     TST.W D4                     Test data.
          BMI PKMAXNO                  If negative then no new peak.
          CMP.W D0,D4                  Check against current peak.
          BLE PKMAXNO                  If LE then no new peak.
          CMP.B D0,D0                  Set EQ to show new peak.
          RTS
PKMAXNO   ANDI #$FB,CCR                Set NE to show no new peak.
          RTS

* Absolute peak detection.

* NOTE:
*         No data correction needed as header for peak detection saves
*         the data values and restores them.

PKABS     TST.W D4                     Check data.
          BMI PKABSDM                  Negative data so look for - peak.
          TST.W D0                     Check peak.
```

```
         BMI PKABSPM              Data and peak not positive.
         BSR PKMAX                Find max peak as both positive.
         RTS
PKABSPM  NEG.W D0                 Make peak positive.
         BSR PKMAX                Look for max peak as both now pos.
         RTS
PKABSDM  TST.W D0                 Check current peak.
         BPL PKABSMP              Data and peak not negative.
         BSR PKMIN                Find min peak as both negative.
         RTS
PKABSMP  NEG.W D0                 Make peak negative.
         BSR PKMIN                Look for min peak as both now neg.
         RTS

* This subroutine shuffles the register around so that

*   --------------                  -------------
*  D0:   I7 :  I0 :                 :  I0 :  I1 :
*  D1:   I6 :  I1 :     becomes     :  I7 :  I2 :
*  D2:   I5 :  I2 :                 :  I6 :  I3 :
*  D3:   I4 :  I3 :                 :  I5 :  I4 :
*   --------------                  -------------

SHUFFLE  MOVE.W D6,-(SP)      Save register.
         MOVE.W D0,D6
         MOVE.W D1,D0
         MOVE.W D2,D1
         MOVE.W D3,D2
         SWAP D0
         SWAP D1
         SWAP D2
         MOVE.W D2,D3
         MOVE.W D1,D2
         MOVE.W D0,D1
         MOVE.W D6,D0
         SWAP D0
         SWAP D1
         SWAP D2
         SWAP D3
         MOVE.W (SP)+,D6      Restore register.
         RTS
```

G.B Atari Transputer Development Workstation.

The following listing shows the main bulk of the Atari Transputer Development Workstation software which was used in conjunction with the Inmos link adaptor unit described in section 5.3.

The software makes reference to service routines which were formed into libraries and included at compile time. One of these libraries is an equation handler which allows the user to enter equations which may contain predefined labels, in response to numerical questions.

```
*         Transputer Development Workstation.
*         -----------------------------------

*         In this program an abort mechanism operates in all sections
*         were if the program is waiting for a link operation the
*         operator may terminate this process and return to main menu
*         by pressing X. A lowercase X is also recognised.

*         To save time and space subroutines are directly terminatable.
*         This is accomplished by restoring the stack pointer to its
*         original value as on entry to the program.

*         The code is all position independent and the program
*         terminates to desktop.


          LEA ABORTSP(PC),A0       Point to abort stack pointer copy.
          MOVE.L SP,(A0)           Save stack pointer.
MENU      LEA $FA0000,A4           Link 0.
          LEA $FA2000,A5           Link 1.
          LEA $FA4000,A6           Status register.
          LEA LINK0(PC),A3         Point to link parameter block.
          LEA T414T1(PC),A0
          BSR WRITESTR
WHICHOPT  BSR INKEY0               Wait for response from operator.
          CMPI.B #"0",D0
          BLT WHICHOPT
          CMPI.B #"9",D0
          BGT WHICHOPT
          MOVE.B D0,D7             Save option chosen.
          BSR WRITECHR             Echo chosen option to screen.
          BSR NEWLINE
          CMPI.B #"8",D7
          BEQ SETTYPE
```

```
         CMPI.B #"9",D7
         BEQ EXIT
         CMPI.B #"5",D7
         BEQ MONITOR
         LEA T414T2(PC),A0
         BSR WRITESTR
WHICHLNK BSR INKEY0
         CMPI.B #"0",D0
         BEQ START
         CMPI.B #"1",D0
         BNE WHICHLNK
         LEA LINK1(PC),A3        Set for link 1 communication.
START    LEA LINKNAME(PC),A0
         MOVE.B D0,(A0)          Save name of link.
         BSR WRITECHR            Write chosen link number.
         BSR NEWLINE
         LEA LINKLIST(PC),A0
         BSR WRITESTR
         BSR LINKINFO            Display information about link.
         CMPI.B #"1",D7
         BEQ DSKTRAN
         BSR SETUP
         CMPI.B #"6",D7
         BEQ ASCIIMON
         BSR BLOCK
         CMPI.B #"2",D7
         BEQ EXAMINE
         CMPI.B #"3",D7
         BEQ FILLT4
         CMPI.B #"4",D7
         BEQ TESTT4
         BRA SEARCH

T414T1   DC.B 13,10,10,10,10
         DC.B 13,10,"Transputer Workstation."
         DC.B 13,10,10
         DC.B 13,10,"Options:"
         DC.B 13,10
         DC.B 13,10,"1: File Transfer."
         DC.B 13,10
         DC.B 13,10,"2: Memory Examine."
         DC.B 13,10
         DC.B 13,10,"3: Memory Fill."
         DC.B 13,10
         DC.B 13,10,"4: Memory Test."
         DC.B 13,10
         DC.B 13,10,"5: Monitor Links."
         DC.B 13,10
         DC.B 13,10,"6: Terminal monitor."
         DC.B 13,10
         DC.B 13,10,"7: Search For Data."
         DC.B 13,10
         DC.B 13,10,"8: Define transputer type."
         DC.B 13,10
         DC.B 13,10,"9: Exit."
         DC.B 13,10
         DC.B 13,10,"Which One? (1-9): ",0
T414T2   DC.B 13,10,"Which link is to be used for operation? (0 or 1):"
         DC.B 0
         CNOP 0,4
LINKADDR DC.L $00FA0000          Link port address
LINKMASK DC.L $00000001          Status register masks
LINKLIST DC.B 13,10,10,"Operating on link "
LINKNAME DC.B "n"
         DC.B ":",13,10,10,0
         CNOP 0,4


*        Setup address pointers and status register masks.

*        A4.L points to link 0.
*        A5.L points to link 1.
*        A6.L points to status register.
*        A3.L points to start of active link table.
*        A2.L points to active link.
*        D3.L holds status register mask.

SETUP    LEA $FA0000,A4        Link 0.
         LEA $FA2000,A5        Link 1.
         LEA $FA4000,A6        Status register.
         MOVEA.L 18(A3),A2     Point to port.
         MOVE.L 22(A3),D3      Load status register mask.
         RTS

TRANSFER BSR SETUP             Setup pointers and masks.
         LEA BUFFER(PC),A1     Point to disk buffer.
         MOVE.L D1,D2          Copy amount read.
         SUBQ.L #1,D2          Drop count by one for transfer.
TRANS    BSR CHKLINKS          Check for data received.
         MOVE.B (A1)+,D6       Copy data from buffer into transfer reg.
         BSR TRANSMIT          Output data on link.
         BSR INKEY0            Check for abort. Don't wait for response.
         ORI.B #32,D0          Force to lowercase.
         CMPI.B #"x",D0        Check for valid abort character.
         BEQ ABORTOP          Leave if requested.
         DBRA D2,TRANS        Repeat until buffer cleared.
         RTS

CHKLINKS BTST #0,1(A6)        Check link 0.
         BEQ NR0
         BSR REC0
NR0      BTST #2,1(A6)        Check link 1.
         BEQ NR1
         BSR REC1
NR1      RTS

*        Monitor both links for any input.

MONITOR  BSR SETUP
         LEA MONT1(PC),A0
         BSR WRITESTR
MON1     BSR CHKLINKS
         BSR INKEY0           Check for abort. Don't wait for response.
         ORI.B #32,D0         Force lowercase.
         CMPI.B #"x",D0       Check for valid abort character.
         BEQ ABORTOP          Leave if requested.
         BRA MON1

MONT1    DC.B 13,10,"Monitoring the links:",13,10,10,0

         CNOP 0,4

*        Blind read of link 0 with echo to screen.

REC0     LEA RC0(PC),A0
         BSR WRITESTR
         MOVE.B 1(A4),D4
         BSR HEXBYTE
         BSR NEWLINE
         RTS

RC0      DC.B "Received on link 0 : ",0

         CNOP 0,4

*        Blind read of link 1 with echo to screen.

REC1     LEA RC1(PC),A0
         BSR WRITESTR
         MOVE.B 1(A5),D4
         BSR HEXBYTE
         BSR NEWLINE
         RTS

RC1      DC.B "Received on link 1 : ",0

         CNOP 0,4

*        Transmit a byte down the active link with echo to screen.

*        D6.B holds the byte.

TRANSMIT BSR WAITLNK                   Wait until active link clear.
TRNOK    LEA TRNDATA(PC),A0            Point to echo text.
         BSR WRITESTR                  Display it.
         MOVE.B D6,D4                  Copy byte to output.
         BSR HEXBYTE                   Display copy.
         BSR NEWLINE                   Terminate display line.
         MOVEQ #0,D0                   Clear modified data register.
         MOVE.B D6,D0                  Copy byte to transmit.
         LSL.L #1,D0                   Modify it.
         MOVE.B 0(A2,D0.W),D0          Output it on link.
         RTS

TRNDATA  DC.B "Transferring to link: ",0

         CNOP 0,4

*        Macro for clearing a disk filename buffer.

CLRNAME  MACRO
         LEA FNAMEI(PC),A0
         MOVEQ #21,D7
CLRNLOOP CLR.W (A0)+
         DBRA D7,CLRNLOOP
         ENDM

*        Routine for disk file transfer to active link.

DSKTRAN  BSR SUPERMODE
         LEA BUFFER(PC),A2   Point to disk buffer.
         MOVE.L #1024,D3     Set for 1K buffer.
MLOOP    CLRNAME             Clear filename.
         LEA MTEXT2(PC),A0   Point to next prompt.
         BSR WRITESTR        Display it.
         LEA FHEADI(PC),A0   Point to header.
         BSR READSTR         Read filename 1.
         TST.B 1(A0)         Test length of name.
         BEQ MENU            Leave if null.          No files open.
         LEA FNAMEI(PC),A1   Point to actual name.
         BSR OPENIN          Open file.
         TST.L D0            Check for an error.
         BMI MERGEXIT        Trap errors.            No files open.
         BSR WAIT            Wait for operator.
MMORE    MOVE.L D3,D1        Set max buffer size.
         BSR DISKRD          Read file.
         TST.L D0            Check for a error.
         BMI TEXIT           Error found.            File open.
         BEQ TEXIT           None read.
         MOVE.L D0,D1        Copy number of bytes read into buffer size.
         BSR TRANSFER        Write block
         CMP.L D3,D1         Check to see if buffer was full.
         BEQ MMORE           Merge more if it was.
TEXIT    BSR CLOSE           Close file.
MERGEXIT BSR USERMODE        Back to user mode.
         BRA MONITOR         Monitor links.

MTEXT2   DC.B "File Transfer.",13,10
         DC.B 13,10,"Input file (RETURN to exit) ",0

         CNOP 0,4

FHEADI   DC.B 40,0
FNAMEI   DCB.B 46,0

         CNOP 0,4

BUFFER   DS.B 1030

LAST     DC.L 0

         CNOP 0,4

*        Obtain block parameters allowing equation inputs.

*        On exit D2.L contains start address.
*                D1.L contains longword count.

*        All other registers are unaffected.

BLOCK    MOVE.L D3,-(SP)      Save working register.
         MOVEQ #0,D3          Clear all of D3 for mask formation.
         SUB.L 12(A3),D3      Form mask for round down of first address.
         LEA BLOCKT1(PC),A0   Point to first question.
         BSR WRITESTR         Display it.
         BSR BLOCKPG           Get parameter in D1.L.
         AND.L D3,D1           Round down start address.
         MOVE.L D1,D2          Save result in D2.
         LEA BLOCKT2(PC),A0    Point to next point.
         BSR WRITESTR          Ask question.
         BSR BLOCKPG           Get parameter.
         BSR NEWLINE           Start a new line.
         SUB.L D2,D1           Calculate length of block required.
         MOVE.L 12(A3),D3      Get number of bytes in word.
         ADD.L D3,D1           Increase byte count to pass next word.
         LSR.B #1,D3           Divide no. of bytes by two.
         LSR.L D3,D1           Form word count by number of bytes / 2.
         MOVE.L (SP)+,D3       Restore register.
         RTS

BLOCKT1  DC.B 13,10,"Start address = ",0
BLOCKT2  DC.B 13,10,"End   address = ",0

BLOCKPG  MOVEM.L D7/A1,-(SP)       Save working registers.
         MOVEA.L A0,A1             Copy prompt address.
BLOCKP   LEA TXTEQTN(PC),A0        Point to parameter area.
         BSR READSTR               Read in a string.
         MOVEQ #0,D7               Clear offset pointer.
         MOVE.B 1(A0),D7           Obtain length of string.
         MOVE.B #13,2(A0,D7.W)     Add CR to equation.
         LEA EQTNDAT(PC),A0        Point to the parameter equation.
         BSR EQTN                  Process as an equation.
         BEQ BLOCKPX               Exit if no error.
         LEA BEQTNER(PC),A0        Point to error text.
         BSR WRITESTR              Display it.
         MOVEA.L A1,A0             Restore prompt address.
         BSR WRITESTR              Re-display prompt.
         BRA BLOCKP                Go back and re-try.
BLOCKPX  MOVEM.L (SP)+,D7/A1       Restore registers.
         RTS

BEQTNER  DC.B 13,10,"Error in input!"
         DC.B 13,10,10,"Re-do.",13,10,0
TXTEQTN  DC.B 100,0
EQTNDAT  DCB.B 110,13

         CNOP 0,4

*        Wait for active link to clear.

WAITLNK  LEA WAITLT(PC),A0     Point to text in case it is needed.
         BTST D3,1(A6)         Check link status.
         BEQ WLNKOVER           Leave as link ready.
         BSR WRITESTR           Write message once.
WLNKLOOP BSR CHKLINKS           Check links for received data.
         BSR INKEY0             Check for abort. Don't wait for response.
         ORI.B #32,D0           Force lowercase.
         CMPI.B #"x",D0         Check for valid abort character.
         BEQ ABORTOP            Leave if requested.
         BTST D3,1(A6)          Check link status again.
         BNE WLNKLOOP           Repeat checks until link clear.
WLNKOVER RTS

WAITLT   DC.B 13,10,"Waiting for link to clear.",13,10,0

         CNOP 0,4

*        Transmit longword down link.

*        D1.L contains the data to be transmitted.

*        Registers are not affected by this routine.

TRNLONG  MOVEM.L D0-D2/A0,-(SP)  Save registers.
         MOVE.W 16(A3),D2        Set counter for n byte output.
TMLONG   BSR WAITLNK             Wait for link to clear.
         MOVEQ #0,D0             Clear data carrier.
         MOVE.B D1,D0            Copy active data byte into carrier.
         LSL.W #1,D0             Adjust data for from offset address.
         MOVE.B 0(A2,D0.W),D0    Output data.
         ROR.L #8,D1             Shift long data to point to next byte.
         DBRA D2,TMLONG          Repeat until all bytes out.
         MOVEM.L (SP)+,D0-D2/A0  Restore registers.
         RTS

*        Peek at a memory location.

*        D1.L holds address of memory location.
*        D4.L returns with the data.

*        All other registers are unaffected.

*        If abort is requested during PEEK then this routine and
*        the calling routine is terminated.

PEEK     MOVEM.L D2/A0,-(SP)     Save registers.
         BSR WAITLNK             Wait for active link.
         MOVE.B 2(A2),D0         Send a control byte of 1 to link.
         BSR TRNLONG             Transmit D1.L as the address.
         MOVE.W 16(A3),D2         Set for n bytes to receive.
         SWAP D3                  Change mask to receive mask.
         MOVEQ #0,D4              Clear received data register.
PEEKW    BSR INKEY0               Get chr from keyboard, no wait.
         ORI.B #32,D0             Change to lowercase.
         CMPI.B #"x",D0           Check for abort key.
         BEQ ABORTOP              Leave if aborted.
         BTST D3,1(A6)            Check link for answer.
         BEQ PEEKW                Wait until it is received.
         MOVE.B 1(A2),D4          Read in new part of answer.
         ROR.L #8,D4              Shift answer.
         DBRA D2,PEEKW            Repeat until word read back.
         MOVE.W 16(A3),D2         Set for n bytes shift.
PEEKSR   ROR.L #8,D4              Shift answer.
         DBRA D2,PEEKSR           Repeat until word reordered.
         SWAP D3                  Restore status register mask.
         MOVEM.L (SP)+,D2/A0      Restore.
         RTS

*        Poke a memory location.

*        D1.L holds the address.
*        D6.L holds the data.

*        Registers are unaffected.

POKE     BSR WAITLNK              Wait for link to clear.
         MOVE.B 0(A2),D0          Send a control byte of 0 to link.
         BSR TRNLONG              Transmit D1.L as the address.
         EXG D1,D6                Swap address and data over.
         BSR TRNLONG              Transmit the data.
         EXG D1,D6                Restore address and data.
         RTS

EXAMINE  MOVE.L D1,D5             D5 = byte count.
         MOVE.L D2,D1             D1 = start address.
EXAMLOP  LEA EXMT1(PC),A0
         BSR WRITESTR
         MOVE.L D1,D4             Copy address for displaying.
         BSR HEXLONG              Display it.
         BSR PEEK                 Read location.
         LEA EXMT2(PC),A0         Point to result text.
         BSR WRITESTR             Display text.
         BSR HEXLONG              Display result found.
         ADD.L 12(A3),D1          Advance address by byte count.
         SUBQ.L #1,D5             Reduce byte count.
         BNE EXAMLOP              Repeat until cycle complete.
         LEA EXMT3(PC),A0        Point to 'completed' text.
         BSR WRITESTR            Display text.
         BSR WAIT                Wait for operator.
         BRA MENU                Return to main menu.

EXMT1    DC.B 13,10,"Peeking address ",0
EXMT2    DC.B " found ",0
EXMT3    DC.B 13,10,10,"Cycle complete."
         DC.B 13,10,10,0

         CNOP 0,4

FILLT4   MOVE.L D1,D5         Copy word count.
         LEA FILLTX1(PC),A0
         BSR WRITESTR
         BSR BLOCKPG          Get fill parameter in D1.L
         BSR NEWLINE
         BSR NEWLINE
         MOVE.L D1,D6         Save data in D6.L
         MOVE.L D2,D1         D1 = start address.
FILLLOP  LEA FILLTX2(PC),A0
         BSR WRITESTR
         MOVE.L D1,D4         Copy address for displaying.
         BSR HEXLONG          Display it.
         BSR POKE             Poke address.
         BSR INKEY0           Check for abort. Don't wait for response.
         ORI.B #32,D0         Force character to lowercase.
         CMPI.B #"x",D0       Check for abort.
         BEQ ABORTOP          Leave if requested.
         ADD.L 12(A3),D1      Advance address by count bytes.
         SUBQ.L #1,D5         Reduce word count.
         BNE FILLLOP          Repeat until cycle complete.
         LEA EXMT3(PC),A0     Set text to finish.
         BSR WRITESTR         Display it.
         BSR WAIT             Wait for operator.
         BRA MENU             Return to main menu.

FILLTX1  DC.B 13,10,"Data to fill memory with = ",0
FILLTX2  DC.B 13,"Filling address ",0

         CNOP 0,4

TESTT4   MOVEQ #0,D6
         MOVE.L D1,D5          D5 = word count.
         MOVE.L D2,D1          D1 = start address.
         MOVE.L D5,D7          Copy of byte count.
         BSR NEWLINE
         LEA TXTERRC(PC),A1      Point to error counter.
         MOVE.L D6,(A1)          Clear counter.
         MOVE.L D6,4(A1)         Clear control.
TESTLOP  LEA TXTT1(PC),A0
         BSR WRITESTR
         MOVE.L D1,D4            Copy address for displaying.
         BSR HEXLONG             Display it.
         LEA TXTT12(PC),A0
         BSR WRITESTR
         MOVE.L D6,D4
         BSR HEXLONG             Display data being used for test.
         BSR POKE                Poke address.
         BSR PEEK                Peek same address.
         CMP.L D4,D6             Check poke = peek.
         BNE TT4ERR              Error.
         ADD.L 12(A3),D1         Advance address by byte count.
         SUBQ.L #1,D5            Reduce longword count.
         BNE TESTLOP             Repeat until cycle complete.
         MOVE.L D2,D1            Restore base address.
         MOVE.L D7,D5            Restore longword count.
         ADDQ.L #1,D6            Increase data value.
         BNE TESTLOP             Repeat until counter folds over to zero.
         LEA TXTT2(PC),A0        Point to 'completed'.
         BRA TESTFIN             Leave as test cycle complete.
TT4ERR   LEA TXTT3(PC),A0        Point to error text.
         BSR WRITESTR            Display it.
         BSR HEXLONG             Display value just read from memory.
         BSR NEWLINE             Terminate line.
         ADD.L 12(A3),D1         Advance address by byte count.
         SUBQ.L #1,D5            Reduce word count.
         BNE TT4ERRX             Leave if counters still valid.
         LEA TXTT2(PC),A0        Point to 'completed' in case needed.
         MOVE.L D2,D1            Restore base address.
         MOVE.L D7,D5            Restore word count.
         ADDQ.L #1,D6            Increase data value.
         BEQ TESTFIN             Leave as no counter foldover.
TT4ERRX  MOVE.L (A1),D4          Get number of errors.
         BNE TERRCNT             Counting errors.
         TST.B 4(A1)             Check if options on errors chosen.
         SEQ 4(A1)               Set if not chosen.
         BNE TERRNC              No counting required.
         LEA TXTT6(PC),A0        Point to error options.
         BSR WRITESTR            Display them.
TERROPT  BSR INKEY               Get result.
         CMPI.B #"3",D0          Check for exit.
         BEQ MENU                Go back to main menu if required.
         CMPI.B #"2",D0          Check for normal action.
         BEQ TERRNC             Treat as normal.
         CMPI.B #"1",D0         Check for counting.
         BNE TERROPT            Invalid option, repeat.
TERRCNT  ADDQ.L #1,D4           Increase count.
         MOVE.L D4,(A1)         Store result.
         BSR INKEY0             Check keyboard.
         ORI.B #32,D0           Force lowercase.
         CMPI.B #"r",D0         Test for exit condition.
         BNE TESTLOP            Continue testing.
         MOVE.L (A1),D4         Get number of errors.
         BEQ MENU               No errors so move to main menu.
         LEA TXTT7(PC),A0       Point to report.
         BSR WRITESTR           Display it.
         BSR HEXLONG            Display total number of errors.
         BSR WAIT               Wait for operator.
         BRA MENU               Return to main menu.
TERRNC   LEA TXTT5(PC),A0       Point to normal error message.
TESTFIN  BSR WRITESTR           Display appropriate text.
         BSR INKEY              Wait for response to question.
         ORI.B #32,D0           Force lowercase.
         CMPI.B #"x",D0         Test for exit condition.
         BEQ MENU               Exit.
         BRA TESTLOP            Else continue testing.

TXTT1    DC.B 13,"Testing address ",0
TXTT12   DC.B " with ",0
TXTT2    DC.B 13,10,10,"Cycle complete."
         DC.B 13,10,10,"No errors."
         DC.B 13,10,10,"Press X to exit or any other key to restart."
         DC.B 13,10,0
TXTT3    DC.B ". ERROR: Found ",0
TXTT5    DC.B 13,10,10,"Press X to exit or any other key to continue."
         DC.B 13,10,0
TXTT6    DC.B 13,10,10
         DC.B "1: Ignore errors and continue until end.",13,10
         DC.B 10
         DC.B "2: Continue but stop at each error.",13,10
         DC.B 10
         DC.B "3: Exit memory test.",13,10
         DC.B 10,10
         DC.B "Which one? (1-3):",0
TXTT7    DC.B 13,10,10,"Total number of errors found = ",0

         CNOP 0,4

TXTERRC  DC.L 0
         DC.L 0
```

```
SEARCH    MOVE.L D1,D5          Copy word count.
SEARGET   LEA SEART1(PC),A0
          BSR WRITESTR
          BSR BLOCKPG           Search parameter in D1.L
          BSR NEWLINE
          BSR NEWLINE
          MOVE.L D1,D6          Save data in D6.L
          ANDI.L #$FF,D1        Clear data to byte size.
          CMP.L D6,D1           Check that search data is one byte only.
          BEQ SEARCHST          If one byte start search.
          LEA SEART2(PC),A0     Point to error message.
          BSR WRITESTR          Display it.
          BRA SEARGET           Re-try.
SEARCHST  MOVE.L D2,D1          D1 = start address.
SEARCHL1  LEA SEART3(PC),A0
          BSR WRITESTR
          MOVE.L D1,D4          Copy address for displaying.
          BSR HEXLONG           Display it.
          BSR PEEK              Get data from that address.
          MOVE.L 16(A3),D7      Set D7 for DBcc byte count.
SEARCHK   ROL.L #8,D4           Rotate data.
          CMP.B D6,D4           Check byte in data for match.
          DBEQ D7,SEARCHK       Continue search until all check or match.
          BNE SEARFAIL          No match found in this word.
          LEA SEART4(PC),A0
          BSR WRITESTR          Over-print 'Searching' with 'Found at '
SEARFAIL  ADD.L 12(A3),D1       Advance address.
          SUBQ.L #1,D5          Reduce byte count.
          BNE SEARCHL1          Repeat until cycle complete.
          LEA EXMT3(PC),A0      Point to 'completed' text.
          BSR WRITESTR          Display text.
          BSR WAIT              Wait for operator.
          BRA MENU              Return to main menu.

SEART1    DC.B 13,10,"Search for data byte of : ",0
SEART2    DC.B "Data NOT byte sized."
          DC.B 13,10,10,"Re-do!",13,10,0
SEART3    DC.B 13,"Searching address ",0
SEART4    DC.B 13,"Found at ",10,0

          CNOP 0,4

ABORTOP   LEA ABORTT(PC),A0
          BSR WRITESTR
          BSR WAIT
          MOVEA.L ABORTSP(PC),A7  Restore stack pointer.
          BRA MENU
```

```
ABORTSP  DC.L 0
ABORTT   DC.B 13,10,10,"Operation aborted.",13,10,0

         CNOP 0,4

*        ASCII monitor of active link.

ASCIIMON SWAP D3               Change mask to receive mask.
ASCIIGET BSR INKEY0            Check for abort. Don't wait for response.
         ORI.B #32,D0          Force to lowercase.
         CMPI.B #"x",D0        Check for valid abort character.
         BEQ ABORTOP           Leave if requested.
         BTST D3,1(A6)         Check link for answer.
         BEQ ASCIIGET          Wait until it is received.
         MOVE.B 1(A2),D0       Read answer.
         CMPI.B #10,D0
         BRA ASCIIOK
         CMPI.B #13,D0
         BRA ASCIIOK
         CMPI.B #32,D0
         BLT ASCIIERR
         CMPI.B #127,D0
         BLE ASCIIOK
ASCIIERR MOVE.B #"_",D0
ASCIIOK  BSR WRITECHR          Display character.
         BRA ASCIIGET

*        Define transputer type for each link.

SETTYPE  MOVEM.L D0/D4-D6/A0/A1/A3,-(SP)  Save registers.
SETTLOP  LEA STYPEM1(PC),A0    Point to menu 1.
         BSR WRITESTR          Display it.
         LEA LINK0(PC),A3      Point to link 0 data table.
         BSR LINKINFO          Display it.
         LEA STYPEM2(PC),A0    Point to menu 2.
         BSR WRITESTR          Display it.
         LEA LINK1(PC),A3      Point to link 1 data table.
         BSR LINKINFO          Display it.
         LEA STYPEM3(PC),A0    Point to prompt.
         BSR WRITESTR
SETTLOP2 BSR INKEY             Get response.
         MOVE.B D0,D4          Copy result.
         CMPI.B #"0",D4        Test for set link 0.
         BEQ SETTYPEN          Find type and update link 0.
         CMPI.B #"1",D4        Test for set link 1.
         BEQ SETTYPEN          Find type and update link 1.
         CMPI.B #"2",D4        Check for exit to main menu.
```

```
          BNE SETTLOP2          Incorrect response so get another.
          MOVEM.L (SP)+,D0/D4-D6/A0/A1/A3  Restore registers.
          BRA MENU              Return to main menu.
TYPEA     LEA STYPEA(PC),A0     Point to error text.
          BSR WRITESTR          Display it.
SETTYPEN  LEA STYPEN(PC),A0     Point to prompt.
          BSR WRITESTR          Display it.
          BSR WRITECHR          Display link number.
          LEA STYPEEG(PC),A0    Point to examples
          BSR WRITESTR          Display them.
          LEA STYPER(PC),A0     Point to response area.
          BSR READSTR           Get response.
          BSR NEWLINE
          MOVE.W (A0)+,D5       Get size of response.
          CMPI.B #4,D5          Check length.
          BNE TYPEA             Invalid response.
          MOVE.L (A0),D5        Read in full string.
          LEA TYPET(PC),A0      Point to type table.
          MOVE.W (A0)+,D6       Load number of types to check.
          BRA TYPECKS           Skip over next command.
TYPECKT   LEA 18(A0),A0         Skip data in table.
TYPECKS   CMP.L (A0),D5         Check type header.
          DBEQ D6,TYPECKT       Repeat checks.
          BNE TYPEA             Invalid type so go to error response.
          LEA LINK0(PC),A1      Point to Link type data table for link 0.
          CMPI.B #"1",D4        Check for link 1 update.
          BNE TYPEUP            Go to update.
          LEA LINK1(PC),A1      Point to link 1 data table.
TYPEUP    MOVE.L (A0)+,(A1)+    Copy actual transputer name.
          MOVE.L (A0)+,(A1)+    Copy over MEMSTART.
          MOVE.L (A0)+,(A1)+    Copy over LASTINT.
          MOVE.L (A0)+,(A1)+    Copy over number of bytes in data word.
          MOVE.W (A0)+,(A1)+    Copy byte count as for DBcc format.
          BRA SETTLOP           Got back to sub-menu.

STYPEM1   DC.B 13,10
          DC.B "Transputer Link Definition:",13,10
          DC.B 10
          DC.B "Options:",13,10
          DC.B 10
          DC.B "0: Set transputer type on link 0.",13,10,0
STYPEM2   DC.B "1: Set transputer type on link 1.",13,10,0
STYPEM3   DC.B "2: Return to main menu.",13,10,10
          DC.B "Which one? (0-2): ",0
STYPEA    DC.B 13,10,"Invalid transputer type or "
          DC.B "type not supported!",13,10
          DC.B 10
```

```
         DC.B "Please try again.",13,10
         DC.B 10,0
STYPEN   DC.B 13,10,"Transputer attached to link ",0
STYPEEG  DC.B " is ? (eg T414 or T212 etc ): ",0

         CNOP 0,4

STYPER   DC.B 10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

         CNOP 0,4

TYPET    DC.W 3
         DC.B "T212"
         DC.L $8024
         DC.L $87FF
         DC.L 2
         DC.W 1
         DC.B "T414"
         DC.L $80000048
         DC.L $800007FF
         DC.L 4
         DC.W 3
         DC.B "T800"
         DC.L $80000070
         DC.L $80000FFF
         DC.L 4
         DC.W 3
         DC.B "M212"
         DC.L $8024
         DC.L $87FF
         DC.L 2
         DC.W 1

* Link 0 data table preset for T414

LINK0    DC.B "T414"
         DC.L $80000048
         DC.L $800007FF
         DC.L 4
         DC.W 3
         DC.L $FA0000
         DC.L $00000001

* Link 1 data table preset for T414

LINK1    DC.B "T414"
         DC.L $80000048
         DC.L $800007FF
         DC.L 4
         DC.W 3
         DC.L $FA2000
         DC.L $00020003

         CNOP 0,4

LINKINFO MOVEM.L D4/A3,-(SP)  Save registers.
         LEA LNKTYPE(PC),A0
         MOVE.L (A3)+,(A0)    Copy transputer type into text.
         LEA LNKINF1(PC),A0   Point to full text.
         BSR WRITESTR         Display it.
         MOVE.L (A3)+,D4      Get MEMSTART.
         BSR HEXLONG          Display it.
         LEA LNKINF2(PC),A0   Point to next line of text.
         BSR WRITESTR         Display it.
         MOVE.L (A3)+,D4      Get LASTINT
         BSR HEXLONG          Display it.
         LEA LNKINF3(PC),A0
         BSR WRITESTR
         MOVE.L (A3)+,D4
         BSR HEXBYTE
         LEA LNKINF4(PC),A0
         BSR WRITESTR
         MOVEM.L (SP)+,D4/A3
         RTS

LNKINF1  DC.B 13,10,"      Transputer connected to the link is a "
LNKTYPE  DC.B "T414"
         DC.B 13,10,10,"        MEMSTART = ",0
LNKINF2  DC.B 13,10,"        LASTINT  = ",0
LNKINF3  DC.B 13,10,10,"      There are ",0
LNKINF4  DC.B " bytes in a word for this transputer.",13,10,10,0

         CNOP 0,4
```

G.C Main Atari Program Support Routines.

The following routines as needed to change the 68000 processor in the Atari into its supervisory state and back to user mode. In supervisor mode the programs may access protected areas of the Atari's memory which would normally cause a **BUS ERROR** and the suspension of the program.

```
* These routines are based around a GEMDOS call.

* GEMDOS 32 will change the system mode based on the parameters found
* on the stack. If the stack contains zero, then the system is put
* into supervisor mode and the supervisor stack pointer is returned in
* D0. If, however, a value other than zero is found on the stack then
* the system is placed into user mode and the system stack pointer
* assumes the value that was on the stack.


* Set the system into supervisor mode.

            CNOP 0,4                    Align program.

SUPERMODE   MOVEM.L A0-A6/D0-D7,-(SP)   Save registers.
            LEA SUPERSP(PC),A0          Point to SP copy.
            TST.L (A0)                  Test our copy of super SP.
            BNE SUPERET                 If NE the we are in SUPER mode.
            CLR.L -(SP)                 Stack a zero SP for the routine.
            MOVE.W #32,-(SP)            Set for change of mode.
            TRAP #1                     GEMDOS call.
            LEA SUPERSP(PC),A0          Point to SP copy.
            MOVE.L D0,(A0)              Save system stack.
            ADDQ.L #6,SP                Remove rubbish from stack.
SUPERET     MOVEM.L (SP)+,A0-A6/D0-D7   Restore registers.
            RTS

* Set the system into user mode.

USERMODE    MOVEM.L A0-A6/D0-D7,-(SP)   Save registers.
            LEA SUPERSP(PC),A0          Point to SP copy.
            TST.L (A0)                  Test our copy of super SP.
            BEQ USERRET                 If EQ the we are in USER mode.
            MOVE.L (A0),-(SP)           Put our copy of SP on stack.
            MOVE.W #32,-(SP)            Set for change of mode.
            TRAP #1                     GEMDOS call.
            LEA SUPERSP(PC),A0          Point to SP copy.
            CLR.L (A0)                  Clear copy to show as USER mode.
            ADDQ.L #6,SP                Remove rubbish from stack.
USERRET     MOVEM.L (SP)+,A0-A6/D0-D7   Restore registers.
            RTS

SUPERSP     DC.L 0                      Copy of super SP.

            CNOP 0,4                    Align program.
```

The next set of routines provide the software with general input/output support. These were formed into a library for including at compile time.

```
* Write out a null terminated string.

* A0 points to the string.

WRITESTR MOVEM.L A0-A6/D0-D7,-(SP)  Save registers.
         MOVE.L A0,-(SP)            Stack message address.
         MOVE.W #9,-(SP)            Set for write string.
         TRAP #1                    Write the string.
         ADDQ #6,SP                 Remove rubbish.
         MOVEM.L (SP)+,A0-A6/D0-D7  Restore registers.
         RTS

* Throw a newline.

NEWLINE  MOVEM.L A0-A6/D0-D7,-(SP)  Save registers.
         MOVE.W #13,-(SP)           Stack CR.
         MOVE.W #2,-(SP)            Set for write character.
         TRAP #1                    Write character.
         ADDQ #4,SP                 Remove rubbish.
         MOVE.W #10,-(SP)           Stack LF.
         MOVE.W #2,-(SP)            Set for write character.
         TRAP #1                    Write character.
         ADDQ #4,SP                 Remove rubbish.
         MOVEM.L (SP)+,A0-A6/D0-D7  Restore registers.
         RTS

* Read a string.

* A0 points to the workspace start which must be set up before calling
* this routine.

* Workspace format:
```

```
* MAX LENGTH, true length,......TEXT......,0

READSTR  MOVEM.L A0-A6/D0-D7,-(SP)  Save registers.
         MOVE.L A0,-(SP)            Store workspace area.
         MOVE.W #10,-(SP)           Set for read string.
         TRAP #1                    Cause read.
         ADDQ #6,SP                 Remove rubbish.
         MOVEM.L (SP)+,A0-A6/D0-D7  Restore registers.
         RTS

* This writes the character in D0 to the screen.

WRITECHR MOVEM.L A0-A6/D0-D7,-(SP)  Save registers.
         ANDI.W #$00FF,D0           Clear top byte.
         MOVE.W D0,-(SP)            Stack character.
         MOVE.W #2,-(SP)            Set for write character.
         TRAP #1                    Write character.
         ADDQ #4,SP                 Remove rubbish.
         MOVEM.L (SP)+,A0-A6/D0-D7  Restore registers.
         RTS

* This routine prints a prompt and waits for return to be pressed.

WAIT     MOVEM.L A0-A6/D0-D7,-(SP) Save registers.
         LEA PRNTRET(PC),A0        Point to message.
         BSR WRITESTR              Write the message.
         MOVE.W #8,-(SP)           Set for character grab.
WAITRET  TRAP #1                   Grab a character from the keyboard.
         CMPI.B #13,D0             Check to see if it's a CR.
         BNE WAITRET               Cycle until ret found.
         ADDQ #2,SP                Remove rubbish.
         BSR NEWLINE               Throw a new line.
         MOVEM.L (SP)+,A0-A6/D0-D7 Restore registers.
         RTS

* The next two routines get a character from the keyboard.

* INKEY0 will see if there is a character ready at the keyboard, if
* there is then it will read this character otherwise it assumes that
* the character is null.

* The value of the character is returned in D0.

* INKEY this routine waits for a key to be pressed and returns its
* value in D0.
* In both cases D0 holds the character read and the character is not
* echoed on the screen.
```

```
INKEY0   MOVE.W #$0B,-(SP)          Set for keyboard status read.
         TRAP #1                    Get status.
         ADDQ #2,SP                 Remove rubbish.
         TST.B D0                   Test status.
         BNE INKEY                  Get character if ready.
         RTS                        Return if not.
INKEY    MOVE.W #8,-(SP)            Set for character grab.
         TRAP #1                    Grab a character from the keyboard.
         ADDQ #2,SP                 Remove rubbish.
         RTS

* This routine causes an exit to DESKTOP.

EXIT     CLR.W -(SP)                Stack exit command.
         TRAP #1                    Cause the exit to DESKTOP.
         RTS                        Not needed.

PRNTRET  DC.B 13,10,"-----PRESS RETURN----",0

         CNOP 0,4                   Align to next long word boundary.
```