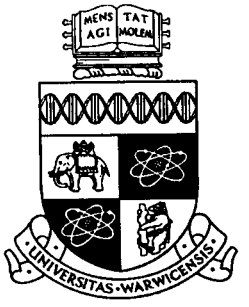


Super-precision programmable current source for coil/magnet actuators

by

David C. Dyer

BSc(Hons) CPhys MinstP CEng MIEE MBCS

Submitted for the award of PhD

at

The University of Warwick

Department of Engineering

May 1995

Dedicated to

My wife
for her kindness, patience and encouragement

My children
who have missed me for several years

My parents
for the education that underpins all that follows

Exhortation

Awake! - brave scholar and resume the fight,

Work all day and then most of each night.

With rigour and method decide what's right,

And then with luck, you'll see a great light

Acknowledgement

I should like to pay special thanks to Dr. D. G. Chetwynd, who provided excellent advice on how a thesis should be written and presented; and also for achieving a happy balance between colleague and mentor.

Summary

This thesis describes the design and development of a super-precision programmable current source that can deliver up to about ± 100 mA to an inductive load. The load is intended typically to be a coil in a coil/magnet actuator that provides a force which is proportional to the current, and results in a linear and well defined movement of an elastic flexure mechanism. The particularly demanding application of long-range x-ray interferometry required two tracking current sources that offered a resolution to better than 1 part in 500,000 and this could not be satisfied by commercially available instruments. Consequently it was necessary to design, construct and test two identical supplies (or *drives*); a non-trivial and very demanding task since exceptionally slow drives scans needed to be accommodated. Temporal stability is therefore critical. Although the operational bandwidth can be kept small, noise up to over 1 kHz must be rigorously suppressed to avoid exciting resonances in the system being driven. Commercial 20-bit digital-to-analogue converters could not be utilised to provide a resolution of 1 part per million, because they are invariably designed for audio applications and have unacceptable drifts with temperature and time. The integral non-linearity had to be less than $\pm 0.0007\%$ (15 ppm) and the design actually achieves ± 0.5 ppm by using an embedded precision analogue-to-digital converter to form a servo-loop within each drive. A desk-top computer (PC) accepts setpoints via a serial communications channel, and simultaneously controls the servo-loops for two drives by the exchange of simple messages via optically isolated links. The major components within each drive are, an embedded 8-bit micro-controller, two DAC's providing coarse and fine voltage settings, a precision voltage-to-current converter, a precision ADC and an ADC which monitors critical nodes, all of which are discussed in considerable detail together with the algorithms and software in the PC and microcontroller. Circuit simulations were an important part of preliminary studies and are presented along with measures of actual performance. It is shown that the drives achieve not only a resolution of 1 ppm but that all other operational parameters are of a similar order. A number of proposals are made for alternative methods which represent the foundations for future work.

Contents

List of Figures

Abbreviations

ADC	Analogue to digital converter
BCD	Binary coded decimal
cps	counts per second
DAC	Digital to analogue converter
dc	Direct current
DNL	Differential non-linearity
DTI	Department of trade and industry
EEPROM	Electrically erasable programmable read only memory
FSO	Full scale output
IEEE	Institution of Electrical and Electronic engineers
INL	Integral nonlinearity
LSB	Least significant bit
MSB	Most significant bit
NPL	National physical laboratory
NS	National semiconductor corporation
PCB	Printed circuit board
ppm	parts per million
PTB	Physikalisch-Technische Bundesanstalt
PWM	Pulse width modulation
PZT	Piezo-electric transducer
RAM	Random access memory
RF	Radio frequency
rms	Root mean square
THD	Total harmonic distortion

Chapter 1 Application of precision coil/magnet actuators

"The first experimental investigation of the interaction between coils carrying electric currents was performed by Ampère during the years 1820-5, and the work was continued by Oersted, Biot and Savart. ... Both a magnet and a current-carrying coil are said to produce a magnetic field described by a flux density **B**, *which exerts forces on other coils or magnets". [1.1]*

It seems unlikely that these scientists could have foreseen the far reaching consequences of the discoveries which underpin the present work. It would probably have been beyond their belief that it is now possible to control current flow so finely that, through the subsequent minute variations in forces between coils and magnets, movements smaller than the atomic spacing of silicon can be made and measured. But such is now the case and the remainder of this thesis is devoted to an explanation of a technique which does just this, and in particular the design, construction and testing of a super-precision programmable current source. Such sources can have many applications but the current work arose from the need for a technology breakthrough to control adequately low-speed nanometric movements using electromagnetic actuators and the thesis addresses issues in this context.

1.1 Coil/magnet geometries and actuators

In regard to the relationship between forces due to current carrying conductors the simplest 'mathematical object' to be analyzed was the force between two infinitely long parallel conductors. Of course this arrangement cannot be realised an practice and further analysis was performed on closed loops, ideal coils and solenoids in order to determine the variation of magnetic field strength, **H**, with position. Probably the most notable early achievement was by Helmholtz (1821-1894) who showed that if two identical circular coils of radius a, are placed parallel and co-axial a distance r apart, the field near their geometric centre is uniform over a significant region when $r=a$. This is because at the centre dH/dr, d^2H/dr^2 and d^3H/dr^3 are all zero. (actually Helmholtz made a more notable contribution to science in 1847 when he put forward the 'Law of conservation of energy'. But it was rejected for publication by the editor of Annalen der Physik! [1.2])

For real coils, which use wire of finite diameter, with subsequent limits on packing density, expressions for **H** in terms of some orthogonal co-ordinate system (x,y,z)

may involve integrals which have no analytic solution. Alternatively expressions may be so complicated that it is difficult to plot graphs of H(x,y,z) by hand. However with the advent of digital computers numerical modelling became easier and new expressions were developed.[1.3]

If a bar magnet is positioned near or within a coil, it experiences a force which is proportional to the current in the coil. If it is fixed to an elastic support, it will move until the force due to the current (driving force) equals the elastic restoring force. A simple arrangement is depicted in Figure 1.1.

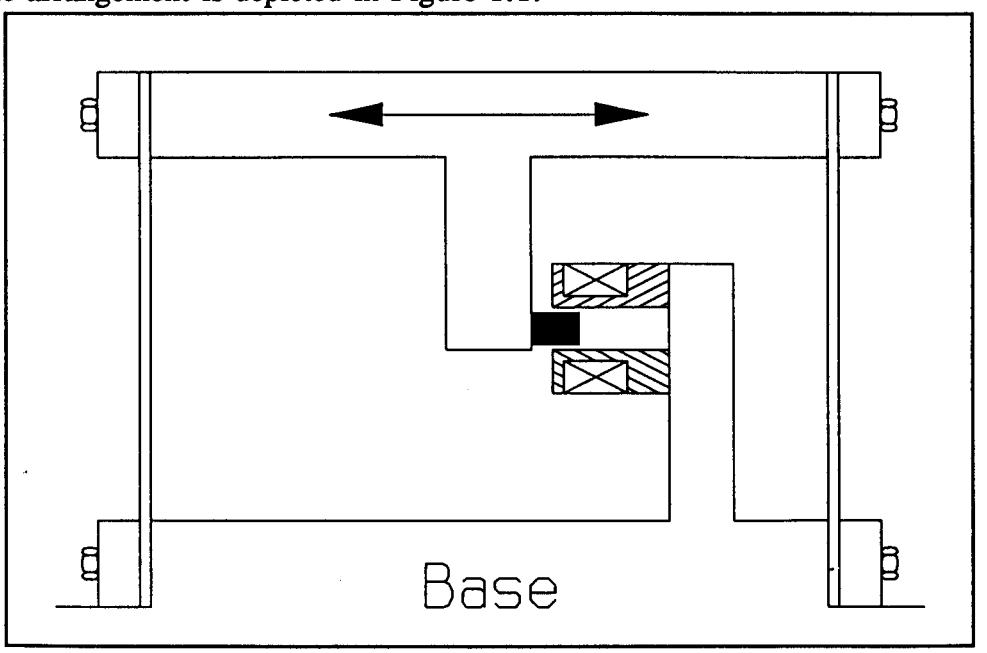


Figure 1.1 - Flexure mechanism with coil/magnet actuator

Even if the restoring force varies linearly with displacement, the net movement is usually not linear because the driving force varies with absolute position. However work done by Smith and Chetwynd [1.4] shows there to be a optimised design of coil such that the driving force is essentially independent of position over a certain region. The significance of this is not in the design of large coils with constant fields over millimetres, but rather small coils which provide a useful range of several micrometres. Such coil/magnet combination can then be used in a wide range of applications under the general heading of *nanotechnology*. For example, in the design of high precision translation mechanisms [1.5] and the control of instruments for the measurement of surface profile [1.6]. Of course coil/magnet combinations have been used for moving-coil motors; e.g. in loudspeakers, and for the head positioning in some disc drives; as well as force balancing techniques for weighing [1.7]. In trying to scale moving-coil devices to provide small movements the problems associated with mechanical coupling become increasingly severe. Consequently moving-coil systems are considered as macroscopic applications and not discussed again. Instead the

emphasis is on systems using moving-magnets to achieve microscopic displacements.

In moving to ever smaller currents, forces and therefore displacements, it becomes increasingly difficult to measure the actual displacement and thereby characterise the behaviour of precision actuators. It should be noted that there is a difference between the detection of small movements and the measurement of absolute position. For example capacitive gauges [1.8] can detect picometre movements and, by a servo-loop, maintain the relative position of two components, but are not so useful for long range absolute measurements. Optical laser interferometers can be used to measure displacements down to about 10 nm relatively easily [1.9], but this is still large compared with the potential resolution afforded by magnet/coil actuators which may better 10 pm [1.10]. Consequently an alternative method of measurement is needed which is reliable, accurate and above all traceable to the international definition of the metre.

1.2 X-ray interferometry

To provide a traceable measurement an internationally recognised scale is needed, and this may be provided by the lattice parameter for silicon, which has been referenced to the absolute length scale at PTB in Germany [1.11] and is characterised to a precision of better than 1 part in 10^7. In 1965 Bonse and Hart [1.12] showed how an X-ray interferometer could be built, and subsequently Hart suggested that it be used to create an 'Ångstrom ruler' [1.13]. However, according to Chetwynd [1.14] it was not until 1983 that the application as a ruler was seriously pursued with the demonstration of its use as a microdisplacement calibrator by Chetwynd *et al* [1.15].

The operation of an x-ray interferometer is theoretically complex (see cited references) and is not directly of concern to the work here. Suffice it to say that scanning one thin 'blade' of a single crystal of silicon past other parallel blades causes a modulation in the intensity of a transmitted x-ray beam that varies sinusoidally with the pitch of the lattice. In effect, the system behaves as an incremental grating of pitch equal to the lattice parameter.

Other work in this area includes that by Becker and Seyfried in 1989 [1.16] who used Piezoelectric (PZT) drives, and Bowen *et al* in 1990 [1.17] who used a silicon monolith and coil/magnet actuator to produce controlled displacements of less than 0.01 nm. A significant advantage of using a coil/magnet actuator over PZT stacks is

that small unwanted movements of the coil caused by vibrations in the ground, do not result in such vibrations in the magnet attached to the monolith, because the force in almost independent of position [1.4].

Bowen, Chetwynd and Smith were all established members of the Centre for Microengineering and Metrology (renamed in 1994 as the Centre for Nanotechnology and Microengineering) at the University of Warwick, when the Department of Trade and Industry (DTI) invited tenders for the design and manufacture of a "Traceable secondary standard displacement facility with sub-nanometre resolution - DTI reference MPU 8/0.13". Their collective experience lead to a proposal for an instrument which used a silicon monolithic x-ray interferometer, which in principle could meet the required resolution of 0.02 nm over a range of 10 μm. That is to say the mechanical properties of a suitable monolith were sufficiently well understood that together with a coil/magnet actuator the required movements could be obtained if the current in the coil could be controlled sufficiently well. In fact two programmable current sources were needed because, in order to obtain fringes over the desired range, there must be active compensation for twisting caused by a variety of secondary and parasitic effects which may be ignored over shorter ranges. At this time the author became involved with the project and defended the view that the design of a programmable current source with a resolution of 1 part in 500,000 was feasible, although a non-trivial and potentially very difficult task. The intended mode of operation of the instrument meant that not only was the resolution to approach 1 part per million (ppm), but also the noise, integral non-linearity and stability over long periods needed to be of the same order.

Figure 1.2 shows the general construction of a silicon monolith for use as an x-ray interferometer, while Figure 1.3 gives the arrangement of the drive proposed to the DTI. It has a cross bar and two magnets and coils in order to provide a trim torque to compensate for parasitic motion. Also, the use of two coils results in a lower total power dissipation for a given force.

Although much of the discussion for the requirements of a precision programmable current source which follow refer to x-ray interferometry, it would be wrong to conclude that this it the only application. Many other applications come to mind including atomic force microscopy, x and y translation for surface measurements, magnetic lensing in electron microscopes and movement of mirrors in optical metrology. The requirements for long-range x-ray interferometry are particularly demanding and are regarded as a sensible vehicle for developments and discussions.

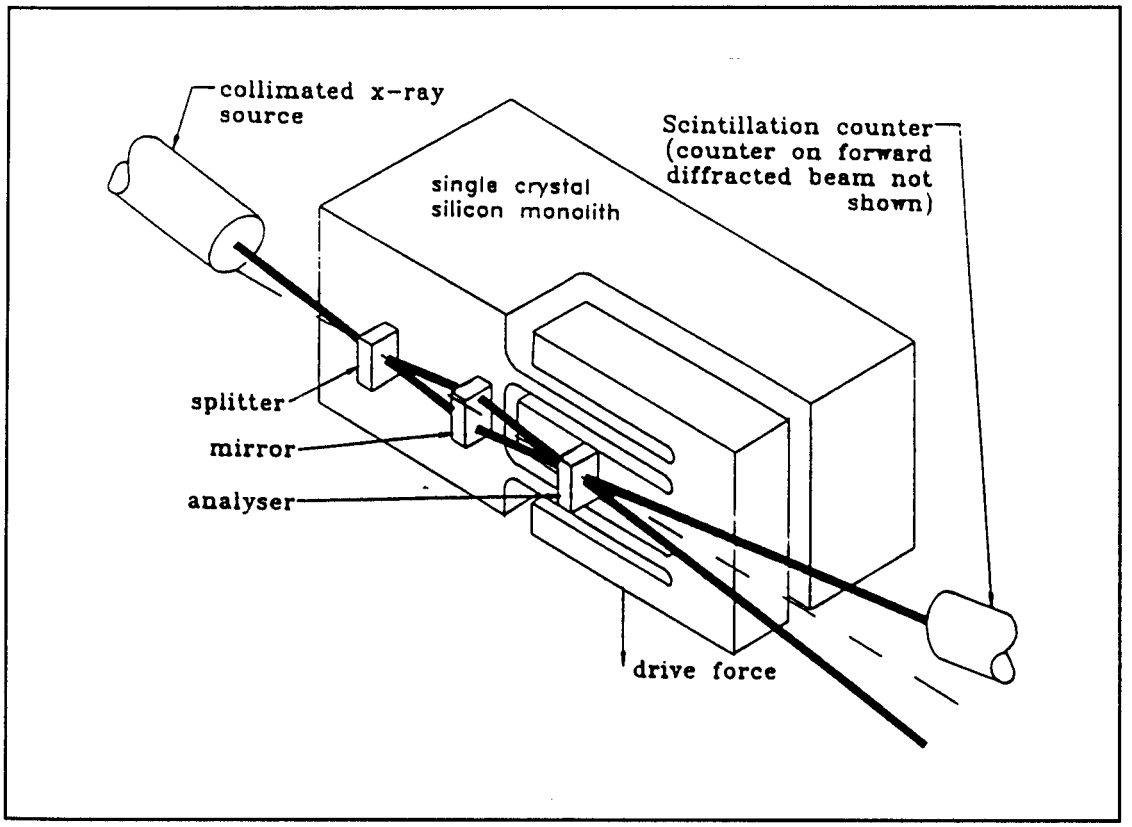


Figure 1.2 - Silicon monolith [1.14]

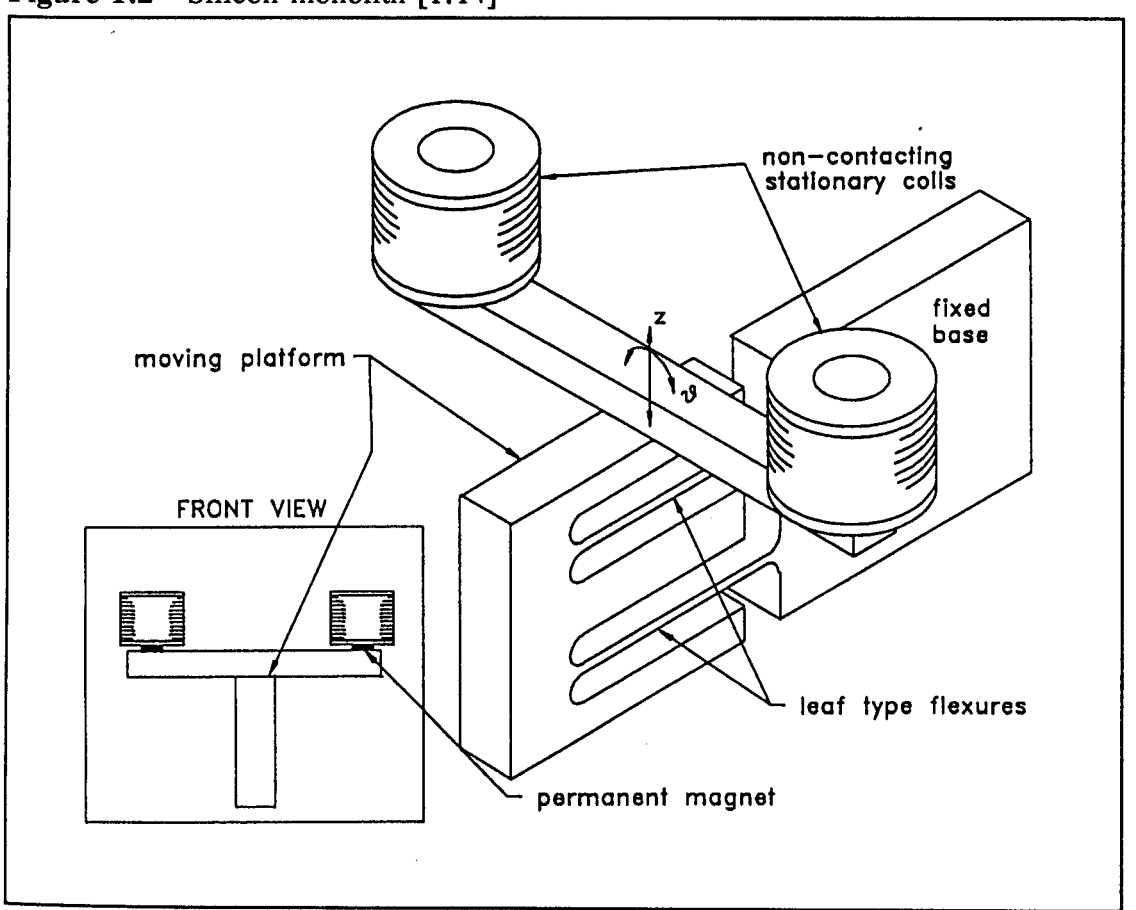


Figure 1.3 - Silicon monolith with two magnets [1.18]

1.3 Coil arrangements for translation and twist

For a given monolith there may be an axis along which a force can be applied that will result in translation without any rotation. However imperfections in the machining of the leaf-springs and the inability to define the exact position of the magnet make it such that a torque, and therefore rotation or twist, is expected. For long-range interferometers the problems associated with unwanted rotation cannot be ignored as they degrade the contrast of the fringes and active compensation schemes must be used to ensure optimum performance.

In order to provide independent control over linear force (for translation) as well as torque (for twist compensation), at least two current sources are needed, and Figure 1.4 shows three topologically different, yet symmetrical, coil arrangements which provide this.

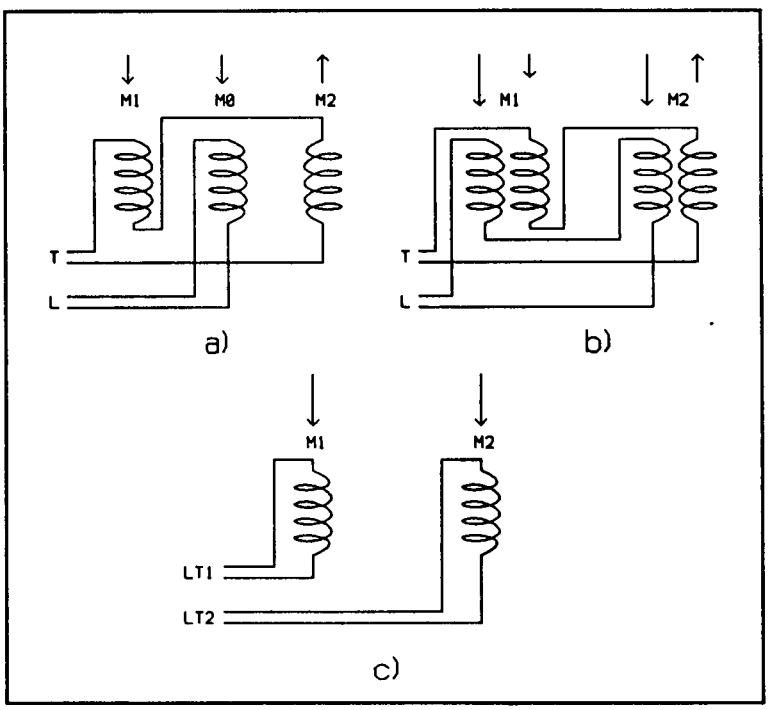


Figure 1.4 - Coil Arrangements

The arrangement shown in Figure 1.4a) is the most obvious extension from previous work. It shows a main linear drive, L, associated with the centre coil and magnet M0, and a torque drive, T, with the outer two magnets M1 and M2. The outer coils are wound in opposite directions so they can induce a torque with no net linear force. Unfortunately, magnets M1 and M2 will not be identical, nor for that matter will be the geometry of the coils or the distances from the main axis of the platform, and this will result in a linear displacement whenever a simple torque is required.

A refinement is shown in Figure 1.4b) which uses only two magnets with over-wound coils. Once again the intention is to be able to trim any twist of the platform by careful control of a torque. There are numerous sources of error which can result in either an undesirable torque or linear force. However the arrangement is better than a) because the force required for linear movement of the platform is now split between two coils. This means that the current in each coil need be only half that using one coil, and the power per coil will be reduced to one quarter. In turn this means the effects of local heating are reduced.

In both a) and b) the current for the linear drive, L, will be orders of magnitude greater than the torque drive, T. This difference is easy to accommodate by using similar drive circuits acting on coils with a differing number of turns.

The arrangement in Figure 1.4c) is easier to build and understand than b) but necessitates two drive circuits with exactly the same performance if both coils are to produce the same forces for the same control codes. In fact this is the ideal situation and differences between M1 and M2, as well asymmetries in the geometry of the coils will require the offsets and gains of the two drives to be different, but maintained at a fixed ratio. The names of the drive signals are changed from L and T, to LT1 and LT2, to reflect the change in use; suitable increases in common to them both will cause a linear movement, while differential changes will cause a rotation.

Two coils may be used to achieve control of translation and twist in one plane but it may be advantageous for some mechanisms to allow controlled movements in several planes. The application of forces at a number of positions may cause direct and desirable movement, or may compensate for unwanted parasitic effects. The designs for flexure mechanisms, possibly using multiple webbed hinges, are beyond the present work, but it is possible that design rules may be reconsidered if programmable current sources are sufficiently cheap and accurate, that numerous coil/magnet actuators may be employed.

1.4 Requirements for a programmable current source

The requirements for the current source depend upon the exact nature of the application but the needs of the x-ray interferometer are considered so severe that they place an upper limit on the performance a source must achieve. Consequently

if a source may be used for the interferometer it will necessarily be capable of numerous other applications, although perhaps limited by economic considerations.

To satisfy the requirements of the DTI project the resolution must be at least 1 part in 500,000, and it is practical to try and better this by aiming for 1 part in 10^6, or 1 ppm. It is reasonable to assume that, whatever means is used to determine the current, it will rely on digital components, which form a binary scale. This binary scale needs 20 bits since 1 ppm most closely maps to 1 bit in 2^{20}, (where $2^{20}=1048576$). Furthermore, given a range of 10 μm and that the <111> lattice parameter for silicon is about 0.313 nm, there will be $(10 \times 10^{-6})/(0.313 \times 10^{-9}) =$ 31949 fringes.

It is now important to mention that during operation of the x-ray interferometer as a calibration facility, other position sensors will measure the displacement of the monolith and, to test their accuracy in different absolute positions, it is convenient to move several micrometres in a single 'jump'. Without this facility it is necessary to track the movement across every (intermediate) fringe, and to do so requires photon counting to take place at two points in each fringe. The x-ray energy is kept low to avoid unwanted heating; consequently the count rates are low, and it takes many (>10) seconds to determine the position within a fringe. For example, a 1 μm movement corresponds with about 3000 fringes, which implies a scan takes 30000 seconds (say), or over 8 hours. Jumping over fringes relies on being confident that the actuator, and indirectly the current source, has indeed placed the monolith to within half a fringe width of the expected fringe. So long as the correct fringe has been found, the exact position ± 10 pm may be found by making small additional movements and monitoring of changes in photon count rates. Alternatively full determination of the position within a fringe may be obtained without further movement if a phase-stepping technique is incorporated [1.19]. If a binary scale is perfectly matched to the position of the fringes, then N binary counts will correspond to exactly N fringes. Unfortunately, whatever digital-to-analogue conversion process is used will be inaccurate and even if it offers monotonic performance and resolution to 1 ppm a lower bound can be calculated for its integral non-linearity if 'jumping' is to be achieved. The permissable uncertainty can be explained with the aid of Figure 1.5. Consider a jump from fringe x to fringe x+n, and that, in the first instance, the position is bounded by points p and q ($\pm$ ½ fringe). A small increase moves these bounds to p' and q', but the interpretation about absolute position is ambiguous because a fringe boundary has been crossed. Suppose, instead, that the position is bounded by the points a and b ($\pm$ ¼ fringe). Then a small increase bounds

the position by a' and b' which is within the same fringe and allows determination of the absolute position by interpolation. In fact the constraint of ± ¼ fringe is unnecessarily tight, but on a binary scale it is the next best range when ±½ fringe is rejected.

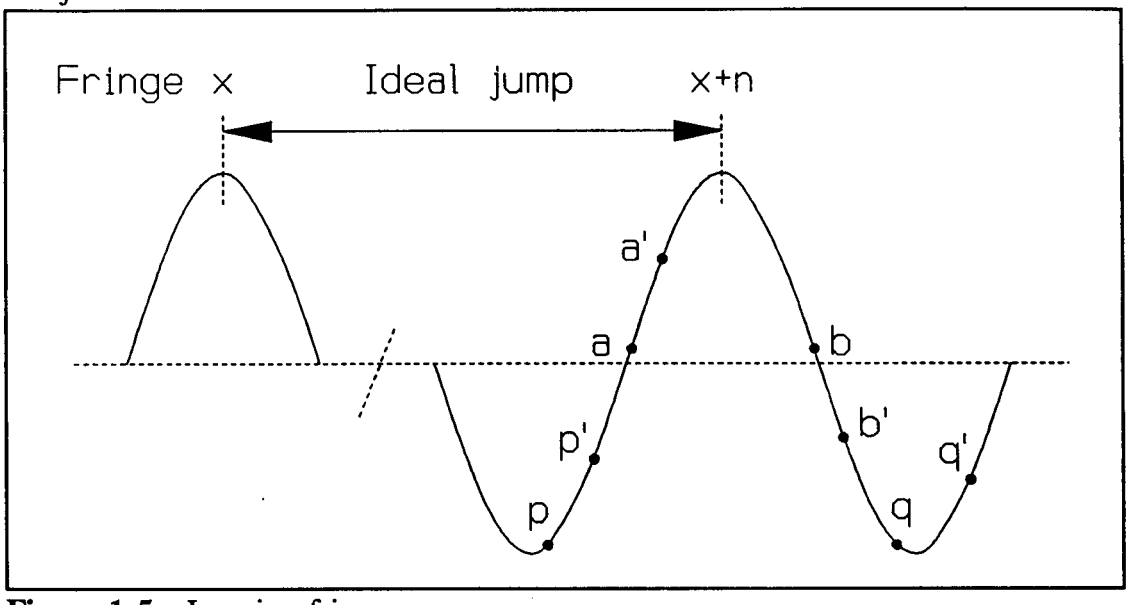


Figure 1.5 - Jumping fringes

Since the range of 10 μm contains 31949 fringes, which maps closely to a 15-bit scale with 32768 intervals, the corresponding required integral non-linearity (INL) is ±¼ of the least significant bit (LSB). Expressed as a percentage of full scale this is ±0.0007%, and if it were not for the need for 20-bit resolution a practical realisation of this bound can be achieved using a 16-bit DAC with an INL of ±½ LSB, such as part DAC729KH by Burr Brown.

The lower bound of ±0.0007% of full scale for INL is actually optimistic since it assumes no other sources of error exist. In general any DAC will experience a drift with time and temperature of both the span and the zero point. Although this may not cause problems for movements throughout fringes, it could cause a fringe to be missed when 'jumping' over many fringes. Chetwynd *et al* in [1.14] propose that running calibrations may be performed which help to track slow changes in the performance of the current source(s), and this may be a practical approach to overcome some of the problems. However it is only made possible because the interferometer provides an accurate ruler, and for other applications it is advantageous to use a programmable current source to provide open-loop positional control. To this end a design target was set to achieve not only a resolution of 1 ppm, but for all parameters to be of this order; including INL, DNL, drift/°C and noise.

Based on previous work with coil/magnet actuators within the Centre for Nanotechnology and Microengineering, an upper bound of ± 100 mA was placed on the current requirements for a single coil. Two identical sources were needed for this application and a thorough survey revealed that no commercial supplies had the necessary performance.

1.5 (Un)suitability of commercial programmable current sources

There are many manufacturers that make programmable voltage and current sources which are designed to deliver power to a load and not intended as reference sources for calibration. In studying the catalogues for companies such as Hewlett Packard, Fluke and Philips, a pattern emerges as to the general characteristics of the supplies. Firstly, there are very many more supplies with voltage outputs than with current outputs, and the maximum output power may be between 50 and 2000 watts. Secondly, the resolution is often stated in such a way as to suggest the use of a N-bit DAC, where N is typically 12 to 14. Figure 1.6 shows the 'best' of what was available in 1992, and only Marconi model 103A comes close to the required performance. Even though it offers a resolution of about 1 ppm the linearity is quoted at only 0.005% (50 ppm) on the +100mA range.

Make	Model	Range	Step	Res. bits	Temp.Co /°C	Linearity	Noise	Comments
Keithley	224	±100mA	50μA	12	12μA	?	100ppm	Non-Inductive load
Fluke	PM2831	±1A		12	?	?	2mA	
HP	6625A	500mA	100μA	14	?	?	0.1mA	
Marconi	103A	±110mA	100nA	20	?	0.005%	?	DC Reference

? = parameter not available

Figure 1.6 - Commercial current sources [1.20 to 1.23]

Although every effort was made to identify all commercial sources at the start of this research work, model 103A by Marconi was not found until after the design ideas based on pulse width modulation discussed in this thesis had already been established independently. It is therefore interesting to note that an entry in a Marconi catalogue [1.20] describes the instrument as " .. a dc voltage and current reference source .. based on a unique pulse width modulation digital/analogue conversion principle..". The resolution is quoted as 100 nA from ± 100 nA to ± 109.9999 mA, which supports the view that internally 6 decades of binary coded decimal (BCD) counters are used to create a digital controlled PWM signal. Detailed study of the specifications shows the linearity for the voltage output to be 0.001%, which is 5

times better than the current output. Perhaps a voltage-to-current converter is employed with inherent non-linearities that limits the performance. Unfortunately no specification is provided for current noise, and with limited linearity the instrument would not have been good enough for the intended application.

The unsuitability of commercial programmable current sources meant that a dedicated supply had to be designed. The rest of the thesis discusses the issues raised in trying create a suitable supply, specific problems and solutions, as well as construction details, performance characteristics and suggestions for future work.

Chapter 2 Design ideas, problems and constraints

The suitability of driving flexure mechanisms by high-compliance coil/magnet force actuators to provide smooth precision movements was discussed in section 1.1. The resolution may be as good as ± 5 pm over 10 μm if the drive current(s) can be controlled sufficiently well, but this is difficult because it represents a range to resolution of 2,000,000:1. Since a typical operating voltage is 2 volts the uncertainty must be less than 1 μV, and at this level electronic noise becomes the limiting factor. Operating the coil at constant voltage was acceptable for the original experiments but since the force depends on current it is much better to operate the coil at constant current, and thereby eliminate changes due to fluctuations in the coil resistance which occur as the temperature changes.

As a design goal, and because of the constraints of a particular application in X-ray interferometry, it was necessary to design a programmable current source with a resolution of 1 part in 1 million (or 1 part in 2^{20}), with correspondingly small drift, noise and non-linearities. This specification is extremely challenging as problems rapidly escalate in trying to go beyond about 17 bits. This chapter discusses these problems and culminates in the proposal for a specific approach and design.

2.1 Generic problems in high-precision electronic instrumentation

To many people the design of electronic circuits is considered as something of an Art [2.1], perhaps because the union of components to make a meaningful whole is seen in much the same way as an artist blending colours and textures to create a pleasing effect. Even in more informed circles this notion is given some credence as the inventiveness needed to design an elegant circuit for a specific function is not commonplace, and the designer is perceived to have a talent akin to that of an artist. However the analogy is deficient and misleading for the electronic design engineer will invariable use models of the real components during the design process, whereas the artist will use real components (pigments) almost straight away. If the object of electronic design is to produce a circuit diagram that has aesthetic appeal then perhaps it is an art form, but unlike a painting the circuit diagram is merely an intermediate stage. What the designer really seeks to create is a physical working circuit, which can only operate as expected if there is a strong correspondence between the meaning attributed to the symbols on the circuit diagram and the behaviour of physical components. Unfortunately the models of analogue circuits become increasingly

simplistic and inaccurate as the desired resolution increases, and very much more effort and understanding is needed to predict the overall performance of each circuit element.

For super-precision current sources where the resolution and linearity are to be about 1 part per million of full scale output (ppm FSO) there is the implied requirement that long term drift, noise and variations with temperature be correspondingly small. This necessitates models of components to hold true for six orders of magnitude, which is often not the case. Models are developed either by theoretical analysis of the physical principles by which a component works, by experimental observations, or more usually by studying data sheets published by a manufacturer. For an individual component it may be possible to characterise its operation to better than 1 ppm under all anticipated operating conditions in the near future, but this does not cater for hitherto undiscovered mechanisms of aging due to, say, background cosmic radiation. It is most unlikely that information such as this is in a data sheet, but that may not be sufficient reason to ignore the possibility.

Very often a circuit diagram shows a line that represents zero volts, and is generally considered as an ideal conductor with no voltage drops along it. Of course at high frequencies this 'ideal' conductor has a significant inductive component which designers of Radio Frequency (RF) circuits are taught to model. At low frequencies and down to direct current (dc) the resistive component of a track on a printed circuit board is very often ignored. Voltage drops of several millivolts are commonplace, and must be considered in precision instruments where a potential difference of even $10\mu V$ may be undesirable. This kind of effect is often static and may be compensated, but it does show an obvious source of difficulty when designing precision circuits.

Four areas which cause dynamic problems are of sufficient importance that they are now mentioned individually. The intention is to highlight specific difficulties so they may be considered appropriately during the description of various designs in later pages.

2.1.1 Drift with temperature

The idea that the behaviour of a component varies with absolute temperature is intuitive and obvious, and in the case of components such as voltage references manufacturers quote a drift in terms of ppm/°C. Careful inspection of data sheets

reveals that the drift figure is obtained by dividing the change in voltage output by the change in temperature, when the temperature changes from 0°C to 70°C. The drift is the mean rate of change and may be much higher at a particular temperature if the response is non linear, see [2.2]. On the other hand, manufacturers quote the maximum drift that may be expected of, say, 10 ppm/°C with the implication that most components will be significantly less. Is a Gaussian distribution assumed with 3σ limits at 0 and 10 ppm/°C, or is some other distribution assumed ? What percentage of components will have a drift of less than 4 ppm/°C ? These are typical and reasonable questions that are not readily answered, but affect greatly the expected performance or a particular instrument built from such components.

Specifications for the drift of input or output voltage with temperature are available for most components, and these often show the effect to be positive or negative. The distribution is usually Normal with a mean of zero and the limits given are derived from $\pm$ 3 standard deviations. A naïve approach is to characterise the worst case drift of an instrument by adding together all of the worst case drifts given in the data sheets for each of the components in use. This approach is flawed because

a) It assumes the contribution to overall drift from each component is additive with the same weighting, and in general this will not be so. The very least that should be done for a particular circuit is to consider a weighted sum.

b) The changes with temperature may be non-linear so that for a particular temperature the drift may be much better/worse that expected. For example, in the Schlumberger digital voltmeter type 7081 an extensive number of components are used to compensate for the 'curved' voltage drift of the reference diode [2.3].

c) There are sources of drift with temperature other than active components. Passive components also have temperature coefficients and in critical sections of the circuits these must be considered. However what is often forgotten are thermoelectric potentials caused by junctions of dissimilar metals, and these may be several μV/°C [2.4]. Some precision amplifiers have drifts which are several orders of magnitude lower than thermoelectric effects so the latter will dominate.

d) Self-heating may cause unexpected local temperature gradients and changes in ambient temperature may not be of such importance.

It is clear that drifts caused by temperature changes need to be considered most carefully.

2.1.2 Drift with time

Drift with time, when the temperature is held constant, is another type of drift that is often ignored because it is so small. This may well be true for many applications but for precision measuring/controlling instruments long term drift may be a significant concern. Manufacturers do not provide such specifications except for components which have to respond to d.c. conditions. For example chopper-stabilised amplifiers, precision voltage references, and high resolution ADCs and DACs. Chopper-stabilised amplifiers, by their very design, have extremely low drifts with time; typically less than 0.01 ppm/month. For the others, drifts of about 10 ppm/1000 hours are often quoted, and since slow 'runs' of 10 hours are possible a change of 0.1 ppm may not be insignificant. Also over a period of several months the drift may be more significant than that caused by changes in operating temperature.

2.1.3 Noise

The effect of random fluctuations of current in electronic components have been studied extensively and characterised [2.5]. Many physical processes are present in real semiconductor devices which result in noise with frequency components that vary throughout the spectrum. For a circuit with many components, noise at its output may usually be considered as a combination of incoherent voltage (or current) sources. In data sheets values for noise in particular frequency bands may be given, but for a specific band it is usual to compute the noise based on the graphs provided showing the noise power spectral density in $\mu V/\sqrt{Hz}$ versus frequency. Idealised models of components [2.6] often assume that flicker (1/f) noise is dominant below some critical frequency f_c, while above that white noise is present. This leads to equations which may not model a real device. For example the low frequency noise spectrum of bipolar operational amplifiers may be close to 1/f but for amplifiers with JFET inputs $1/f^2$ is more likely, and many real amplifiers have $1/f^n$ where n is between 1 and 2. Consequently for precision instruments it is important to model noise using published graphs of spectral density. This may result in numerical integration using data points and/or non-linear regression analysis to obtain equations for definite integration.

Noise is also presented in data sheets in terms of a peak-to-peak (pk-pk) value which is usually in μV over some frequency band. Such figures may be combined in a weighted sum to provide an estimate for the maximum pk-pk value for a compound circuit, but this is not the same as the rms value. If the peaks are short-lived they may have a significant amplitude but negligible energy, and care is needed in the prediction of the overall effect(s).

2.1.4 Interference in mixed digital/analogue circuits

The functionality of many instruments can be increased by the inclusion of a local or embedded microprocessor, which may be programmed to perform a wide range of useful operations. All microprocessors require additional memory and input/output devices to operate, and due to the high speed of execution a complex digital waveform is generated on all interconnecting buses. The waveforms have high frequency components of the order of Megahertz, but may also have low frequency components of the order of 10-100 Hz, because of the cyclic nature of many control programs. When precision analogue components share the same power supply, or even just a common ground line, these frequency components are seen as unwanted disturbances on the analogue output. The method of coupling is not usually by electromagnetic interference but rather by changes in the potential of the shared wire(s) due to transient currents caused by changes in digital outputs. With the advent of optical-couplers, which allow the exchange of digital information without direct connection of electrical circuits, isolated sub-systems can be formed. Only the minimum of data is passed across the coupler(s) to control the analogue circuits, and interference is kept to a minimum. Unfortunately, in the case of an analogue subsystem which uses multiple digital-to-analogue circuits it is necessary either to use numerous couplers in parallel, or to employ digital circuits to decode a serial data stream passed via one coupler. The former choice is expensive while the latter becomes self-defeating as digital logic is needed in the supposedly analogue-only circuit.

An alternative approach now exists because complete low-power microcontrollers are available where the microprocessor, memory and input/output ports are all integrated into one device. Consequently all interconnections are kept 'on-chip' and signals on the output ports may be connected directly to analogue subsystems without causing undue interference. There is still the possibility of unwanted coupling but separate power supplies for digital and analogue subsystem joined at only one star point and

with suitable decoupling capacitors minimises the effect of digital transients. It is also necessary to arrange for digital control signals to be routed on the printed circuit board (PCB) so that they are kept clear of analogue components.

For reasons of safety all mains-operated instruments in metal chassis should have the metalwork connected to the earth of the supply. Many instruments also connect the zero volt rail to the chassis and therefore to the local earth. But if a microcontroller has a serial port for communication with a remote computer which itself is joined to a local earth it is possible to create an earth loop with one connection via a cable between the two ports and another via the mains earth wiring. Unfortunately mains earth points around a building are not always at the same potential and unwanted interference is generated within 'ground' loops. It is not essential to have the zero volt rail(s) of the instrument joined to its chassis, and this removes the ground loop, however this may lead to undesirable potentials between 0 volts and the chassis. Isolation may also be provided using opto-couplers, and in particular with an optically coupled serial link which is now available in integrated form.

2.2 Applicability of published designs for programmable current sources

About a 12 papers from the last 10 years were found using the BIDS [2.7] computer database services, but not all were appropriate because of differences in interpretation of the term 'programmable'. Four different categories of current sources were identified which were programmable

a) at the time of manufacture
b) by selection of suitable resistors or switches
c) by digital codes but for high voltages >200 V
d) by digital codes and for use in instrumentation

Only 3 papers were found in category d) and each of these is discussed in more detail to gain insight into what has been achieved, and to highlight good and bad features. This style of presentation and comparison is a little unusual but appropriate because of the lack of published work in this area. All three papers have the common feature that the DACs used do not have sufficient resolution and there is no real likelihood of using a basic design with a 'better' DAC.

2.2.1 Review of - Programmable current supply for inductive load (1986) [2.8]

The circuit diagram is difficult to follow because all the elements shown as operational amplifiers have unrecognised (Russian) part codes. However it is clear that the voltage output of a 12-bit digital-to-analogue converter feeds a power amplifier using discrete transistors, which achieves voltage-to-current conversion with current gain. A 2-terminal reference resistor driven by an ungrounded inductive load provides voltage feedback. The DAC is controlled by digital latches which are connected directly to the data bus of an Elektronika NTs-80-20/1 microcomputer. The age and tone of the paper suggest this to be a desk-top computer with a microprocessor and auxiliary components rather than a single-chip device. The text also states "The preamplifier has a Goldberg circuit; the high-frequency circuit is regulated by a low-frequency modem channel. This ensures an output current instability of ≤ 0.1 mA over the temperature range 20-50°C" The exact meaning of these words is unclear but it is apparent from the circuit that there are both ac and dc signal paths around the power amplifier.

There are four aspects of the design which limit its application as a high precision current source.

a) The digital logic is not isolated from the analogue components and there may be ground 'loops' and/or other interference effects.
b) The DAC is only 12 bits giving a resolution of 1 part in 4096
c) There is no analysis about noise contribution of the components
d) End effects at the 2-terminal reference resistor will be significant at higher precision.

2.2.2 Review of - Microcomputer-controlled, programmable current source for NMR measurements at very low temperature (1991) [2.9]

This paper describes a circuit which is intended to offer 16-bit resolution. Serious attention is given to the problems encountered in the design of precision electronic instruments. A complete circuit is not presented but the general description is of a microprocessor-based (Z80) instrument with an IEEE-488 interface, and an analogue sub-system that is optically isolated. The needs for low-noise operational amplifiers and a reference resistor with a low thermal coefficient of resistance are mentioned, as is the problem with glitches from the DAC. The usefulness of monitoring the state

of the power amplifier is also explained and provision is made for appropriate components. There are graphs which show the change in current divided by the mean current taken over several hours to be less than 5 x 10^{-5}. This corresponds to about 1 part in 2^{14}, which is large considering a 16 bit DAC is used. There are two areas of weakness which limit the resolution and accuracy of the design.

a) The DAC is stated to be a 16-bit DAC703JP by Burr-Brown; however inspection of the data sheet shows the 'J' part to guarantee monotonicity to only 13 bits. Also there is no guarantee on the maximum temperature coefficient for the internal voltage reference, although ± 10 ppm/°C is given as typical. Selection of the 'C' grade part would have offered monotonicity to 15 bits and a maximum temperature coefficient of ± 15 ppm/°C.

b) The text states "..The reference resistor R_N made from manganin has a nominal value of 0.2 Ω..", and the circuit diagram shows a 2 terminal device. There will be end effects caused by the connection of copper wires or printed circuit board traces to the manganin and a 4 terminal resistor is preferable, where the current sensing connections are not the same as the current forcing connections.

2.2.3 Review of - Development of a programmable current source (1993) [2.10]

The programmable current source described here once again uses a voltage to current converter, where the voltage is derived from a DAC. A 12-bit DAC provides 4096 steps over 6 different ranges selected by relay contacts. A great deal is said about the IEEE-488 interface but relatively little about the analogue components. It is stated that the stability of the output current depends mainly on the stability of the voltage references (plural) whose temperature coefficient is less than 50 ppm/°C. This implies that the coefficient of the reference resistors and DAC are much lower, but this is not clear. There is no consideration of long-term drift or the nature of any noise, and while it is true that chopper-stabilised amplifiers provide very low d.c. drift they are prone to clock feed-through and nothing is said of this or anything done to compensate for the effect. There is no isolation between digital and analogue grounds, but a 4.5 digital meter measures what voltage the DAC actually generates rather than what it is supposed to. This meter may have its own imperfections but potentially it is better than the DAC. Also significant is the description that the design is for a "..high precision.." current source, as in the context of the present work the circuit is considered to provide only modest precision.

2.3 Design ideas for a suitable 1 ppm DAC

Beginning with the assumption that the current source to be designed would contain a DAC to generate a reference voltage followed by a near perfect voltage-to-current converter, it was thought appropriate to investigate further commercial DACs.

There are many semiconductor manufacturers who supply digital-to-analogue converters but after a thorough study of the parts available none was considered able to meet the design target, which requires resolution and monotonicity to 20-bits. Figure 2.1 shows the best of the parts that were studied together with important operational parameters and general comments.

Device	Res.	Mono.	Gain ±ppm /°C	Power mW	Manufacturer	Comments
DAC-HP16B	16	14	15	675	Datel	Resolution too low
AD1145BG	16	16	0.1	2.5	Analog Devices	Lowest power and drift
AD1139J	18	18	4	825	Analog Devices	'Best' DAC found
AD1860	18	15	25	110	Analog Devices	PCM Audio DAC
DAC1138K	18	NA	0.8	900	Analog Devices	Requires external reference to achieve quoted drift.
DAC729KH	18	16	5	1200	Burr Brown	Monotonic to 16 bits, but high power
PCM63P	20	16/17	25	225	Burr Brown	PCM Audio DAC
Texas Instruments, Motorola, National Semiconductor, Plessey - No suitable DACs						

Figure 2.1 - Commercial DACs

Burr Brown describes its PCM63P device as a 20-bit monolithic Audio DAC, but careful inspection of the text reveals a statement which says ".. the extremely low Total Harmonic Distortion (THD) performance is typically indicative of 16-bit to 17-bit integral linearity,... the relationship between THD and linearity, however, is not such that an absolute linearity specification for every individual output code can be guaranteed". These comments are typical of DACs designed for use in Compact Disc audio systems where THD is more important than absolute linearity, and therefore such DACs should be avoided. It is also worth noting that the PCM63P has a full-scale current output of 2.00 mA, which means that a precision current (rather than voltage) amplifier is needed in order to maintain accuracy whilst providing a nominal 100 mA of coil current.

Although the AD1139J offers monotonic performance to 18 bits it is obviously not capable of 20-bit performance and is therefore not sufficiently 'accurate', where 'accurate' is used to mean that

- resolution and non-linearity are < 1 ppm
- temperature coefficient is < 1 ppm/°C
- long-temp drift is below 1 ppm / 100 hours , (say)

What was required is an almost ideal 20-bit DAC, and no commercial source can supply it. Instead a special system must be built. Three different techniques were considered which might achieve this and these are discussed in turn.

2.3.1 Pulse width modulation ($1:10^6$)

It is well known that digitally controlled pulse-width-modulated signals may, in principle, be filtered to produce analogue signals whose level can be increased linearly with respect to digital control signals. An arrangement for a 20-bit converter is shown in Figure 2.2 where the digital demand is continually compared with the output of a 20-bit counter. When **a** is greater than **b** the **a > b** output becomes a logic 1 and causes the analogue switch to connect the resistor to a reference voltage rather than ground. When **REF** $= 2^{19}$ a square wave is produced with a 1:1 mark to space ratio, and a mean value of $V_{ref}/2$ appears after the filter along with unfiltered ripple.

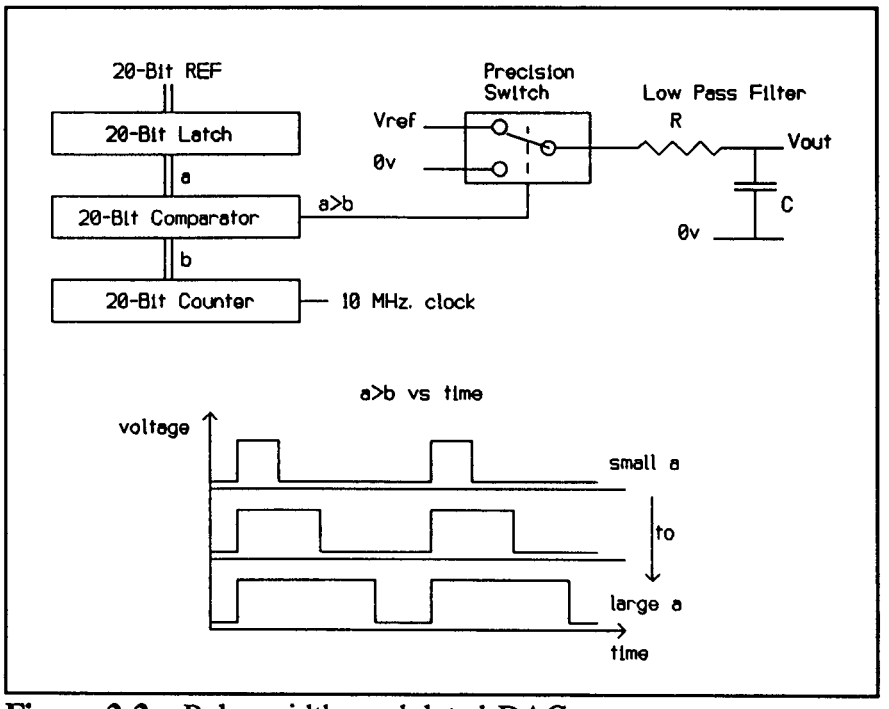


Figure 2.2 - Pulse-width-modulated DAC

With present technologies there are two practical limits to this approach. Firstly the time for the comparison to be performed, and secondly the response time of the analogue switch. If the counter is clocked at 10 MHz, the comparator has only 100 ns to perform its function, and in the worst case, a delay of less than 5 ns per bit is needed. Unfortunately even at 10 MHz. the cycle time is about 0.1 s and requires low-pass filters with long time constants to extract the d.c. component of the waveform. Active filters may reduce the settling time but they introduce their own drifts. Digital circuits may operate at more than 100 MHz and the period T may be reduced, but there is still the requirement for an analogue switch with sufficiently fast response times, and this is not available. Even at 10 MHz, when **REF**=1 only one pulse of 100 ns duration results and the analogue switch must respond within a negligible time of say 5 ns. Charge injection from the digital control line to the filter is also a major limitation. Consequently the technique was considered infeasible.

2.3.2 Multiple level pulse-width-modulated DAC (1:64)

In the previous approach the clock rate is limited by delays in the comparator and temporal characteristics of the analogue switch. It is advantageous to reduce the number of bits in the comparator, so that the cycle time may be decreased without increasing any frequency components. This may be achieved by splitting the range, and Figure 2.3a shows the arrangement when two references are used, together with a three-way analogue switch. The most significant bit (msb) of the latched input is used to decide *which* reference voltage is used, while the **a > b** output still determines *when* it is used. When msb=0, $V_{ref}/2$ and 0v are used, and when msb=1, V_{ref} and $V_{ref}/2$ are used. Saving only one bit in the comparator is not very significant but the idea may be extended in principle to use 2^n reference levels. Unfortunately an analogue switch with 2^n+1 inputs is not very practical when n is greater than about four. However the same effect will be achieved by using an ideal N-bit DAC and modulating its output between levels n and n+1, as shown in Figure 2.3b. With more analogue levels the number of time zones for pulse-width-modulation may be reduced and a higher repetition frequency obtained. The higher the frequency the more easily the signal may be filtered to remove ripple in the 0 to 1000 Hz band to which many precision mechanism can respond.

The idea of modulating the output of a DAC is intuitively correct but requires formal analysis. Consider a converter with a 20-bit overall performance where N-bits are devoted to a normal DAC and M-bits to the control of its output. In particular the M-

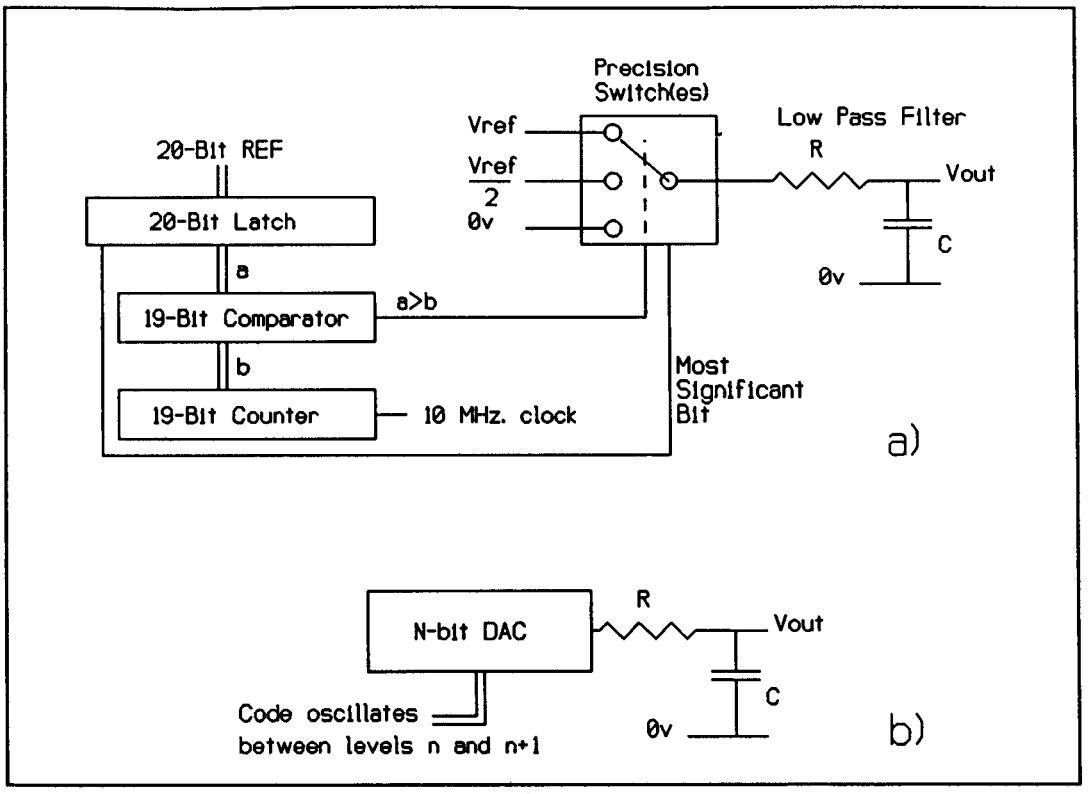


Figure 2.3 - Multiple-level PWM N-Bit DAC

bits determine the ratio of time the DAC spends at its n and n+1 levels.

Let any 20-bit code be n:m where $0 \le n \le (2^N-1)$ *and* $0 \le m \le (2^M-1)$ *and suppose*

$$V_{dac} = f(n) = \frac{n}{2^N} \times V_{ref}$$

then average output = V_{out} *is given by*

$$V_{out} = \frac{f(n) \,.\, (2^M - m) + f(n+1) \,.\, m}{2^M}$$

$$V_{out} = \frac{V_{ref}}{2^N}\left[\frac{n \,.\, (2^M - m) + (n+1) \,.\, m}{2^M}\right] = \frac{V_{ref}}{2^N}\left[n + \frac{m}{2^M}\right]$$

then when

$$m = 0 \qquad V_{out} = \frac{n \,.\, V_{ref}}{2^N}$$

$$m = 2^M - 1 \qquad V_{out} = \frac{V_{ref}}{2^N}\left[n + \frac{2^M - 1}{2^M}\right] = \frac{V_{ref}}{2^N}\left[n + 1 - \frac{1}{2^{N+M}}\right]$$

It can be seen that, for any n, held constant, the output voltage varies linearly with m from the nth DAC level to just below the (n+1)th, and a resolution of 1 part in 2^{N+M} is obtained.

There are conflicting requirements for the values of N and M. On the one hand a small value for M is desirable because this would allow a high repetition rate for the pulse-level-and-width-modulated signal, which could then be filtered more quickly. However, because N+M=20, a small value for M implies that a high precision N-bit DAC is needed, which is precisely what is not available. With current technology a sensible compromise is achieved when M=6 and N=14. By switching between levels n and n+1 in the ratio 0:64 to 64:0 and filtering the output, a piece-wise linear interpolation is obtained. The settling time for the DAC limits the maximum frequency but in principle the output may be filtered much more readily and have a settling time which is only 10 s of milli-seconds. Unfortunately the output of most DACs have a characteristic 'glitch' when switching between levels, which is worst for a majority change. For a 14-bit DAC this occurs at code levels of 8191 to 8192, but still occurs at all other levels to a greater or lesser extent. Consequently the output of the filter will not be an interpolation between levels. Even in the absence of 'glitches' the net output may not have the desired integral non-linearity which is limited by the linearity of the 14-bit DAC. Figure 2.4a shows a typical response of a 14-bit DAC where, for the sake of explanation, the non-linearity appears greatly exaggerated, while Figure 2.4b shows the interpolation between levels around zero, but is representative of the response throughout the range. If the interpolation is exact there is a piece-wise linear fit between adjacent DAC levels, which themselves have a deviation from a straight line. Stated another way, it is clear that the local gains vary and that the integral non-linearity over the entire range is determined by the 14-bit converter and is therefore no better than ½ least significant bit, or 1 part in 2^{15}. The values of N and M may be changed but the nature of the response will remain the same. Once again the technique was considered infeasible. (In fact it is infeasible in its basic form, although it may form the basis of a more subtle approach, see chapter 7.)

2.3.3 Coarse and fine DACs

A well known measurement technique is to use coarse and fine scales. The same principle may be applied to two DAC in order to create a DAC with greater resolution. However it is naïve to believe that two 8-bit DACs may be used to create a single 16-bit DAC simply by summing the two DACs with relative weightings of 1 and $^1/_{256}$. The difficulty lies in the fact that the output levels of the coarse DAC are not evenly spaced and the fine DAC needs to have a different gain factor associated with it for every pair of adjacent levels from the coarse DAC. In this way the full-

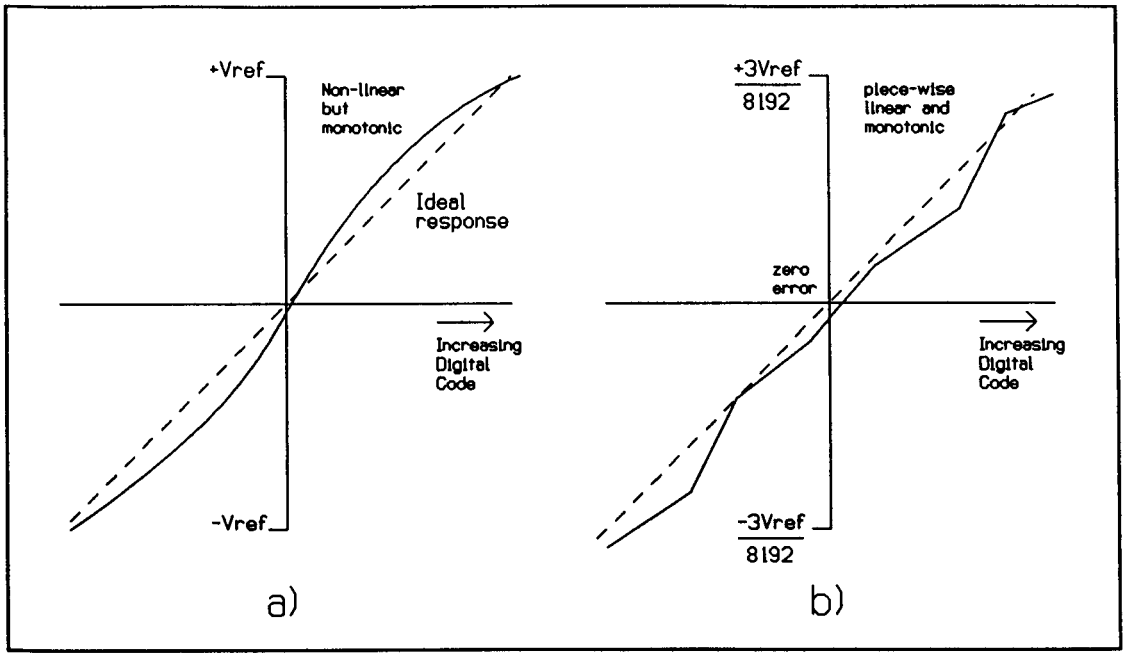


Figure 2.4 - Piecewise linear DAC

range output of the fine DAC may be made to fit between all levels of the coarse DAC. In the case of 8-bit DACs if the weighting of the fine DAC is fixed at $^1/_{128}$ then in principle it may trim the output by ± 1 bit of the coarse DAC. If the differential non-linearity of the coarse DAC is less than ± 0.5 bit, it is possible to create a 15-bit DAC by measuring the actual output of each level of the coarse DAC, and defining the coarse and fine codes so as to achieve any desired value. The overall linearity then depends not on the DACs but on the ADC used to measure the composite voltage. A self-contained DAC does not result, but with a suitable high resolution Analogue-to-Digital Converter (ADC) a subsystem may be formed with the required linearity and resolution. Most 6-digit voltmeters are too slow but an acceptably fast ADC module [2.11] is made by Analogue Devices, see Appendix [A]. It achieves integral non-linearity of ± 1 ppm, and gain drift of ± 1 ppm/°C.

2.4 Minimising risk in the NPL/DTI project

Some of this research work was carried out in order to satisfy the requirements of a contract between the Department of Trade and Industry (DTI), and the University of Warwick (UW) [2.12]. The DTI wanted a measuring facility to be provided for the National Physical Laboratory (NPL) which would give a resolution of 20 pm over a range of 10 μm. After a competitive tender a design based on an X-ray interferometer using a silicon monolith was accepted, with constraints on time and budget. The capital cost of the programmable current sources was expected to be no more than about £10k in a project worth £390k. Development time and costs for one-off designs

are critical and dominate over small cost-saving, changes to the hardware. The economic decision involves taking a secure route in the design strategy which is believed most likely to get close to the full specification even if full compliance is not achieved. This mediates to some extent against introducing newly developed devices for which little practical evidence of exact behaviour. Also, in a private communication with Mr. P. Cooke of Cooke Consulting, who had experience in this kind of project work and detailed circuit design, concerns were expressed about various kinds of pulse modulated techniques when compared with closed-loop methods using coarse/fine DACs and a precision ADC. Consequently maximising the chance of success of the electronic circuits was considered more important than minimising the cost.(In other applications cost may be a significant issue and is discussed further in chapter 7.).

The project required the use of 2 coils to provide forces for translation and rotation. The latter needed to compensate for the effects of parasitic torques introduced by inaccurate machining and/or misalignment of the monolith. Therefore, 2 programmable current sources, or drives, were needed that could be controlled simultaneously. Two particular concerns were identified and given extra consideration. Each is discussed below.

2.4.1 Coil self-heating effects and cures

Each coil operates at near room temperature and the fine copper or silver wire used to form it has a finite electrical resistance. When a current passes through a coil there is local heating and the temperature rises. This results in thermal expansion of the coil, and may reduce the field strength of the permanent magnet which lies within it. Based on the ideas presented in [1.4], if the pole of the magnet lies near the centre of the coil and expansion is symmetric about this point there is negligible change in the force exerted on the magnet. However it is not clear what the local temperature rise might be and how this affects the strength of the field. In discussions with one of the authors, Dr. D.G.Chetwynd, it was thought unlikely that local heating would be a problem, because a) the coil was usually wound on a brass former and bolted to baseplate, both with good thermal conductivity, and b) the optimally designed coils have a resistance of less than 20Ω and operate at 70 mA maximum, giving a power of less than 0.1 W. The magnet does not touch the coil and its temperature is dominated by ambient conditions. However some doubt still existed and in any event the design was to be suitable for other environments.

Consequently it was considered worthwhile to provide a means of regulating the temperature near the coil. One technique considered was to try and maintain the coil at ambient temperature by use of a Peltier heat pump, but the inherent inefficiencies would mean some other part of the apparatus would have a local temperature rise, and it would be necessary to screen against additional periodic electrical noise. Another, and preferred technique, relied on surrounding the coil with a resistive heating 'jacket' as depicted in Figure 2.5, and controlling the current in the resistors such that as the current in the coil was varied, the total power dissipation remained the same. Once in equilibrium, constant power dissipation gives a constant temperature. Rare-earth magnets, based on samarium/cobalt alloys, have a negative temperature coefficient of about -0.025%/°C (as for Incor 30HB) for their intrinsic field strength at room temperature. Therefore it is necessary to stabilise the operating temperature to 0.004°C to achieve a stability of 1 ppm. Based on practical experience it is sensible to put an upper bound on the temperature rise in a coil at 10°C, which means the temperature regulation is at most 1 part in 2500. This suggest the use of a 12-bit DAC driving a jacket of equal resistance to the coil, and with similar voltage swings, is sufficient. It is important that changes in current in the jacket do not cause any change in the magnetic field, and the possibility of cross-effects may be minimised. This may be achieved by using a combination of series and parallel resistors in order to achieve current flows in opposite directions.

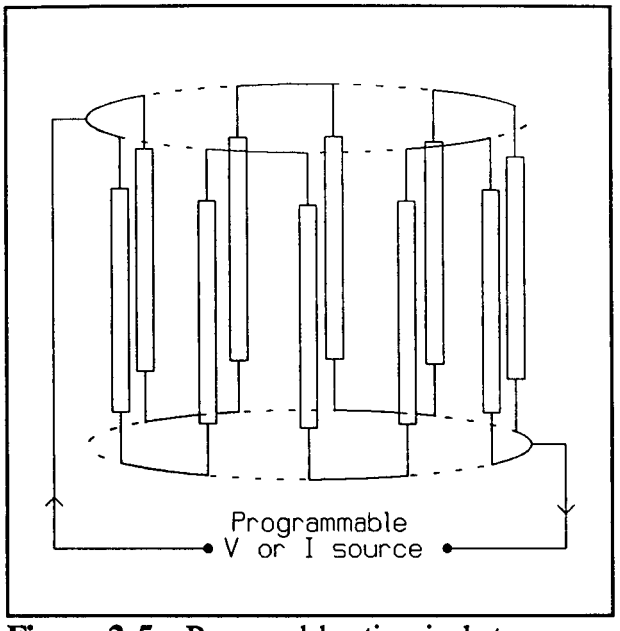


Figure 2.5 - Proposed heating jacket

2.4.2 Embedded versus sub-system controllers

Embedded systems are often tedious to develop because it is difficult to programme microprocessors which interact with real-time events and perform extensive calculations. Initially it was thought that each drive should be self-contained, but upon further consideration it was decided advantageous to introduce a desk-top computer between the (expected) host computer and each of the drives. This 'slave' computer could be programmed much more easily in a high-level language than the assembly

language needed by the embedded microcontroller. Also with the addition of another display, diagnostic information about the performance of the drives could be provided. In this way the program in each of the drives performs the real-time control activities associated with the DACs and ADCs as well as dealing with primitive communications, while the 'slave' PC implements the closed loop control algorithms. Although this approach increases the cost by that of a desk-top PC at the time this was only about £1k and well within the budget. The price was considered small when compared with the increased flexibility. The arrangement is shown in Figure 2.6, where interrupt driven communication buffers simplify passing messages whilst allowing critical foreground activities to continue.

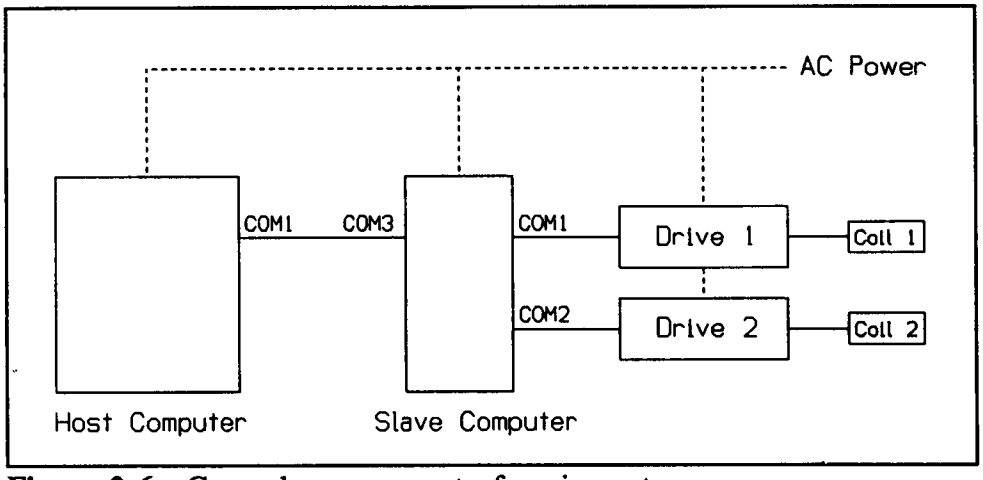


Figure 2.6 - General arrangement of equipment

For 'volume' production there is a significant potential advantage if the cost of the desk-top PC can be saved by making the embedded software perform the control function. In this regard it is appropriate to suppose the design will incorporate an embedded micro-controller with sufficient memory and data-processing capability, so the PC is not needed.

2.5 Proposed design

An elegant and very worthwhile modification to the basic idea of using two DACs and a precision ADC as described in section 2.3.3, is for the ADC to measure the voltage across the voltage-sensing contacts of the reference resistor in a voltage-to-current converter (VtoI). This single modification (innovation) has the effect of making the whole design insensitive to errors and drifts in the performance of the VtoI, since these will be detected by the ADC and the control strategy can modify the DAC levels to provide the required output. So long as the resistance of the precision

reference resistor does not vary, the current in the coil is directly proportional to the measured voltage, and hereafter discussions are based upon voltage levels which are quite reasonably assumed to have a linear relationship to current in the coil. A suitable 4 terminal resistor is available commercially with a nominal drift of 0.0 ppm/°C at 25°C, see Appendix [B]

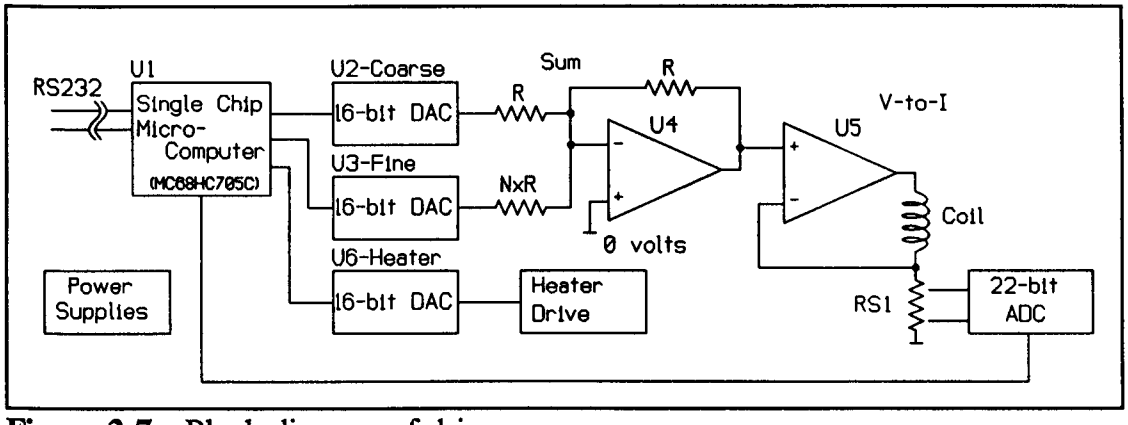


Figure 2.7 - Block diagram of drive

The major components of the proposed drive circuit are shown in the Figure 2.7. Two 16-bit DACs and a 22-bit ADC are combined together to create a 22-bit subsystem. The output of the DACs is summed by U4 but with significantly different weightings (1:N) such that U2 represents a coarse setting, and U3 a fine setting, of the reference voltage seen at the non-inverting input of U5. The feedback around U5 creates a voltage-to-current converter, by ensuring that the voltage across RS1, is very close to that generated by U4. It will not be the exact current that is expected because of the offset voltages in, and finite gain of, U5, as well as errors caused by the end effects of the forcing terminals of RS1. However all these effects are deterministic and repeatable, so by measuring the voltage across the sensing terminals of RS1 with the ADC, the actual current may be calculated. By suitable modification of the DAC codes any required current may be obtained, subject to limits imposed by considerations of the power dissipated by real components. The exact values of the components and the voltage swings depend upon the devices actually employed and are discussed in detail in section 3.4. There it is shown that there are conflicting requirements, but that a useful set of values may be obtained. (In particular N is given a value of 64).

Although U2 is a 16-bit DAC the intention is that only about 2N equally spaced levels will ever be used, and that the ADC can provide an accurate measure of the voltage associated with every one. A preliminary and relatively brief calibration cycle will allow the voltage level associated with each coarse DAC code to be known and held

in a look-up table. Subsequently, for any desired level, the look-up table may be used to identify the exact value of the nearest level, which in turn permits the contribution of the fine DAC to be calculated. The transfer function of the U3 may be assumed from published specifications, but more likely determined by linear regression analysis on the changes measured by the ADC as U3 is varied over its range during an extended calibration cycle. See page 69 for further details.

Figure 2.7 also shows an embedded single-chip microcontroller (U1) whose main function is to decode messages sent from a controlling computer via an optically isolated RS232 serial link. These messages include requests for the DAC levels to be changed and for the reading of the ADC to be passed back. A 16-bit DAC (U6) may used to determine the power dissipated by an electrical heater surrounding the coil, so it may be operated at constant temperature. The heater driver could be a simple voltage power amplifier but it is relatively simple to replicate the voltage-to-current converter and provide a secondary programmable current source of lower resolution, if temperature compensation is unnecessary. The simplicity of this block diagram belies the complexity of the real design which is discussed in the next chapter. Subtle issues such as multiple power supplies with star points, and techniques to avoid aliasing in chopper-stabilised amplifiers are introduced.

Chapter 3 Drive circuits in detail

In various parts of this thesis the word 'drive' is used and there is the possibility of some confusion when reading the text in isolated sections. To clarify the situation the reader is asked to remember that in each of two chassis there are two driver circuits; one connected to a coil and the other connected to a resistive heating element. Depending upon the context, 'drive' may mean one of the two sub-chassis, or one of the two current sources within a particular chassis. Since the two sub-chassis, or drives, are identical descriptions given in terms of one drive apply to both.

The following detailed descriptions relate to various circuit diagrams which appear in Annex [1]. A total of 6 circuit diagrams were produced with the aid of a computer-aided design package called OrCAD SDT [3.1]. A block diagram on sheet 1 shows the main interconnections between the remaining 5 sheets. For the sake of clarity the circuit elements described below are ordered by function rather than sheet number, and several important parts of the circuit have been copied from the Annex and appear along with the descriptions. Also provided in the Annex is a complete listing of components showing the reference name, part code, suppliers order number, function and cross-reference to sheet number.

3.1 Power supplies - *Circuit sheet 2*

The unit is mains powered, with a conventional approach to safety issues. The supply of 240v 50Hz ac enters the drive via a 3-terminal IEC chassis-mounting plug with an integral fuse and filter. The Live and Neutral signals are switched via the double pole switch connected to JP2, while the Earth line is taken to the chassis and shield within each toroidal transformer. The Earth is not taken to the 0v of the circuit and this means the whole circuit is floating. If for some reason it is considered necessary to join the Earth to the 0v of the circuit this may be achieved by joining points on the Printed Circuit Board - J2A (near mains inlet JP1) to J2B (near C14 and REG5). Large voltage transients which pass through the filter are absorbed by a transient voltage suppressor, VAR1.

Three transformers are used to provide power for different parts of the circuit. Each is of the toroidal type in order to minimise magnetic and mechanical interference, and reduce internal heating when compared with conventional transformers on a standard soft iron E-core. One transformer provides an independent supply for the ADC

module, so that a 'star point' may be created within it. This is necessary to avoid unwanted earth loops.

T1 is used to provide +15v and -15v supplies for the precision Analogue-to-Digital Converter module, via regulators REG1 and REG2 respectively. Although these appear to be of a popular generic type, they were selected after careful inspection of the data sheets associated with several similar voltage regulators. Specifically a comparison was made between regulators from Motorola and National Semiconductors (NS). The NS parts have significantly better specifications for line and load regulation, and consequently NS parts were used. During the design stages the importance of the stability of the supply voltages was considered in order to decide if it was necessary to build discrete regulator stages with better characteristics that simple 3-terminal regulators. The supply rejection ratios for the precision ADC and the high current power operational amplifiers in the output stages are so high that precision regulation was deemed unnecessary. However, since so much depends on the 'quality' of the overall design, and component cost was not a significant constraint, the general decision was taken to use higher-grade parts wherever the opportunity existed. This philosophy is shown in the choice of all smoothing and decoupling capacitors. It is important to note that the port labelled 0vADC is connected directly to the power supply input of the precision ADC and that a star point exists within this ADC which joins 0vADC to 0vDRV. Originally it was anticipated that the whole design would be on a single PCB, but the foot-print of the ADC module made this impractical and a daughter-board was created during the PCB design phase. The connections between the main PCB and the daughter PCB are made with a ribbon-cable via JP8.

T2 is used to provide +15v and -15v supplies for all other analogue circuits via regulators REG3 and REG4 respectively. The design is identical to that described above, with the exception that a star point is created between the 0v pins of REG3, REG4 and REG5, where REG5 supplies +5v for the digital circuits. The circuit diagram shows jumpers J1A and J1B with an optional link but they are overlaid on the PCB, thereby creating a permanent connection.

T3 has separate full-wave rectifiers on each of its secondary windings. The choice of a 9v transformer was so that approximately +12v unregulated is available for a miniature dc fan, while an independent +12v supply may be regulated by REG5 to provide +5v. In this way the +5v supply has a high immunity to dips in the mains input because the voltage drop across REG5 is relatively large at about 7v. The

digital circuits require only tens of milli-amps so the power dissipation of REG5 is quite acceptable at about 0.35W. At this level a heatsink is not essential but one had to be provided for the other regulators so it was little trouble to mount REG5 on this heatsink in case modifications and/or additions at a later stage resulted in a much larger current. If modifications are made, up to 100mA may be taken from the +5v rail without concern. The value of C17 is unnecessarily large if the only load is the ventilation fan, however it was easy to incorporate the same value as C13 in the +5v supply and provide another well smoothed power source if needed later.

All supplies incorporate tantalum and ceramic decoupling capacitors to minimise high frequency noise. In particular C2 and C3 for +15vADC; C5 and C6 for -15vADC; C8 and C9 for +15vDRV; C11 and C12 for -15vDRV; and finally C15 and C16 for +5vDIG.

When the completed drives were running it was found that when the switch on the front panel is rocked from the on position to the off position by hand as fast as possible, there is a break in the supply for about 0.25 seconds. During this time there is no significant change in the overall operation of drive circuits.

3.2 Microcomputer - *Circuit sheet 3*

At the outset of the design there was concern over the effect of interference between any microprocessor controlled digital subsystem and analogue circuits. The use of a traditional approach using a microprocessor with external ROM/RAM and I/O etc. was considered but with the required resolution of 1 part in 500,000 undesirable interference effects from high frequency data and address buses were thought quite likely. A number of single-chip microcontrollers offered the facilities to control the various peripheral components with an input/output port programmed to behave as a data-bus. The availability of a low-cost in-circuit emulator, as well as previous personal experience lead to the use of a Motorola MC68HC705C8S microcontroller [3.2] as the digital controlling element U1. It has a power supply requirement of less than 10 mA at 5v, and using I/O ports means that all high frequency signals remain on-chip. The resulting external digital signals have high frequency components of much lower energy than bused systems. A 4MHz quartz crystal connected directly to U1 provides a reference for the asynchronous communications link, as well as local data transfers.

3.3 Precision analogue-to-digital converter module - *Circuit sheet 3*

An analogue-to-digital converter module (ADC) which has a 22-bit resolution is used to measure the voltage drop across the sensing contacts of a precision power reference resistor. This voltage is proportional to the current flowing in the coil and is used to provide a means of calibration and closed loop control. The ADC is made by Analogue Devices with part number AD1175K, and is referred to as a High Accuracy Integrating A/D Converter . A copy of the data sheet appears in Appendix [A] and provides complete information about its characteristics. It was selected because of a number of important features, which include:-

- Low Integral nonlinearity of ± 1 ppm
- Gain stability of ± 1 ppm/°C
- Internal voltage reference stability ± 0.4 ppm/°C
- 16 conversions per second
- Bus orientated interface

In an attempt to minimise 'earth currents' the AD1175K has an internal star point where the digital, analogue and power grounds are joined together. This is the only place where the 0vADC signal is joined to the other 0v rails and shown in Figure 3.1. Consequently when the PCB daughter board containing the ADC module is not plugged into the main PCB the +15vADC and -15vADC signals are floating.

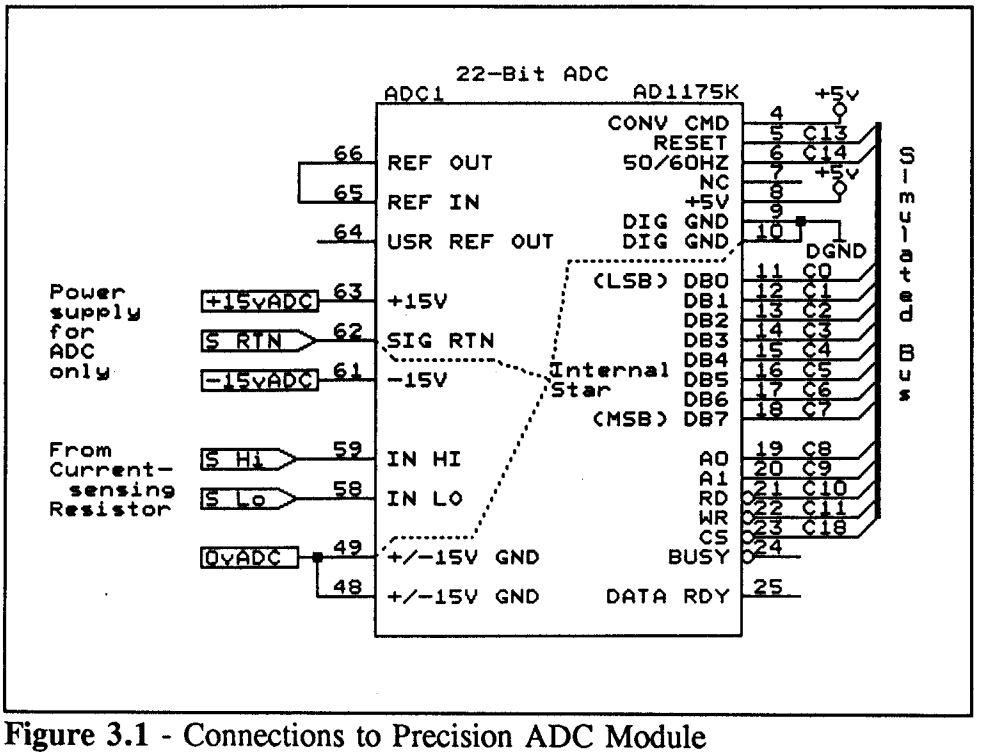


Figure 3.1 - Connections to Precision ADC Module

The ADC is normally connected via address and data lines to a microprocessor, and appears as four byte-wide locations, some of which may be read and written. In this design the address and data lines are simulated by ports on the microcontroller. The program within the microcontroller uses its internal timer to create an interrupt-driven interface which tries to communicate with the ADC every 2.5 milli-seconds. This communication may be to determine the current status, read the previous conversion or start a new one.

In order to obtain maximal rejection of mains/line interference the microcontroller can alter the level on the 50/60Hz pin to suit the operating environment. The present configuration is for 50Hz, but this may be altered by changing the configuration tables within the microcontroller when it is programmed.

3.4 Coil drive circuit - *Circuit sheet 4*

The coil drive circuit was the subject of considerable design effort and was built up by hand prior to the design of a printed circuit board. In principle it consists of a digital-to-analogue voltage converter (DAC) followed by a voltage-to-current converter, however the required resolution necessitates the use of two DACs, a summing amplifier, numerous filters, and a high-precision voltage-to-current converter.

The DACs were selected because of their low power requirements, monotonicity to 16-bits, low temperature coefficient and long-term stability. The data-sheet for the AD1145BG [3.3] shows the gain temperature coefficient to be ± 0.1 ppm/°C and long-term stability to be ± 0.1 ppm/1000 hours. Two similar DACs are used together to implement a control strategy which uses U6 as a primary voltage reference, with U7 acting as a secondary reference, such that the contribution of U7 to the output of summing amplifier U10 is $1/64^{th}$ that of U6. See Figure 3.2. The nominal range of U6 is ± 5v and this is scaled via the drive coil and monolith to a movement of ± 5 micrometres. U6 alone cannot achieve the required resolution of 1 part in 500,000 and U7 is used to 'trim' the output of U6. In this way the DAC combination is considered together as a coarse and fine control. See sections 5.3 and 5.5 for complete description of the programmed used of the DACs.

The ratio of resistors R9 and R10 is 1:64, but this is somewhat arbitrary and not critical, although whatever the ratio is it should not change with temperature or time.

It is vitally important that changes in secondary DAC can always exceed the change between adjacent measured levels used for the primary DAC, so that it can always 'fill-in' between the gaps. The greater the ratio the greater the resolution, but the more the number of measurements during the calibration cycle. Also a lower limit of 5 kΩ is placed on the value of R9 because of the current drive capabilities of the chopper-stabilised amplifier. Using Bulk Metal Film resistors from Vishay Resistive Products [3.10] with temperature coefficients of nominally 0 ppm/°C at room temperature placed an upper limit of 500 kΩ on what was available. A value of 320kΩ represents an acceptable compromise, and results in a ratio of 1:64.

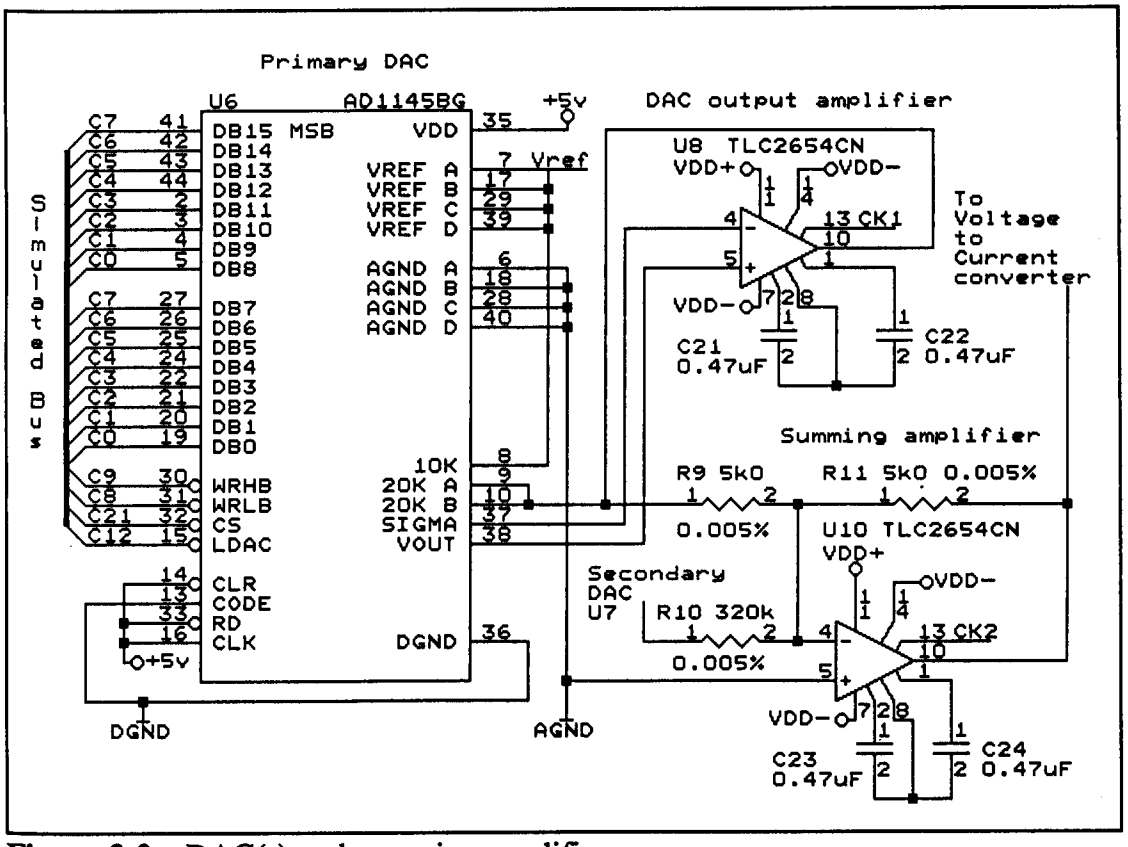


Figure 3.2 - DAC(s) and summing amplifier

To match the stability of the DACs, chopper-stabilised amplifiers are used throughout the circuit in order to achieve very low drift. The actual devices are all type TLC2654AC [3.4], which have a drift with temperature of 0.05 μV(max)/°C, and a long-term drift of 0.02 μV(max)/month. The stability of the voltage reference is 5ppm/°C [3.5] and when considered in conjunction with the nominal operating range of $\pm$5 volts the drifts of the DACs and the chopper-stabilised amplifiers are negligible. At first sight the drift of the voltage reference seems unacceptable but the figure relates to the open-loop drift characteristics and the overall performance depends not on the stability of individual components but on the stability of the current sensing resistor and precision ADC. However it is worth using components

with a good stability as this improves the ability to predict settings for various setpoints, and thereby minimises the time needed to trim dynamic offsets.

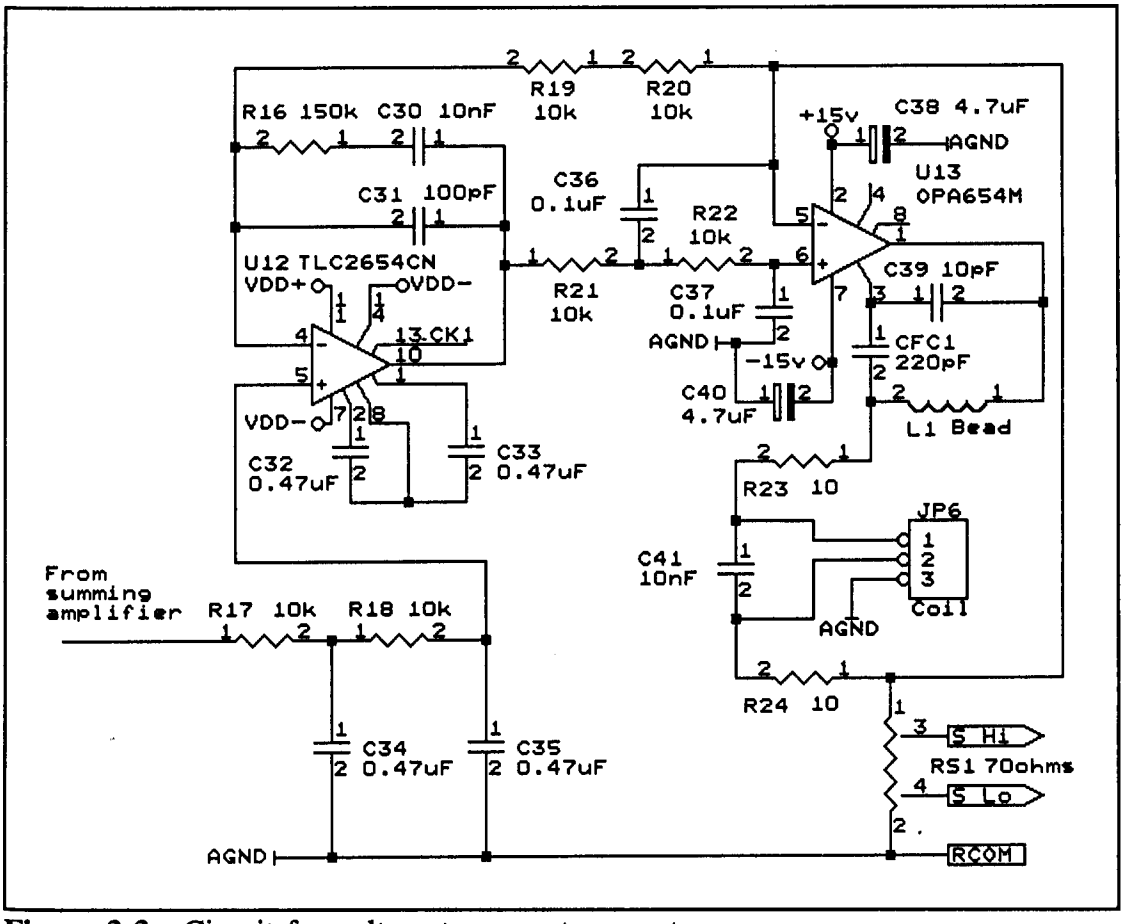


Figure 3.3 - Circuit for voltage-to-current converter

With reference to Figure 3.3, the output of U10 is filtered by R17,C34 and R18,C35 to provide a reference voltage at the non-inverting input of U12. Since the input current of this amplifier is 20 pA (typical) the voltage drop across the resistors is only 0.4μV. Filtering the output of U10 was found to be essential so that fast-rising transients are not seen by the voltage-to-current converter formed by U12, U13 and associated components. Prior to assembly on a PCB, a prototype circuit was build up on the bench and the output of U13 could be made to saturate and latch-up if the input to U12 was changed too quickly. A single-stage delay at the input with a time constant of about 2 ms was sufficient to stop this effect, but a two-stage filter with individual time constants of 4.7 ms was incorporated into the final design. A pragmatic view suggested the resulting rise-time is more than long enough to stop U13 going into saturation. Two analogue feedback loops exist; one directly from the current sensing resistor RS1 to the inverting input of U13; the other from RS1 to the inverting input of U12 via R19 and R20. Two resistors are used here to complement the parasitic effects of R17 and R18. RS1 is a 4-terminal device and the voltage

across the sensing pins must be used as a measure of the current flowing through it. The analogue servo loops ensure that the voltage across the forcing pins equals that derived from the DACs, and this leads to an error caused by end-effects within RS1. However the effects are repeatable and the calibration process using the 22-bit ADC effectively removes them. The gain of U12 is very high at 135dB and C31 provides a high frequency roll-off, while R21,C36,R22,C37,U13 create a second-order low-pass filter which reduces the chopping noise at 10kHz which appears at the output of U12. R16 and C30 were included when the prototype circuit oscillated at several hundred Hertz; similarly C39,L1,CFC1 are needed to stop radio frequency oscillation of U13 and were found empirically. The input bias current of U13 is only 3 pA [3.6], and may be neglected, and although its offset voltage and drift with temperature are unacceptably large the overall performance depends on the characteristics of U12 and not U13. Thermal protection is provided within U13 but R23,C41 and R24 attempt to provide additional protection in the event that the external coil is shorted out, or shorted to 0v. The frequency response, transient response and noise characteristics, of the voltage-to-current converter were extensively studied using computer aided simulation, as well as direct measurement. The analysis may be found in chapter 6.

3.5 Heater/Drive 2 circuit - *Circuit sheet 5*

The required resolution of 1 part in 500,000 prompted concern about the effect of local heating near the coil. It was decided to allow for constant power dissipation, and by implication constant operating temperature, in the vicinity of the coil by the use of a resistive heating element. The idea was to construct a small 'jacket' of resistors around the coil and adjust the applied voltage so that the power dissipated in total by the heater and the coil was a constant. This requires a non-linear decrease in heater current as the current in the coil is increased. To this end an additional 16-bit DAC together with a voltage-to-current converter of exactly the same design to that describe above was built. This allows the flexibility of providing either a heater controller or another coil-drive.

In practice the heat dissipated by the coil is only about 0.3 watts and undesirable effects of local heating have not been apparent. Consequently no heater was actually provided but the controlling program still attempts to modulate the current in the second drive circuit so as to keep the total power constant. To ensure closed loop stability of the output amplifier, the load for the heater is currently a short-circuit; effected by a link in the push connector at the rear panel. When the current in either

drive changes there are small changes in the voltage level of the 0v tracks on the PCB. Although every effort has been made to minimise cross-talk between the two drives by the use of copper planes and star-points, some cross-effects are inevitable. In the final version of the software (named cc1v15fp) the currents flowing in drive 2 are altered synchronously with changes in drive 1, and the calibration procedure compensates for any cross-effects, however if the software is altered to provide two independent drives it may be necessary to compensate for cross-effects in a more sophisticated way.

3.6 Auxiliary devices

There are a number of supporting circuits which together represent auxiliary devices and these are described in turn below.

3.6.1 Power supply for chopper-stabilised amplifiers - *Circuit sheet 4*

The CMOS chopper-stabilised amplifiers are specified at a maximum supply of $\pm$8v, and two linear regulators were build from a dual op-amp and bipolar pass transistors. It was decided to make this supply as stable as possible even though the amplifiers have a high power supply ripple rejection ration of 120dB. The 5v output of REF1 is taken to a non-inverting amplifier U11A, with a gain of 1.59 determined by R12 and R13, which creates via Q2, a voltage of +7.95v. This is inverted with U11B, R14, R15 and Q3 to create -7.95v, which tracks any changes in the +7.95v supply. C28 and C29 provide local decoupling.

3.6.2 Clock generator for chopper-stabilised amplifiers - *Circuit sheet 5*

The chopper-stabilised amplifiers were chosen because the typical chopping frequency is much higher than any mechanical resonances of the monolith and associated mechanical components. This is in contrast with all other commercial chopper-stabilised available at the time (August 1992) which have chopping frequencies of the order of 250 to 450 Hz. Unfortunately their use is complicated by the possibility of 'beat' frequencies and aliasing caused by un-synchronised clocks of similar frequency. Consequently a clock generator was built which provides clocks at the correct frequency, phase and amplitude to synchronise all chopper-stabilised amplifiers. Each

amplifier incorporates a divide-by-two circuit at its clock input which is edged-triggered, and by supplying adjacent amplifiers with anti-phase clocks there can be no 'beat' frequencies, and aliasing is removed because the transient outputs of one stage appear at the input of the next during a time when that stage is not being switched.

Figure 3.4 shows the two SYNC signals which cause amplifies U8,U9,U12 and U15 to operate synchronously but phase shifted with respect to U10 and U16. The signal flow is such that when the output of any amplifier goes to the input of the next it is operating with a different phase.

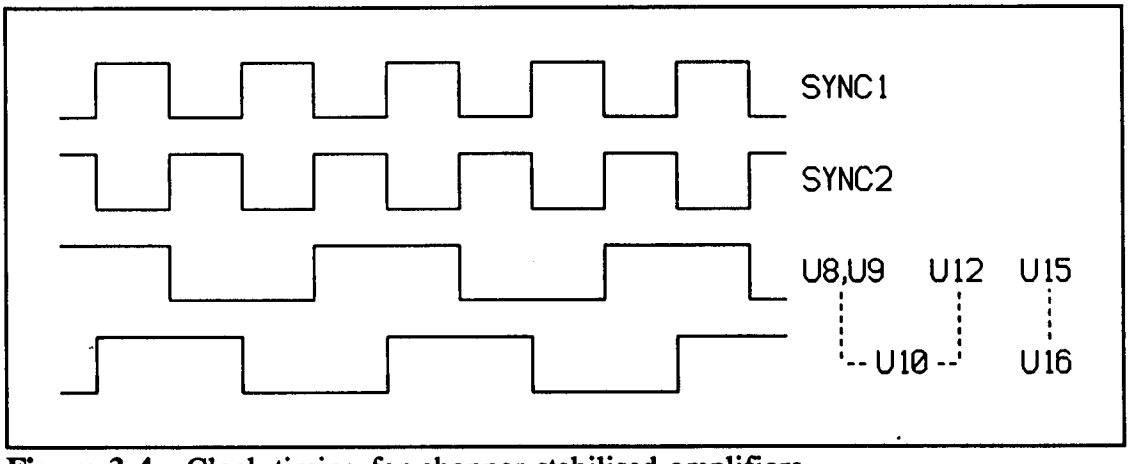


Figure 3.4 - Clock timing for chopper-stabilised amplifiers

The amplifiers require a clock signal which switches from their negative supply to +5v above it, and so U18 is used as a voltage regulator with -7.95v as the 0v reference! The normal 0v is effectively regulated down to about -3v, which creates a supply at +5v with respect to -7.95v. The exact voltage is determined by R25 and R26, and provides current for the high-frequency clock-generator U19, as well as counter U20 and inverter/buffer U21. U19 incorporates a 20MHz crystal-based oscillator and a divider chain which gives 78.125kHz at pin 2, which is further divided by 4 to give 19.53125 kHz for each of the anti-phase clock signals from the inverters in U21. Local division by 2 in each amplifier means that they operate at 9.795625kHz, which is just below their nominal operating frequency of 10kHz.

3.6.3 Monitor ADC and temperature sensors - *Circuit sheet 6*

It is possible for the loads of either drive output to become open circuit and thereby cause incorrect operation of the entire system. If there is no load the output becomes

saturated at about ±13.5 volts and this may be used as an indication that all is not well. U25 is an integrating ADC with 18 bit resolution and is used to monitor the voltage at the output of the power amplifier for each drive. The embedded software passes a number to the slave computer which is related to the measured voltages, and this is multiplied by appropriate constants to give true voltage levels. These constants were found empirically so that the computed voltage agreed with that measured by a Hewlett Packard 5-digit voltmeter. Resistors R36,R37,R38 and R39 attenuate the main drive voltages to provide levels which are in the range of U25, which is about ±1.2v.

The ADC has four programmable digital outputs, of which three are used to control a low-leakage analogue multiplexer,U26. The embedded software uses a timer to create interrupts every 2.5ms and the service routine controls the ADC via dedicated serial communication facilities, called the serial peripheral interface (SPI).

Apart from the drive voltages, the outputs of two linear temperature sensors are also monitored, to determine the temperature of the air at the inlet and the exhaust. If the exhaust temperature is more that about 3°C higher than the inlet temperature there is cause for concern as the internal power dissipation is too high. In fact the absolute accuracy of the sensors means that there is an offset in the measured temperature and the true difference may not be presented. It is therefore better to note the readings when the drive is operating correctly and become concerned if the readings change in the future.

The ADC requires +5v and -5v power supplies and these are obtained locally by REG6 and REG7 respectively, with decoupling provided by capacitors C73,C74,C75 and C76. The stability of the measurements made by the ADC is limited by the voltage reference D3, which has a temperature coefficient of about 20ppm/°C.

3.6.4 Isolated serial link - *Circuit sheet 3*

The drive is controlled by messages which are passed via an RS232 compatible interface. In order to provide isolation of the ground-rail the interface circuitry is optically coupled, via U5. This is a dedicated interface circuit [3.7] which allows for handshake control lines as well as serial data lines. The Request to Send output level may be controlled, or the Clear-to-Send input read, by the microcontroller U1; although in the present software they are not actively used.

3.6.5 Non-volatile memory - *Circuit sheet 3*

At the start of the design it was thought that messages would be sent directly from the host computer to the drive(s) and it would be necessary for control functions to be implemented internally. Since calibration tables are needed to implement the proposed control strategy some form of non-volatile storage had to be included to avoid re-calibration at every power-on. Battery backed Random Access Memory (RAM) could have been used but using an Electrically Erasable Programmable Read Only Memory (EEPROM) obviated the need for a battery and was a practical proposition because many milli-seconds are available for each write cycle during the calibration process. The memory requirement is dependent on the control strategy and for that proposed only a few hundred bytes were needed. Consequently a small 1kByte EEPROM could have been used but the additional cost of providing a much larger memory was negligible when considered with respect to the cost of other components. U4, a 32k Byte EEPROM, [3.8], derives its address from U2 and U3, which are serial-in/parallel-out shift registers connected to the serial peripheral interface of U1. Data is supplied in parallel directly from port A of U1, with three control functions from other ports. Reading and writing is controlled by U1, which uses the internal timer interrupt service routine to synchronise access to U4, so as not to disturb the timing of accesses to the precision ADC (ADC1) or the monitor ADC (U25). When the decision was taken to include a slave PC between the host computer and the coil-drivers it was realised that all calibration information could be held on the hard-disc of the slave PC. This means that the non-volatile memory is not actually used and serves no active purpose, however instructions may be sent over the serial link to write or read any location. See commands T,U and V in section 4.4

3.6.6 Configuration switches - *Circuit sheet 3*

Extensive experience with the design and construction of a number of other embedded control systems has shown that it is useful to be able to alter the performance and/or characteristics by the movement of a few small switches. Consequently an 8-way dual-in-line switch bank was included together with pull-up resistor pack R41, and interface buffer U22. In fact U22 is an octal latch with tri-state outputs, but the latch is wired to be transparent. Once again, when the decision was taken to include a slave PC, changes in performance can be brought about by revisions to the software running on the slave PC. Consequently in the present implementation the switches have no active function, but may be read by appropriate commands if required.

3.7 Component layout and Printed Circuit Board

A substantial amount of time was spent considering the relative positions of the components as a printed circuit board was designed using a software package called CADSTAR [3.9]. A number of constraints had to satisfied, including:-

- Use of a standard 19 inch rack-mounting chassis.
- Separation/spacing of power supply, digital and analogue components.
- Allowing forced air flow over all precision analogue components, especially sensing resistors and the voltage reference(s).

Figure 3.5 shows clearly the relative positions of the components, and in particular the section on the right containing the analogue components which is separated from the power supplies and digital components with an internal baffle. The precision ADC is mounted above the digital components but is shielded by its own casing. Figure 3.6 shows this and the direction of the fins on the heatsinks for the power amplifiers and the current sensing-resistors. The heatsinks were machined from blocks of aluminium so that the fins would be in line with the air flow which is from front to back. It is not clear from the photographs that other holes were drilled at appropriate positions so that air is forced over the power supply and ADC module, to the exhaust fan, without passing over the precision analogue components in the baffled section.

The Printed Circuit Board (PCB) took over 100 hours to design even with the CAD tools and extensive previous experience. The apparent simplicity of the finished layout, shown in centre-line form at 70% of true size in Annex [2], belies the numerous layouts that had to be abandoned because they would not route satisfactorily. A number of important features are either not obvious or not shown, and attention is drawn to the following:-

- The board has 4 layers. Routing top, inner and bottom, with an inner copper layer acting as a shielded between critical analogue routes on other layers. The shield is not the same as a ground plane and there are no steady currents within it. It is joined to analogue ground at only one place near J1A.
- All digital signals are kept well away from analogue signal conditioning done by the chopper-stabilised amplifiers.
- All power-supply routes and critical signal routes use a single layer. Consequently there are no changes in layer using via-holes, and therefore no undesirable parasitic series resistances.

- Where possible signals that pass on different layers without the shield layer between them, do so at right-angles, so as to minimise coupling.
- The digital components, monitor ADC, precision drive and heater drive have 0v rails that are 'starred' out from the voltage regulators, so as to minimise cross-effects.
- Where possible, and useful, the analogue 0v routes have sheet copper overlays to minimise voltage drops.
- The RS232 signals are well away from all other routes.
- Mains 240v ac. is carried on wide routes on the bottom side of the board, underneath the toroidal transformers, but there is no shield in this region, and therefore no possibility of electrical breakthrough.

Technician support was sought for the layout, but it was soon apparent that it is only with a critical understanding of the whole circuit that necessary constraints may be identified and accommodated. Initial layouts were not optimal, and eventually the whole layout was done personally.

To complete this chapter, Figure 3.7 shows the appearance of the complete system.

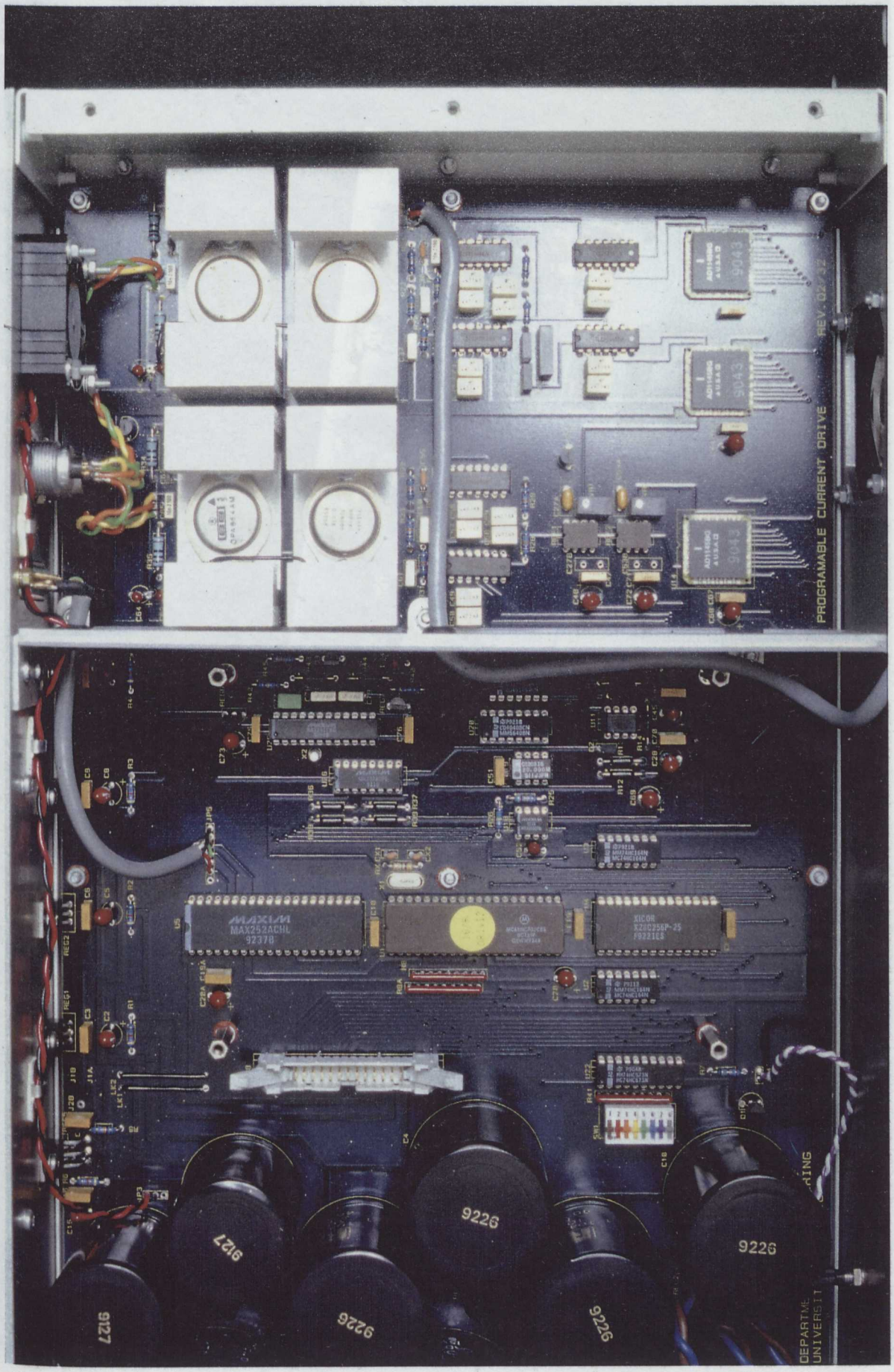


Figure 3.5 - Top-view of assembled PCB

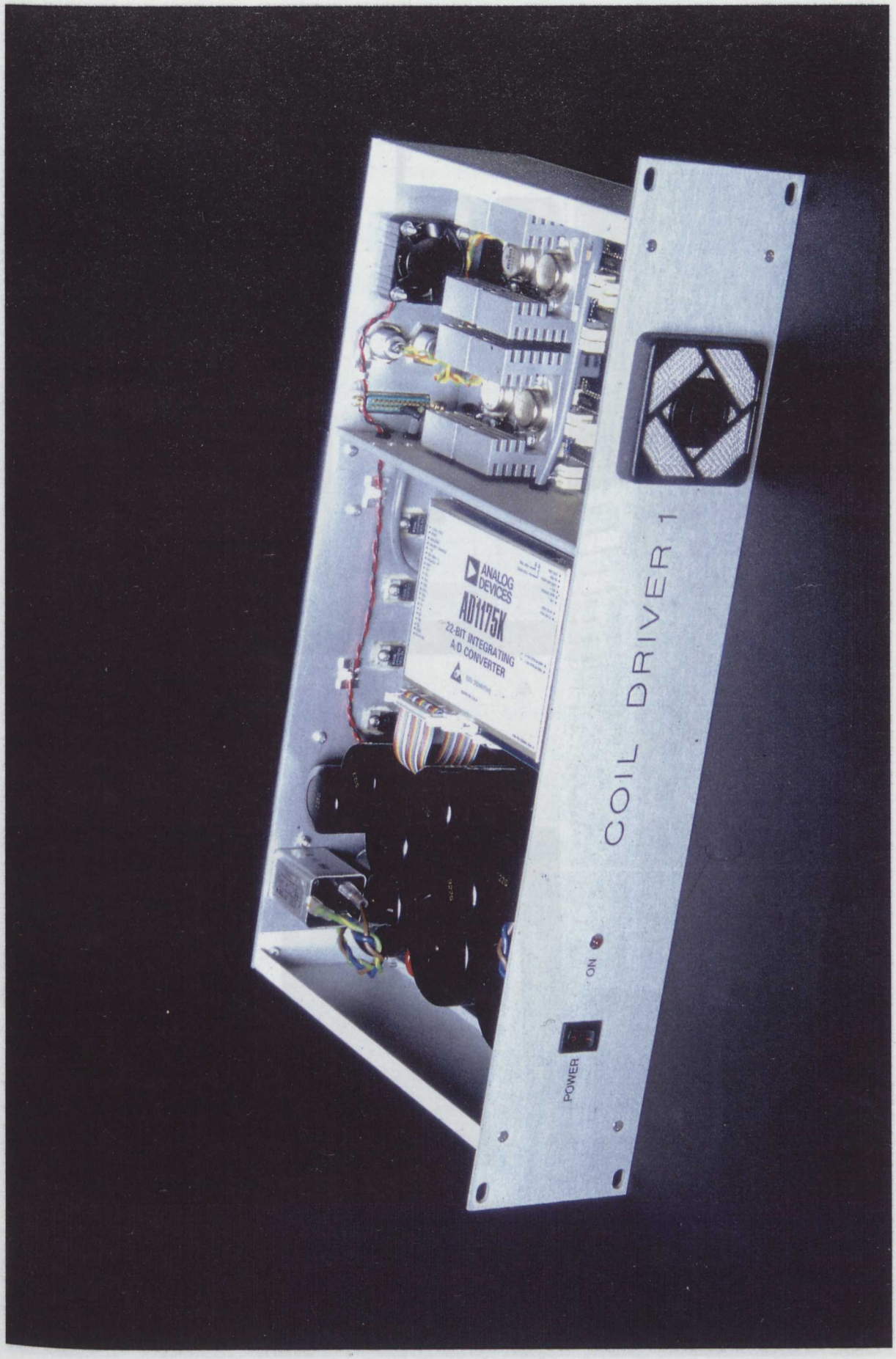


Figure 3.6 - Oblique-view of chassis and PCB

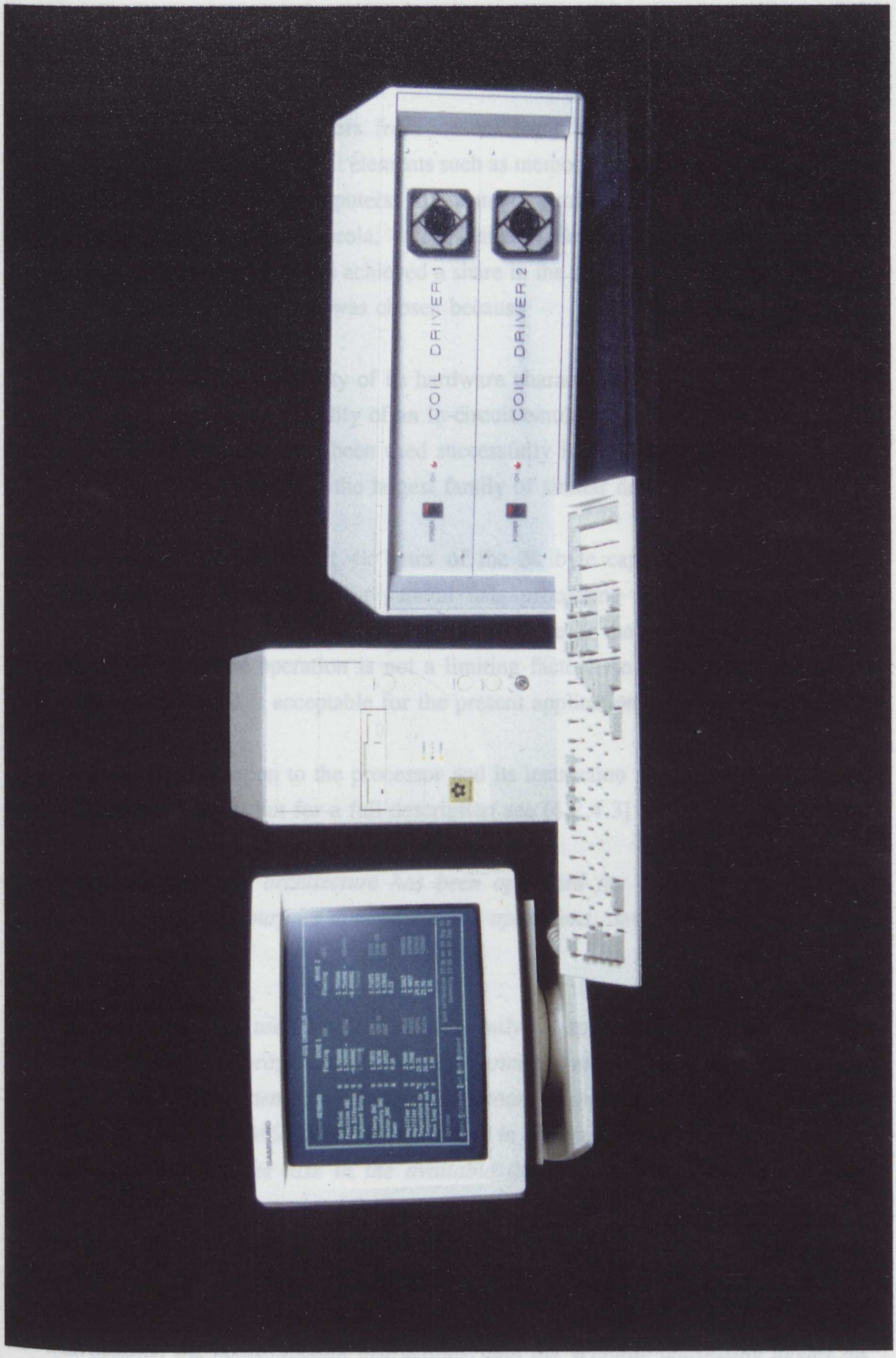


Figure 3.7 - Appearance of completed system

Chapter 4 Embedded Software

Many different microprocessors from various manufacturers have been integrated together with peripheral circuit elements such as memory, parallel ports and counters, to form single- chip microcomputers. All the major semiconductor manufacturers such as Texas Instruments, Motorola, Intel, National Semiconductor etc. now make microcontrollers and each has achieved a share in the market with niche products. A microcomputer by Motorola was chosen because

a) of the suitability of its hardware characteristics
b) of the availability of an in-circuit emulator
c) it had already been used successfully in of other applications
d) it comes from the largest family of similar devices.

The software occupies about 4k bytes of the 8k byte capacity and although the processor was not designed for general data processing it manipulates numeric variables faster than either the ADC conversion-rate or the serial communications-rate, so its speed of operation is not a limiting factor. An up-date rate of several readings per second is acceptable for the present application.

For a brief introduction to the processor and its instruction set the following text is included from [4.1], but for a full description see [4.2,4.3]

"The M6805 Family architecture has been optimized for controller applications, rather than general purpose data processing operations. Several features contribute to this optimisation.

The instruction set, used in the M6805 Family, is specifically designed for byte-efficient program storage. Byte efficiency permits a maximum amount of program function to be implemented within a finite amount of on-chip ROM. Improved ROM efficiency allows the M6805 Family to be used in applications where other processors might not perform the task in the available ROM space. More features may be included in applications where the ROM space is more than adequate. In some cases the user may wish to include programs for more than one application. In such cases the appropriate program could be selected by the power-up initialization program. The ability to nest subroutines, the addition of true bit test and manipulation instructions, the multi-function instructions, and the versatile addressing modes all contribute to the byte efficiency"

In 1980 when this was written, program storage space was at a premium, but now there are family members with 32k bytes of program memory, and byte efficiency is not such a key issue in the design process. However the fact that the family dates back over 14 years means that the parts available today have been greatly refined and are the result of considerable research effort and development. In particular, in the MC68HC705C8S the processor core has been integrated onto a single chip with 8k bytes of EPROM, 304 bytes of RAM, an asynchronous communications controller, 24 bi-directional Input/Output lines, a 16-bit timer, and a synchronous serial peripheral interface (SPI).

It is not practical to describe in exhaustive detail the purpose of every instruction in the embedded software, since this may be inferred by reading the annotated listing which appears in Annex [3]. However it is appropriate to describe some of the more important features, and these appear below. In order to refer to various parts of the listing label names are shown in *italics*, line numbers a 5-digit decimal strings, and addresses as 4 hexadecimal characters preceded by '$'. A cross-reference table for all labels is shown at the end of the listing.

4.1 Initialisation

After a hardware reset when power is first applied the software configures the microcontroller for operation in the particular hardware environment by setting up a number of special locations (see *cfig* at line 02076) and then waits for a command to be received via the serial link from the controlling computer. Port A is manipulated so that it appears like a data bus in an ordinary microprocessor based system, but it changes only when necessary so as to minimise the electrical noise produced by its output stage. Port B is a collection of control lines for enabling read/write functions in other devices, and Port C acts as individual 'chip select' lines. The pins associated with port D are used for specific dedicated functions such as serial input and output.

All pins of the microcontroller are used except for -IRQ, PC0 and TCMP. The allocation of each pin is summarised in Figure 4.1.

MC68HC705C8S Pin No.	Circuit Label	Port ID Name	Function
11	C0	PA0	DB0 (Data Bus LSB)
10	C1	PA1	DB1
9	C2	PA2	DB2
8	C3	PA3	DB3
7	C4	PA4	DB4
6	C5	PA5	DB5
5	C6	PA6	DB6
4	C7	PA7	DB7 (Data Bus MSB)
12	C8	PB0	NWRLB/A0
13	C9	PB1	NWRHB/A1
14	C10	PB2	NRD/NOE
15	C11	PB3	NWR
16	C12	PB4	NLDAC
17	C13	PB5	RESET of AD1175K
18	C14	PB6	50/60Hz select of AD1175K
19	C15	PB7	RTS auxiliary RS232 output
28	C16	PC0	not used
27	C17	PC1	-CS switch input select
26	C18	PC2	-CS AD1175 select
25	C19	PC3	-CS AD1145BG Heater select
24	C20	PC4	-CS AD1145BG Secondary DAC select
23	C21	PC5	-CS AD1145BG Primary DAC select
22	C22	PC6	-CS MAX132CNG ADC select
21	C23	PC7	-CS RAM/Flash EEPROM select
29	C24	PD0	RS232 input
30	C25	PD1	RS232 output
31	C26	PD2	MISO from MAX132CNG
32	C27	PD3	MOSI to MAX132CNG and Address Generator
33	C28	PD4	SCLK to MAX132CNG and Address Generator
34	C29	PD5	-SS tied to +5v
-	C30	PD6	Does not exist - preserves numbering
36	C31	PD7	CTS auxiliary RS232 input
37	C32	TCAP	unused capture input tied to +5v
35	C33	TCMP	output to LED via driver
2	C34	-IRQ	-Interrupt tied to +5v
1	C35	-RESET	-Reset from LM2925T regulator

Figure 4.1 - Port allocation

4.2 Serial interface and command structure

The microcontroller incorporates an asynchronous serial interface and this is configured to transmit and receive at 9600 baud, with 1 start bit, 8-data bits and 1 stop bit. As individual characters are received and assembled by the *rdl* routine at 00461 the most significant bit (msb) is forced to zero. During transmission the msb is always zero.

Commands take the form of a command letter, which may be either upper or lower case, optionally followed by parameters which depend upon the actual command. If the command is recognised a reply is generated which terminates in the sequence 'Y'<CR><LF>, however if the command is not recognised or the parameters are in some way invalid the reply consists of 'N'<CR><LF>. These sequences are

thought of as 'ack' and 'nack'. When a command line has been received its *execution* may take many milli-seconds, and it is important for the controlling computer to wait for an acknowledgement sequence before sending a further command. This is necessary because received characters are not buffered.

Commands are provided for many functions which include reading the Precision ADC and setting the levels of the DACs. Some of the commands are included for diagnostic purposes and were used during the development of the drives and are of little general use. Commands 'X', 'Y' and 'Z' are used for normal operation and are described in detail later.

4.3 Timer interrupt and service routines

The hardware timer, its related registers, and service routine, create an interrupt every 2.5 milli-seconds which is independent of all other activity. Within the service routine *tims* at 02135 are calls to *padcs* at 02172, and *adcs* at 02297, which read the precision ADC and the monitor ADC respectively. The precision ADC is used to measure the voltage across the current-sensing resistor, while the monitor ADC provides measures of such things as the air inlet temperature, and the voltage at the output of the power amplifiers. The overall effect is to access the ADC's at a regular rate and provide the user with the most recently read values in specific memory locations. Not every interrupt causes an ADC value to be copied to memory, as both ADCs take 62.5ms for each conversion.

A cyclic buffer is used to store the most recent 16 samples from the precision ADC so that a moving average may be calculated. Every timer interrupt routine *padcs* at 02172 first calls *getst* at 02183 to discover if the ADC is busy. To do this the simulated address select lines PB0 and PB1 are both cleared to gain access to the status register. When the select line PC2 (also called N1175) is put low together with PB2 (also called NRD) the status register drives port A, which is read by the microcontroller. Only one in twenty-five calls to *getst* will show the ADC to be ready, and when this happens *getr* at 02195 is called to read the latest conversion, followed by *putcm* at 02210 to output the DEFCON command to the ADC which starts another conversion. Once the ADC has been restarted, the last reading is buffered to allow the average of the last 16 readings to be found. 16 was chosen because the average corresponds to the mean of the readings taken over the last second, and division by 16 is easily implemented in binary as 4 right shifts. Buffering

and averaging is done by a call to *sampb* at 02223. For more information on controlling the precision ADC see the data sheet for Analogue Devices part AD1175K in Appendix [A].

Control of the monitor ADC is complicated and determined by a *STATE* counter which increments every timer interrupt (see *adcs* at 02297), and only when this counter has certain values does additional activity take place. Since 8 inputs are monitored a new block of readings is available every 8 x 0.0625 seconds or 0.5s. The monitor ADC is Maxim part MAX132CNW. Unfortunately some of the technical descriptions in its data sheet [4.4] are confusing and much time was spent making the service routine work correctly, and efficiently. When a reading has been made the parallel output bits of U25 are altered to agree with the *CHAN* counter, so that a new analogue channel is selected with the multiplexer U26.

The routine *tims* also manages variable *TOUT* which is decremented if non-zero every 2.5ms, as well as controlling the blinking of the light emitting diode on the front panel. The blinking action is accomplished by co-ordinating the overflow flag, the *OLVL* bit in the Timer Control Register, and the values of *BLSTAT, BLCNT, BLIVL.* See lines 02136 to 02155.

4.4 Command decoder and command execution

There is provision for up to 26 commands, each beginning with a single letter from the alphabet. *rdl* at 00461, and *cmd* at 00491 are called in sequence near *main1* at 00450, to read a line from the serial interface and then decode it. The line is terminated by <CR>. Use is made of a command table, *cmdtb* at 00509, which defines the start address of each routine. Unallocated commands are all associated with a 'nul' routine, while all others have a label which includes the command letter to allow for easy identification. The table is shown in Figure 4.2.

For a detailed understanding of the operation of each command the listing in Annex [3] may be studied at the relevant address. A brief description of the more important commands follows. All commands were used during the development of the drives, but only commands X,Y and Z are currently used by the software in the slave PC.

```
00508                          * a list of command from A..Z
00509  0288 033D   cmdtb  fdb   cmdA              synchronises ADC then outputs 8 numbers
00510  028A 035F          fdb   cmdB              repeated cmdA
00511  028C 0367          fdb   cmdC              repeatedly outputs PADC, averaged PADC
00512  028E 0388          fdb   cmdD              repeatedly outputs averaged PADC
00513  0290 02BC          fdb   nul               E
00514  0292 02BC          fdb   nul               F
00515  0294 02BC          fdb   nul               G
00516  0296 03DA          fdb   cmdH              Prepare Heater DAC
00517  0298 03A7          fdb   cmdI              Output a ramp to Heater DAC
00518  029A 02BC          fdb   nul               J
00519  029C 02BC          fdb   nul               K
00520  029E 0404          fdb   cmdL              Load all DACs - pulses -LDAC line
00521  02A0 0410          fdb   cmdM              set MODE byte
00522  02A2 02BC          fdb   nul               N
00523  02A4 02BC          fdb   nul               O
00524  02A6 041A          fdb   cmdP              Prepare Primary DAC
00525  02A8 0461          fdb   cmdQ              Read precision ADC 20 times
00526  02AA 0471          fdb   cmdR              Read precision ADC as 6+2 hex
                                                  characters
00527  02AC 0495          fdb   cmdS              Prepare Secondary DAC
00528  02AE 04C2          fdb   cmdT              Enable/Disable software protection of
                                                  EEPROM
00529  02B0 057B          fdb   cmdU              Read DIL switches as hex pair
00530  02B2 05C3          fdb   cmdV              Read bytes from EEPROM
00531  02B4 05EA          fdb   cmdW              Write a byte to EEPROM
00532  02B6 0615          fdb   cmdX              Read moving average of precision ADC in
                                                  Hex
00533  02B8 063B          fdb   cmdY              Read all MAX132 channels in hex
00534  02BA 064C          fdb   cmdZ              Set some/all DACs

00536                          * nothing to do
00537  02BC 81             nul   rts
```

Figure 4.2 - Command address table

COMMANDS H,P,S,L,Z:

In order to change the level on a DAC output it is necessary to write to the high- and low-order bytes of its input latch, and then pulse the Load-DAC pin. Bytes may be written in any order to each of the three DACs without altering the voltage outputs. It is only when -LDAC is put low that the input buffers are copied to the main DAC latch. In this way all DACs may be made to update their outputs simultaneously. Command H,P and S prepare the Heater, Primary and Secondary DACs respectively, and command L pulses -LDAC. In order to provide the expected interpretation to the parameters it is necessary to use the 16-bit two's-complement value of each parameter, so that a positive parameter produces a positive voltage at the output of power amplifiers. Each parameter should be a sequence of 4 Hexadecimal characters, where $8000 corresponds to -5 volts and $7FFF with almost +5 volts. For example 'P0000' will prepare the primary DAC for zero volts next time command L is received.

The assembly language code associated with command P is shown in Figure 4.3, and is identical in structure with that for commands H and S.

```
00774  * write to Primary DAC buffer, 2's complement of input parameter
00775 041A CD 042D   cmdP   jsr   getpw                  get word parameter
00776 041D 24 0D            bcc   cmdPz                  part is missing
00777 041F CD 043C          jsr   paneg
00778 0422 B6 BE            lda   PARH                   output high part
00779 0424 AD 21            bsr   cmdPh
00780 0426 B6 BF            lda   PARL
00781 0428 AD 2A            bsr   cmdPl
00782 042A 1E B9            bset  ACK,MODE               acknowledge
00783 042C 81        cmdPz  rts
.....
00804 0447 9B        cmdPh  sei                           protect sequence
00805 0448 B7 00            sta   PA                     output high byte to Primary DAC
00806 044A 13 01            bclr  NWRHB,PB               select high byte
00807 044C 10 01            bset  NWRLB,PB               but not low byte
00808 044E 1B 02            bclr  N1145P,PC
00809 0450 1A 02            bset  N1145P,PC
00810 0452 9A               cli                          allow interrupts again
00811 0453 81               rts

00813  0454 9B       cmdPl  sei                           protect sequence
00814  0455 B7 00           sta   PA                     output low byte to Primary DAC
00815  0457 11 01           bclr  NWRLB,PB               select low byte
00816  0459 12 01           bset  NWRHB,PB               but not high byte
00817  045B 1B 02           bclr  N1145P,PC
00818  045D 1A 02           bset  N1145P,PC
00819  045F 9A              cli                          allow interrupts again
00820  0460 81              rts
```

Figure 4.3 - Preparing primary DAC

The parameter is negated by a call to *paneg* at 00777 to compensate for the signal inversion caused by summing amplifier U10. Notice the inclusion of the instruction **sei** at 00804 (and 00813) which disables interrupts until **cli** at 00811 (and 00819), so that the value output to port A, cannot be disturbed by the timer interrupt service routine which has also to manipulate the contents of port A to control the precision ADC. These regions of code were not originally protected in this way and the result was for the DACs to be updated with spurious values occasionally.

Command Z calls *cmdZ* at 01088 which then calls the routines corresponding with commands S, P and H in turn. Each accepts one parameter and prepares its own DAC, and so long as there is at least one parameter -LDAC is pulsed. This command was included to reduce (slightly) the time delay associated with sending three separate commands. Also, it is possible to change just the secondary DAC if only one parameter is present. For example 'Z7FFF', puts the secondary DAC to its maximum positive value without altering either of the other DAC's. Similarly 'Z00000000', will clear the secondary and primary DACs, but leaves the heater DAC unchanged. The order S,P,H, was chosen deliberately, based on the frequency with which the various DACs may need to be updated.

In the examples below of commands Y and X the following command had been previously sent

z200040006000

which places the secondary DAC at 1.25v, primary DAC at 2.5v, and heater DAC at 3.75v

COMMAND X:

Results in a reply of 6 hexadecimal characters corresponding to the average of the most recent 16 readings obtained from the precision ADC. Since the conversion rate is 16 per second, this corresponds to an average of the readings taken in the last second. The reading is a 22-bit number placed in a 24-bit field biased by $400000, such that positive full scale = 5volts = $600000, and negative full scale = -5volts = $200000. For more information on the coding scheme see the data sheet for Analogue Devices part AD1175K in Appendix [A]

With the previous Z command active command X gave the reply

502079Y

In an ideal system with no offsets the reply should be

($2000/64+$4000)*64+$400000 = $502000

Which shows there to be an offset of

($502079-$502000)/$200000*5 = 0.000288V = 288μV

COMMAND Y:

The software in the intermediate computer (slave PC) makes use of command Y to access the conversions made by the monitor ADC. Study of *cmdY* at 01077 shows that readings from all 8 channels are first copied to a buffer area which is not corrupted while further readings are made during the timer interrupt service routine. This area of 24 bytes is output as 48 hexadecimal characters. Each reading is represented by 6 characters, with a somewhat obscure coding structure, but which may be manipulated relatively easily by the Pascal program in the slave PC. See function *max132_hex_to_fp* at line 320 of listing in Appendix [4]. The order of the readings corresponds with the order of the signals on the analogue multiplexer, U26. That is to say, Agnd, Slo, Drive1, Drive2, Temp_In, Temp_Out, Agnd, Agnd. See circuit diagram sheet 6 in Annex[1].

A typical reply is

0000BD00048A01359A01A78901D08601F1700000D50000D4Y

which is split as

```
0000BD 00048A 01359A 01A789 01D086 01F170 0000D5 0000D4 Y
```

and for drive 1 with v_{ref}=0.61772 V these correspond to voltages at the monitor ADC of

```
0.00035  0.00214  0.14599  0.19971  0.21904  0.23456  0.00039  0.00039
Agnd     Slo      Ampl     Amp2     Temp_In  Temp_Out Agnd     Agnd
```

with interpretation after shifting and scaling of

```
0.0v     0.00179V  3.52454V  4.82466V  21.87°C  23.42°C
Agnd     Slo       Ampl      Amp2      Temp_In  Temp_Out
```

COMMANDS T,U,V:

The non-volatile memory, is an Electrically Erasable Read Only Memory (EEPROM), and is accessed by commands T, V, and W. Sending 'TP' will protect the EEPROM from further write cycles until 'TU' is sent to un-protect the device. The protection scheme is internal to the EEPROM and will remain in effect even if the power is removed. That is to say the protection scheme is itself non-volatile. When unprotected, command W may be used to write a byte at a given address. The format is 'Waaaadd', where aaaa is the address in hexadecimal from $0000 to $7FFF, and dd is the data byte also in Hexadecimal. Once written, the data is read back and checked locally; if the data is not as expected the ACKnowledge bit in the MODE byte is not set and the reply to the command is 'N'<CR><LF> rather than 'Y'<CR><LF> for a successful write operation. The most likely cause for a negative reply is that the memory is still protected. It is always possible to read the memory by using the V command, where the format is 'Vaaaann'. Again aaaa represents the starting address, but nn is a count, in hexadecimal, of the number of bytes to be read. For example 'V010020' will read and send back 32 (decimal) hexadecimal pairs starting at address $0100.

Non-volatile memory is included to allow important calibration data to be stored locally and thereby permit stand-alone operation. A capacity of 32k bytes far exceeded the needs of any control algorithm that was considered, but it was cost-effective to include this amount of memory.

4.5 Floating-point interpreter

In the belief that local (or embedded) closed-loop control was to be performed by the software, many routines were written for floating-point arithmetic. However when it was decided that an intermediate computer would perform the control function these routines were no longer needed. (See section 2.4.2 for comment on this decision). Considerable time was spent developing these routines, which were tested extensively. A brief description is included for completeness and in case future developments warrant the use of embedded arithmetic functions.

An electronic Bulletin Board run by Motorola [4.5] and containing software for the MC6805 family of microprocessors was accessed during a search for floating-point routines, but unfortunately only simple integer routines for 16-bit numbers were found. Further investigation revealed a source-code listing for a Stack Orientated Arithmetic Processor [4.6] for the 6800 microprocessor. Unfortunately there are no instructions in the 6805 processor for stack-relative addressing and direct translation was not possible. However the source-code did provide a valuable insight to several techniques; notably the use of unsigned arithmetic operations to support signed floating-point operations.

In the interests of speed of operation a floating-point format was used which makes use of an 8-bit biased exponent with a normalised, signed, msb suppressed, 24-bit mantissa. Each floating-point number occupies four bytes and allows numbers to be expressed to about 7 decimal places over 76 orders of magnitude. An exponent of zero means the number is zero. For examples of the encoding scheme see the predefined constants at 02062.

Storage has been allocated for 8 floating-point accumulators starting at $0030. These accumulators are numbered from 0 to 7, of which FP0..3, may be used implicitly by various routines, and in particular FP1 and FP2 have special significance. For example *fpmul* at $0882 multiplies the number in FP1 with that in FP2 and the result replaces FP1, with FP2 left undefined.

It is possible to call various floating-point routines directly in the order they are required but to simplify the process of writing sequences of floating-point operations an interpreter was written which calls floating-point routines depending on the value of a code byte which is taken from a table of code bytes, at an address given by the programmer. Code bytes are divided into left-hand and right-hand nibbles, and each

	Code	Time us	xxx	Operation
Main Group	0		nop	no operation, used to complete a byte
	1	90	sld1	short load of FP1 using low nibble as address
	2	90	sst1	short store of FP1 using low nibble as address
	3	90	sld2	short load of FP2 using low nibble as address
	4	70	ld1	load FP1 - using next byte as address
	5	70	st1	store FP1 - using next byte as address
	6	70	ld2	load FP2 - using next byte as address
	7	260	add	FP1=FP1+FP2
	8	260	sub	FP1=FP1-FP2
	9	350	mul	FP1=FP1*FP2
	A	1850	div	FP1=FP1/FP2
	B		br	branch group
	C		fun	function group
	D		cnt	load/dec and branch counter group
	E		adc	load FP1 with ADC conversion; low-nibble=channel
	F		undefined	
Branch group	0		bra	branch always
	1		beq	branch if FP1=0
	2		bne	branch if FP1<>0
	3		bpl	branch if FP1 sign bit = 0
	4		bmi	branch if FP1 sign bit = 1
	5		call	reserved
	6		ret	reserved
	7		exit	leave interpreter
	8		wt	wait if keyboard activity
	9		syn	synchronise with CHAN counter
	A		bc	branch if bits clear
	B		bs	branch if bits set
	C		dly	delay in 0.1 sec intervals
	D,E,F		undefined	
Function group	0		clr	FP1=0
	1		abs	FP1=abs(FP1)
	2		neg	FP1=-FP1
	3		sgn	FP1=-1,0,+1 matching sign of FP1
	4	4800	sqt	FP1=square root(FP1) [4.6] - uses FP0,1,2,3
	5		sqr	FP1=FP1*FP1
	6		swp	FP1<->FP2
	7		fix	fix FP1
	8		flt	float FP1
	9		int	find integer nearer zero
	A		frac	remove integer part
	B		out	output FP1 in fixed point format to RS232
	C		crlf	output CRLF to RS232
	D		spac	output a space to RS232
	E,F		undefined	

Figure 4.4 - Interpreter operations

nibble is associated with a particular function or sub function. It is not necessary for the programmer to remember the actual value of each nibble because macro definitions were written which achieve the translation automatically. The programmer types 'fxxx' and the assembler creates references to 'fpxxx' where xxx is one of the operations listed in Figure 4.4. When an address is needed for an operand it is encoded in a special way and may cause access to the microcontroller's internal RAM, ROM or external EEPROM. See routines *compx, cstol* at $07E0 and $07E7 respectively.

As an example in the use of the interpreter, consider the calculation needed to determine the correct setting for the heater drive circuit in order to maintain constant total power dissipation. Since the power in the drive coil is proportional to the square of the voltage across the current-sensing resistor, which in turn is proportional to reading of monitor ADC channel 2, the voltage setting for the heater drive is given by:

$$Heater\ voltage = K_1\sqrt{K_2 - ADC2^2}$$

K_1 is a function of coil, heater and reference resistances, and ADC reference voltage; and K_2 is desired total power.

Figure 4.5 shows how this formula may be evaluated using the interpreter.

```
* call interpreter
00608   032D   AE 03    CL  ldx    #L/256          pointer high
00609   032F   A6 3D         lda    #L%256          pointer low
00610   0331   CD 06BD       jsr    fintp           interpret
...
0061x   0334   1E B9         bset   ACK,MODE        acknowledge
0061x   0336   81                    rts

* Formula = K1 * SQRT(K2- (ADC 2)^2)
00619   033D   B9       L    fc     fbr,fsyn        fpsyn creates a 'snap-shot' of
                                                    all 8 channels by copying the
                                                    readings from ADCR at $0070 to
                                                    ADCC at $0088
00620   033E   E2            fc     fadc,2          fpadc accesses current copy
00621   033F   C5            fc     ffun,fsqr       fpsqr multiplies FP1 by itself
00622   0340   C2            fc     ffun,fneg       fpneg negated FP1
00623   0341   67            fc     fld2,fadd       FP2=K2, then FP1=FP1+K2
00624   0342   82            fcb    $82
00625   0343   C4            fc     ffun,fsqt       fpsqt finds sqare root of FP1
                                                    by iteration. see [4.7]
00626   0344   69            fc     fld2,fmul       FP2=K1, then FP1=FP1*K1
00627   0345   81            fcb    $81
00628   0346   85            fc     fbr,fexit       leave interpreter
```

Figure 4.5 - Example of using interpreter

Chapter 5 Slave PC Software

The software in the PC was written in Pascal to suit the Turbo Pascal compiler version 5.0 by Borland International [5.1]. Annex [4] contains a listing of the program named *cc1v15fp*, along with a supporting unit named *ccsup*. All of the main program was written specifically for this project but most of the supporting unit already existed as a result of previous personal work. Additional units to support interrupt-driven buffered asynchronous communications have also been used, and these were purchased from Blaise Computing Inc. [5.2]. Pascal was chosen in preference to other languages such as C or C++ for a number of reasons, including familiarity and subsequent readability and maintainability by others. In the descriptions that follow all line numbers refer to lines in listing *cc1v15fp*.

5.1 Main program

The purpose of the software is to accept messages sent from a host computer via channel COM3: and provide continuous close-loop control of two coil-drivers connected via channels COM1: and COM2:; where all channels use serial asynchronous communications. (Figure 2.6 shows the interconnection of equipment). The control involves interpreting messages from the host (computer) which represent requests for particular set-points, and sending the appropriate low-level commands to each coil-driver. This task is complicated by the need to allow keyboard entry of set-points at the slave PC, while maintaining closed-loop control. The set-points are the readings which should be obtained by the precision ADCs, and therefore represent the voltage across the sensing pins of the precision resistors. The currents in the coils are linearly related to these voltages, with a range of ± 71 mA. There is an additional requirement for calibration of the DAC's within each drive, which means that at any moment the slave PC may be in one of three modes of operation; *host*, *keyboard* or *calibration*.

A full understanding of the operation of the software may be deduced by studying the listing but it is helpful here to describe the operation of the main sections so as to appreciate better the overall operation. Referring to the listing in Figure 5.1, the program section called {main} begins with initialisation of general variables and then of variables specific to drive 1, drive 2 and the host. An attempt is then made to restore the previous set of calibration parameters and tables, but if the relevant files do not exist the mode is changed to force a calibration cycle, before switching back

```
1557      begin {main}
1558        general_initialise;
1559        initialise_drive(drive[1],1);
1560        initialise_drive(drive[2],2);
1561        initialise_host(host,3);
1562        restore_all_parameters;
1563        if drive[1].p.valid and drive[2].p.valid then
1564          set_source(host_pc)
1565        else
1566          set_source(calibrate);
1567        repeat
1568          keyboard_activity;
1569          case source of
1570            host_pc :  begin
1571                         get_host_command(host);
1572                         execute_host_command(host);
1573                       end;
1574            calibrate : begin
1575                          if (drive[1].cal.state=cal_finished) and
1576                             (drive[2].cal.state=cal_finished) then
1577                            begin
1578                              drive[1].p.valid := true;
1579                              drive[2].p.valid := true;
1580                              write_all_parameters;
1581                              set_source(host_pc);
1582                            end
1583                          else
1584                            calibrate_cycle
1585                        end;
1586          end;
1587          control_loop;
1588          inc(tries);
1589          { gotoxy(10,21); write(loop_count:6,reply_count:6,tries:8); }
1590          report_current_time;
1591        until done;
1592        clrscr;
1593        halt; {close files/coms via MyExit}
1594      end. {main}
```

Figure 5.1 - *Main* program listing

to host-mode. During normal operation the repeat..until structure at 1567-1591, loops through four procedures; *keyboard_activity*, *get_host_command*, *execute_host_command*, and *control_loop*. At no time does execution get 'stuck' in any of these procedures, and all activity appears to be concurrent. Within *keyboard_activity* at 1353-1434 the mode may be changed using key letters which start the words in the Options box at the lower left corner of the display; these are Abort, Calibrate, Exit, Host, Keyboard. Selecting C, H or K causes a call to *set_source* at 1323 which updates the screen where necessary and alters the value of various control variables to indicate the change of state. Wherever possible enumerated data-types have been used whose names reflect the intended state or activity. If a complete and valid message has arrived from the host computer, as determined by *get_host_command* at 813-850, this fact is flagged to *execute_host_command* via the variable *host.com.state*.

It is the function of *control_loop* at 1027-1100 to send commands to each drive which depend on the corresponding set-point, which itself may have been defined by either

host messages, local keyboard entry, or the calibration procedure. Use is made of the drive commands X, Y and Z which read the precision ADC, all channels of the monitor ADC, and set the levels of the DACs respectively. The behaviour of the control-loop is determined by the value of the variable named *loop_state* associated with each drive. Not all calls to this procedure result in commands being send or replies received; indeed the speed of operation of the computer is very much faster than the rate of transfer of characters via the serial link and the procedure may be called many times before a complete message is sent or received via the interrupt driven buffers. Timing is co-ordinated by using the real-time counter within the slave PC, which is accessed by the *ticks* function at 246-249. The value returned increments at a rate of about 18.2 per second. Constants whose names begin with *ticks_* at 82-84 determine various important intervals. Currently the values are such that the monitor ADC is accessed every 5 seconds; there is a time-out of 3 seconds for a reply from a drive and the control loop is started every second. As there is not an integer number of ticks per second the timings are approximate.

Also as part of the main program loop is a call to *report_current_time* which updates the displayed time once per second. This provides a visual indication to the user that the software is 'working' even if all other displayed parameters appear static. Considerable time was spent on the creation of clear and sensible displays that were of use during both the development phases and actual use in a commercial metrology laboratory, such as that at the National Physical Laboratory (NPL). The next section describes the screen presentations in more detail.

5.2 Screen presentations

When the PC is turned on, program *cc1v15fp* is run automatically via a call in the AUTOEXEC.BAT file of the PC, and the screen appears very similar to the photograph shown in Figure 5.2. The source is set to HOST PC and the set-point for each drive to 0.0 volts. The screen is divided into three main areas, showing the status of the drives, the options available and the date of the last calibration. Each area is now described in more detail, beginning with the status of each drive, in terms of one drive.

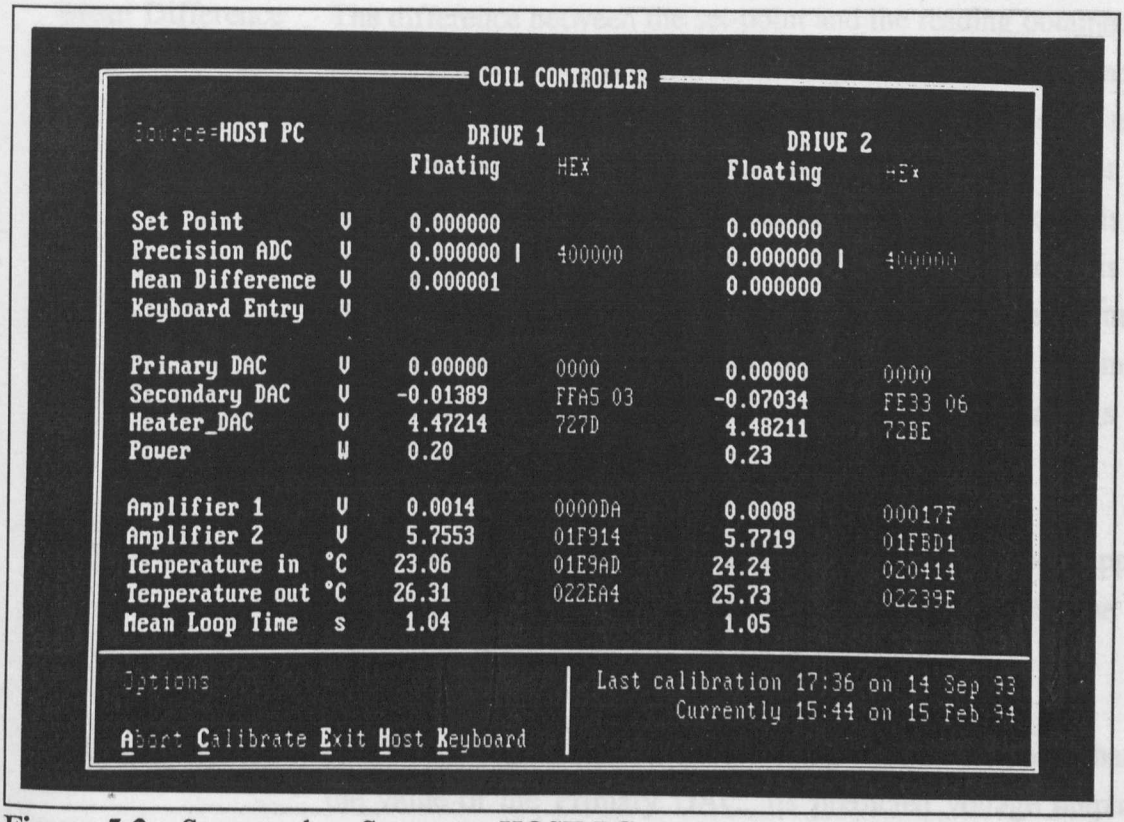


Figure 5.2 - Screen when Source = HOST PC

Set-Point

This represents the voltage that is to appear across the sensing wires of the precision resistor, as read by the precision ADC. It may be defined by a message from the host, or the keyboard, or the calibration procedure. The value should be between -5.0 and +5.0.

Precision ADC

This is the reading from the precision, 22-bit ADC, and should be very close to the set-point, except when the set-point is changed and there is a characteristic settling time. There is an oscillating bar to the right of the number to show when a new reading has been made, and this provides an indication that the control loop is operating in the event that the actual readings are stable.

Mean Difference — The difference between the set-point and the reading obtained by the precision ADC is used as the input to a moving window filter, and the effect is to display the average of the 12 most recent differences. Consequently when the value of either the primary or secondary DAC is altered the mean difference is not valid until 12 reading have been made. When the window is full the display is in white, otherwise it is grey. So long as the mean difference is less than 0.000005 in absolute value no automatic corrective action (trimming) takes place and the readings are continuously shown in white.

Keyboard Entry — When keyboard entry is selected by pressing K, numbers appear in the fourth row and may be edited by pressing keys on the keyboard. See section 5.6 for a full description.

Primary DAC — When the set-point is altered the control algorithm may change the value of the Primary DAC. Its predicted voltage output is shown as well as the associated Hexadecimal code.

Secondary DAC — When the set-point is altered the control algorithm will change the value of the Secondary DAC. Its predicted voltage output is shown as well as the associated Hexadecimal code. The contribution to the net output voltage is $1/64^{th}$ of the displayed voltage. Also shown are two hexadecimal characters which represents the value of a trim factor needed to compensate for changes since the most recent calibration.

Heater_DAC — This entry shows the voltage setting for the resistive heater if present. During the design it was thought prudent to include a way of operating the region near the coil at constant power, and therefore constant temperature. However there is no evidence that a heater is necessary and this reading may be ignored. If no heating element is present the plug in the heater output socket on the rear panel should contain a shorting link.

Power — This entry shows the total power dissipated by the coil and resistive heater if present (see Heater_DAC). It should always appear as the same value for a given coil resistance, which is defined during the calibration process.

Amplifier 1 — The voltage from the power amplifier associated with the coil output is always larger than the set-point because of the resistance of the coil, but if the coil is open circuit the amplifier will saturate at about ±13.5 volts.

Amplifier 2 — The voltage from the power amplifier associated with the heater output is always larger than the heater DAC value but if the heater is open circuit the amplifier will saturate at about ±13.5 volts. (see Heater_DAC)

Temperature In — A linear integrated temperature sensor is used to measure the temperature of the air which is taken into each drive unit near the inlet on the front panel. The absolute accuracy is only about ±1°C.

Temperature Out — A linear integrated temperature sensor is used to measure the temperature of the air as it is exhausted from each drive unit near the rear panel. The absolute accuracy is only about ±1°C. The exhaust temperature is higher than the inlet temperature because the air passes over the power amplifiers and the power supplies, and a difference of up to 3 degrees may be expected. More than 5 degrees is an indication that something is wrong.

Mean Loop Time — The control loop may alter the DAC values approximately once per second, and this entry shows the exact loop-time (1.04s). It provides and indication that the system all is functioning normally.

5.3 Calibration

When the operating environment is changed it is necessary to recalibrate the drives to ensure optimum performance. This should be done after the system has been

powered-up for 2 hours, and is simply initiated by pressing the letter C on the keyboard. After a warning prompt the screen appears similar to the one in Figure 5.3. This particular figure shows the screen for the very first calibration, and more entries are seen if there has been a previous calibration. It is necessary to enter (or edit) the values of the resistances for the coil and heater for both drives. If the heater does not exist its value should be irrelevant, but for the software control algorithm to function satisfactorily the heater resistance must be larger than the coil resistance, by about 10%. In fact without the heater, the whole concept of maintaining constant power dissipation near the coil is not possible and the resistance of the coil is not needed. However a non-zero entry of say 20, must be made for the software to function correctly. The entry of values in this way allows a heater to be added without additional modifications to the software. When all entries are correct, C should be pressed to continue the calibration process.

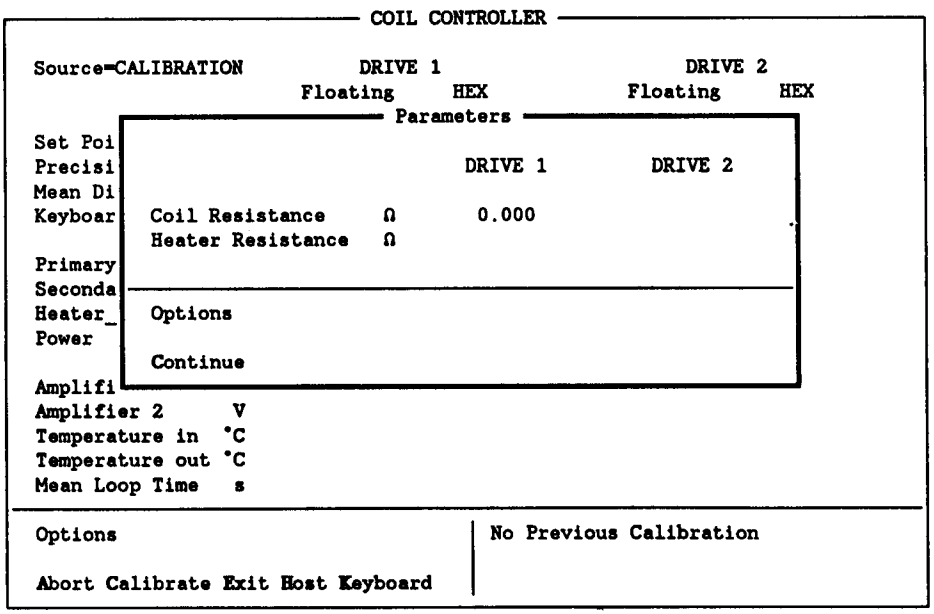


Figure 5.3 - Screen when entering CALIBRATION

IN SUMMARY

During the calibration process a number of set-points are defined and the response of the precision ADC is recorded for future use. Two files are created called *cc1v15f.db* and *cc1v15f.dt*, which represent the same data but the former is in a machine-readable binary format, whereas the latter appears as a printable text file. An abbreviated and rearranged example of a typical text file is appears in Figure 5.4. It shows a record of the voltages produced by the primary and secondary DACs across the sensing wires of the precision reference resistor, for a number of control codes, and for both drive units. From these readings a least-squares linear regression transfer function is

calculated. Since the calibration process is deterministic it is possible to calculate the time needed for completion, and this is added to the current time to provide an estimate of the time of completion. This time is shown at the bottom right-hand side of the display. During calibration both coils experience currents from minus full-range to plus full-range, at approximately the same time. Consequently the monolith will experience full translation but only modest twist, however if there is concern about any movement it may be removed prior to calibration.

Edited and annotated contents of the ascii data file cc1v15f.dt obtained during testing of the calibration techniques.

```
Drive= 1
Coil Resistance  =  22.000
Heater Resistance=  22.000
Primary DAC statistics
Temperature (in) = 23.9600
Number= 127
 Index  Code       Voltage       Error
   1  -32256.0  -4.92171597  -0.00004635
   2  -31744.0  -4.84356380  -0.00002003
..
  63    -512.0  -0.07792139  -0.00005528
  64       0.0   0.00018167  -0.00007808
..
 127   32256.0   4.92212224  -0.00006690
mean x = 0.0    mean_y = 0.00025976
a=  2.59756103275333E-0004
b=  1.52589576521978E-0004
r=  9.99999999924512E-0001

Secondary DAC statistics
Temperature (in) = 24.1393
Number= 63
   1  -31744.0  -0.07546997   0.00001644
   2  -30720.0  -0.07303190   0.00001282
..
  31   -1024.0  -0.00226164  -0.00002596
  32       0.0   0.00018215  -0.00002386
..
  63   31744.0   0.07591248   0.00001403
mean x = 0.0   mean_y = 0.00020602
a=  2.06016358875072E-0004
b=  2.38446421095551E-0006
r=  9.99999922838470E-0001
```

```
Drive= 2
Coil Resistance  =  22.000
Heater Resistance=  22.000
Primary DAC statistics
Temperature (in) = 24.7569
Number= 127
 Index  Code       Voltage       Error
   1  -32256.0  -4.92238855  -0.00004427
   2  -31744.0  -4.84419298   0.00000088
..
  63    -512.0  -0.07709622  -0.00007802
  64       0.0   0.00105834  -0.00007387
..
 127   32256.0   4.92461419   0.00000548
mean x = 0.0   mean_y = 0.00113221
a=  1.13221416323128E-0003
b=  1.52637540139278E-0004
r=  9.99999999858119E-0001

Secondary DAC statistics
Temperature (in) = 24.8809
Number= 63
   1  -31744.0  -0.07461846   0.00001557
   2  -30720.0  -0.07218027   0.00001155
..
  31   -1024.0  -0.00139141  -0.00002361
  32       0.0   0.00105715  -0.00001726
..
  63   31744.0   0.07679677   0.00001392
mean x= 0.0    mean_y = 0.00107441
a=  1.07441062018943E-0003
b=  2.38496859352594E-0006
r=  9.99999937323992E-0001
```

Figure 5.4 - Typical recorded calibration data

IN DETAIL

The general operation of the software is to interleave the control of both drives so quickly that they appear to be controlled concurrently. The following description is in terms of one drive but the strategy is applied to both drives simultaneously. When the calibration process is started *source* = *CALIBRATE* and *cal.state* = *cal_start*,

```
1484  begin {calibrate_cycle}
1485    for i := 1 to max_drive do
1486      with drive[i].cal do
1487      case state of
1488      cal_idle   :  ;
1489      cal_start  :
1490        begin
1491          set_stats(p_stats);
1492          state := cal_p;
1493          define_setpoint(drive[i],2);
1494          drive[i].s_trim := 0;
1495          if i=1 then
1496            display_finish_time;
1497        end;
1498      cal_p      :
1499        begin
1500          if drive[i].padc_filter.valid then
1501            begin
1502              enter_stats(p_stats,drive[i].p_code,
1503                          drive[i].padc_filter.mean);
1504              if p_stats.n=p_limit/2 then { record in middle of run }
1505                p_stats.temperature_in := drive[i].fp_temperature_in;
1506              if p_stats.n<=p_limit then
1507                define_setpoint(drive[i],1)
1508              else
1509                begin
1510                  apply_stats(p_stats);
1511                  set_stats(s_stats);
1512                  state := cal_s;
1513                  define_setpoint(drive[i],2);
1514                end;
1515            end;
1516        end;
1517      cal_s      :
1518        begin
1519          if drive[i].padc_filter.valid then
1520            begin
1521              enter_stats(s_stats,drive[i].s_code,
1522                          drive[i].padc_filter.mean);
1523              if s_stats.n=s_limit/2 then { record in middle of run }
1524                s_stats.temperature_in := drive[i].fp_temperature_in;
1525              if s_stats.n<=s_limit then
1526                define_setpoint(drive[i],1)
1527              else
1528                begin
1529                  apply_stats(s_stats);
1530                  state := cal_finished;
1531                  with drive[i] do
1532                    begin
1533                      drive[i].fp_setpoint := 0;
1534                      set_filter(padc_filter,fp_setpoint,
1535                                 precision_adc_filter_length,3);
1536                    end;
1537                end;
1537            end;
1539        end;
1540      cal_finished :  ;
1541      end;
1542  end; {calibrate_cycle}
```

Figure 5.5 - Coding for *calibrate-cycle*

which forces the main program loop to call repeatedly the procedure *calibrate_cycle* at 1484-1542. See Figure 5.5 for a partial listing and Annex [4] for a full listing. The behaviour of *set_dacs* at 984-1026 is modified so that calls from within *control_loop* 1027-1100, no longer force the DACs to track changes in the set-point. This has to

be so, since the purpose of the calibration is to create look-up tables which are used to permit the DACs to track the set-point. The first task of *calibrate-cycle* is to clear the variables associated with the impending linear regression analysis by a call to *set_stats* at 464-476, and then make *cal.state* = *cal_p* , which makes *set_dacs* calculate the setting of the primary DAC based solely on the set_point, with the secondary DAC set to zero. The set-point is altered by calls to *define_setpoint* at 1442-1458 so that the most significant 7 bits of the primary DAC are cycled through 127 possible values from 1 to 127. For each setting the average value of the precision ADC during a 20 second period is stored, and used to compute regression parameters. When all settings have been tried a call to *apply_stats* at 491-516 computes the coefficients of linear regression in the form $V_{padc} = a + b \times DAC_p$, where DAC_p varied from -32256 to +32256. Values calculated during one calibration cycle were $a=2.5975 \times 10^{-4}$, and $b=1.52589 \times 10^{-4}$, which represents an offset of 0.26mV and near ideal scaling of 5/32768=0.000152587. These coefficients provide confidence in the performance of the drive and are stored but not used by the closed-loop tracking algorithm, which uses the table of stored values.

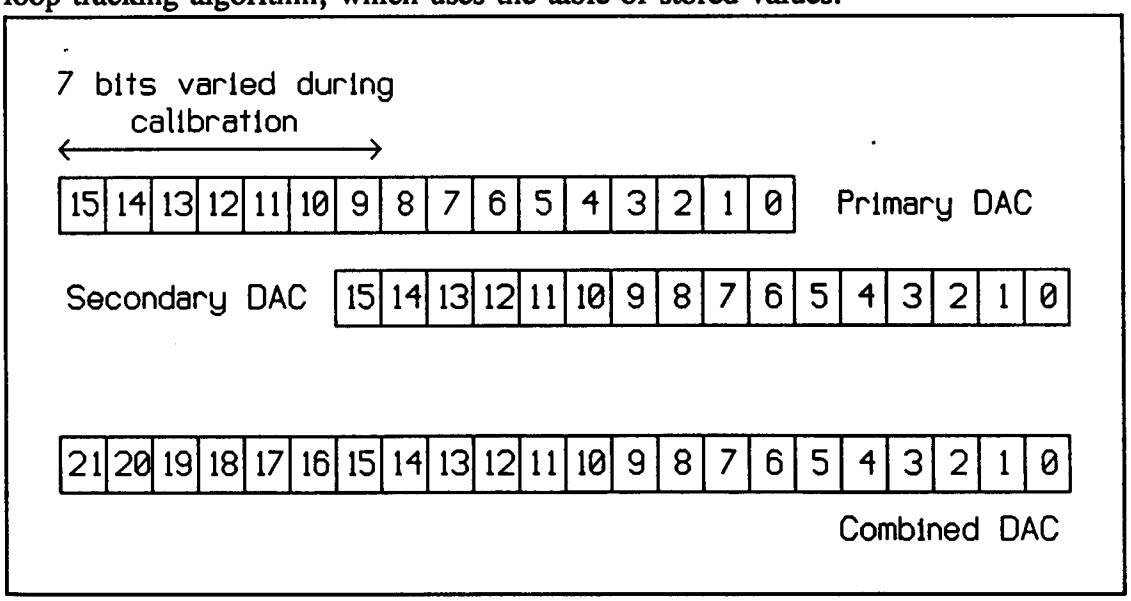


Figure 5.6 - Combined DACs during calibration

The behaviour of the secondary DAC is now determined when *cal.state* = *cal_s*, which makes *set_dacs* calculate the setting of the secondary DAC based solely on the set-point, with the primary DAC set to zero. The set-point is altered by calls to *define_setpoint* so that the most significant 6 bits of the secondary DAC are cycled through 63 possible values from 1 to 63. Once again averaged readings of the precision ADC are taken, stored and used to compute the effect on the net output voltage of the secondary DAC code again in the form $V_{padc} = a + b \times DAC_s$, where DAC_s varied from -31744 to +31744. Values calculated during one calibration cycle were $a=2.0601 \times 10^{-4}$, and $b=2.384464 \times 10^{-6}$ which represents an offset of 0.206mV

and scaling which is close to the theoretical value of 5/32768/64=2.384185 x 10^{-6}. The error was attributed to the tolerances of resistors R10 and R11. Coefficients a and b are stored and used in the form DAC_s = (Offset_voltage-a)/b by the closed-loop tracking algorithm in order to compute the control code for the secondary DAC.

When both drives enter the *cal_finished* state all readings and parameters derived by the calibration process are written to two data files, one in binary which is readily recovered the next time the Slave PC is turned on, and one in text form which may be printed and studied for interest.

5.4 Communication ports and interrupt driven buffers

Three serial communication ports are used by the slave PC, and each is associated with a interrupt driven handler with reserved memory areas for transmit and receive buffers. The use of buffers greatly simplifies the interface and improves operational speed, as the main procedures can initiate messages and gather responses without continuously monitoring the communication channels. The handlers were purchased from Grey Matter, Devon, England, but originally came from Blaise Computing in Berkley, California USA, and packaged under the name Asynch Plus. The associated manual [5.2] describes a number of levels of implementation and the highest level interface is used to create a file variable within Turbo Pascal that actually relates to a physical communications channel. Consequently all serial input and output is then performed via statements such as read(f,ch), where f has been previously assigned to a communications port via *__assignDD* at line 635 in the procedure *prepare_com_file* at 622-658. Also in this procedure the data rate is defined as 9600 baud, with no parity, 8 data-bits, 1 stop bit, and no hardware handshake. When the buffer is empty character code 26 is returned, so it is not possible to receive this character, which corresponds to ASCII control-Z. The input and output queue sizes are set to 1024 characters, which are unnecessarily large as the longest reply from a drive is only about 70 characters, but memory is not at a premium and there is room for expansion!

The assignment of channels is shown in Figure 5.7

Channel	PC I/O	PC IRQ	Connection
COM1	$3F8	4	Drive 1
COM2	$2F8	3	Drive 2
COM3	$3E8	5	Host PC

Figure 5.7 - Assignment of communication channels

5.5 Closed-loop control strategy

The program is written in such a way that messages are sent to, and replies received from, each of the two drives in an interleaved manner. Consequently both drives may be controlled as soon the messages are communicated via the serial links. State variables for each drive determine the action to be taken the next time a reply is received from a drive. Stated another way the program spends time repeatedly calling *control_loop* rather than being stuck within it.

With reference to the main part of the control loop at 1027-1110 it will be seen that the real-time clock within the PC is sampled, via the *ticks* function, at various parts of the control loop to ensure messages are sent at certain time intervals. Command X, which reads the moving average of the precision ADC, is issued every second, as is command Z, which updates the DACs. Command Y is issued less frequently every 5 seconds to obtain information from the auxiliary/monitor ADC about the drive voltages and temperatures. When data is received in response to the X and Y commands the display is updated with calls to *calc_and_disp_precision_adc* at 905-925, and *calc_and_disp_aux_adc* at 884-904 respectively. Each of these procedures uses utility functions to obtain floating point numbers from the different and obscure hexadecimal code structures. When a reply has been obtained from the X command, (delayed by data from command Y if the timing is appropriate), the reading is used by *set_dacs* at 984-1026 to compute revised values for the primary, secondary and heater DACs. The set-point is used in different ways depending whether the control strategy is to track the set-point automatically, calibrate the primary DAC, or calibrate the secondary DAC. In normal operation the set-point is tracked by setting the DACs to minimise the difference between the set-point and the mean reading of the precision ADC. This is achieved in a three stage process. Firstly the setting of the primary DAC which gives the nearest value is obtained from the table created during calibration; secondly the code for the secondary DAC is calculated using the inverse transfer function on the difference between the set-point and the nearest point; finally, a trim is added to compensate for errors in the previous two steps, and/or drift due to time and temperature. The trim called *s_trim* is calculated by *auto_trim* at 954-970, and incremented or decremented each time the filter for the precision ADC holds a valid mean and the absolute value of the difference between the set-point and the mean is greater than *max_error*, which is set at 81 to 0.000005, or 1 ppm.

5.6 Keyboard operation

When keyboard entry is selected by pressing K, the text in the top left-hand corner of the display will show Source=KEYBOARD and numbers appear in the row labelled Keyboard Entry. These may edited by pressing keys on the keyboard; acceptable keys being any of 0 to 9, decimal point, +, -, back-space, left arrow, right arrow or home. A cursor shows the current location of insertion of a digit and when the entry for drive 1 is complete, pressing carriage return (CR) causes the keyboard entry for drive 2 to be available for editing. When both are acceptable pressing CR causes both values to be copied to the set-point variables. If the difference is greater than 2.0, the values are not be copied as this would cause too great a twist to be placed on the monolith. If an entry is incorrect TAB or shift-TAB may be used to toggle between entries, without causing alteration of the set-points. Individual characters are accepted by the keyboard routine and all editing operations are performed by the program, and the cursor is not part of the operating system. This ensures that the control loop may operate during keyboard entry, with timing disrupted only by the time it takes to process an individual key-stroke.

When host control is required, H should be pressed, and when calibration is required, C should be pressed. If E is pressed the program may exit after a warning prompt, and return control to the operating system. If H is pressed, all characters in the input buffer are removed and the set-point is put at 0.0 for both drives before any new message is accepted from the host.

5.7 Messages from and to host computer

Characters from the host computer may arrive at any time and are buffered in an input queue until they are read during *get_host_command* at 813-850 called at 1571 in the main loop. They are copied into *hs.com.last_command* until the terminator character (ASCII 4) is received. The characters in the message are then tested to ensure they conform to a certain pattern and that all implied numbers have certain ranges, see 832-846. If the command is deemed valid, it is executed at 1304-1322 which involves copying the received set-point to the set-point of the identified drive, and sending the letter E back to the host as an acknowledgement. The unusual code structure is part of the commercial software running on the host computer and had to be accommodated. Unlike keyboard entry which traps for a difference between drive set-points which is larger than 2.0, values from the host are not tested or trapped.

5.8 Utility functions and procedures

A number of functions and procedures are used throughout the software but have not been described individually. These appear in the listing between lines 200-618, and are sufficiently short to be followed by reading the listing without additional comment here. In addition, use was made of a support unit which was already under development. In particular this unit, called *ccsup*, incorporates primitives which support the windows which appear for various prompts. Some of the line drawing routines have been used but there are several procedures which are not referenced. However the 'smart' linker within Turbo Pascal 5.0. ensures the executable file does not include any unnecessary code. References occur to a unit called *win* which was copied from Turbo Pascal 5.5's applications disc.

Chapter 6 Performance Measures

The performance of the system may be discussed in terms of macroscopic and microscopic behaviour. The former are concerned with such questions as 'Are messages passed correctly from the host PC to the slave PC ?' or 'Do the voltage regulators deliver ± 15 volts where expected ?'. These are relatively easy to answer by performing tests in software or making simple voltage measurements. Similarly it is easy to see that the cooling fan is rotating. However it is much harder to determine the microscopic behaviour because this involves the measurement of small currents and voltages, with a resolution-to-range which is below the limits of most common voltmeters. Of particular interest are the effects of electronic noise, and settling time after a step change in the set-point. The internal precision ADC guarantees a resolution to 22-bits or about 1 part in 4.5 million, and was used to measure dc conditions, but transient effects could not be measured to sufficient accuracy. Consequently extensive computer simulations were performed on the voltage-to-current converter stage, as its response dominates that of the digital-to-analogue converters. Much quantitative analysis was done during the design process and prior to design of the printed circuit board (PCB). The analysis provided evidence that a particular approach was feasible, and in many cases involved computer simulations of circuits.

At the beginning of the design process the goal was to provide a programmable current source capable of delivering ± 70mA to an inductive load with a nominal resistance of 20 ohms. The range was to be divided into 500,000 steps of equal spacing to achieve the resolution required by the X-ray interferometer. At an early stage it was decided to use Digital-to-Analogue converters with a range of ± 5 volts followed by a precision voltage-to-current converter. Using a particular circuit topology only one precision resistor was needed, across which 5v must be generated for 70 mA. The problem was thus transformed into the need to generate ± 5v with 500,000 intermediate steps, each of which corresponds to 20 μV. The actual design goal was to better this requirement and provide steps of 10μV with noise of no more than $\pm 5\mu$V pk-pk. Because of the electro-mechanical nature of the coil driver fixed to the silicon monolith there is mechanical damping, and electronic noise above a frequency of about 1 kHz is quickly attenuated. Mechanical resonances are known to exist below this frequency, and electronic noise may cause unwanted vibration of the monolith. Assuming the precision ADC provides a sufficiently accurate reading every second, then noise below about 0.5 Hz may be reduced/removed by the closed-loop control algorithm implemented in the slave PC. Therefore noise in the band from 1Hz

to 1kHz is of particular interest and importance, together with the behaviour of the circuits in this part of the frequency spectrum. Also of importance are the characteristics of the voltage-to-current converter at the chopping frequency of the chopper-stabilised amplifier within the analogue servo-loop, which is just below 10kHz. (All component references relate to circuit diagrams in Annex [1])

6.1 Noise sources and predicted performance

All electronic components exhibit random variations in their behaviour, due to a number of different phenomena. For a comprehensive introduction to the sources associated with resistors and operational amplifiers see [6.1].

The circuit may be divided in two parts as far as noise-analysis is concerned. Firstly from the voltage reference, via the DACs to the voltage-to-current converter (V-to-I), and then within the V-to-I. This separation is appropriate because the noise sources up to the V-to-I are additive, while those within the V-to-I, are removed or at least significantly reduced because of the closed servo-loop around U12, a chopper-stabilised amplifier.

6.1.1 Noise in voltage reference and DAC's

Noise from REF1 will pass through, and add to, noise in DACs U6 and U7, amplifiers U8 and U9, before reaching the passive filter at the input to the V-to-I. The noise of these sources is summarised in Figure 6.1

Component Number	Component Type	Noise μV pk-pk	Bandwidth Hz
REF1	AD586LQ	4	0.1 to 10
U6,U7	AD1145BG	5	0.1 to 10
U8,U10	TLC2654ACN	1.5	0 to 10
R9,R11	5k ohms	0.45	0 to 100
R10	320k	3.6	0 to 100
R17,R18	10k ohms	0.64	0 to 100

Noise for integrated circuit was obtained from manufacturers data sheets.
Noise for resistors was calculated using $E_R^2 = 4kTRdf$ integrated from 0.1 to 100. See [6.1]

Figure 6.1 - Component noise

The contribution of the thermal noise in the resistors is small compared with the noise from the voltage reference and the DAC. The worst case pk-pk deviation is the sum of the noise from each component, thus $4+5+1.5+1.5+0.45+0.45+0.64+0.64 = 14.18\mu V$. The contribution from DAC U7 is attenuated by the ratio of R11 to R10 (1:64), as is the contribution from R10 itself, and these terms have been neglected. In the $\pm 5v$ range this noise represents 1.4 ppm, which is slightly more than the 1ppm which was aimed for but is still with the specification of 1 part in 500,000 or 2 ppm. The figure may be unduly pessimistic as pk-pk values are usually quoted to ensure a 99% or 99.9% confidence by multiplying the rms value by 5 or 6.6 respectively [6.1]. Consequently much of the time the noise peaks will be lower than the quoted figures.

6.1.2 Response of Voltage-to-Current converter

The operation of the voltage-to-current converter was described in section 3.4, but little was said about the precise performance that could be expected in terms of stability and noise. To quantify the performance a number of computer aided simulations were carried out using PSPICE [6.2]. The relevant part of the circuit diagram titled 'dr1drv1.sch' was annotated with node numbers to allow entry into PSPICE as a netlist. The circuit diagram appears in Figure 6.2 and the corresponding PSPICE definition in Annex [5].

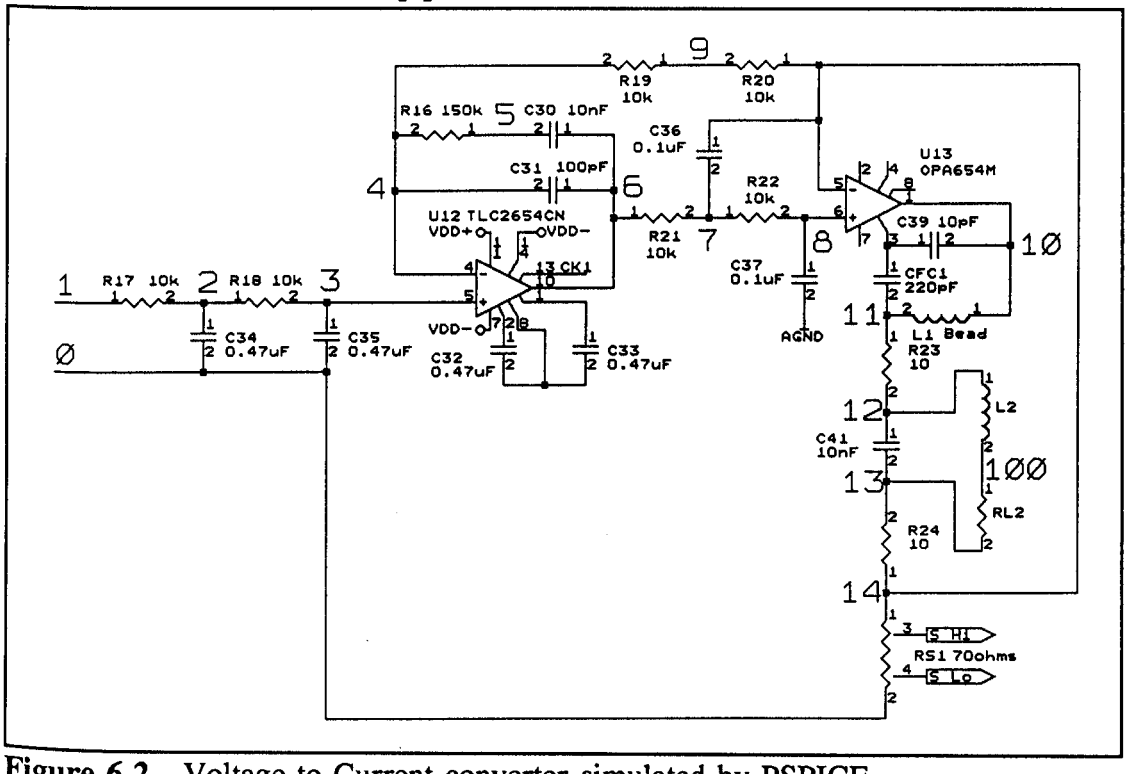


Figure 6.2 - Voltage-to-Current converter simulated by PSPICE

The integrated circuit amplifiers were modelled by voltage-controlled voltage sources, with appropriate gains, and consequently introduce none of the undesirable effects associated with the real amplifiers, such as input offset voltages. This is not an unreasonable approach because the chopper-stabilised amplifier has almost no offset effects and is the main component affecting the level to which the output current settles. The high frequency compensation components, C39 and CFC1, associated with U13 were not modelled as their function is to suppress radio frequency oscillations in the real power amplifier.

The change in the coil current when the input voltage changes from 0 to 1 volt at time t=0 is shown in Figure 6.3. The loop is settled when the voltage across the

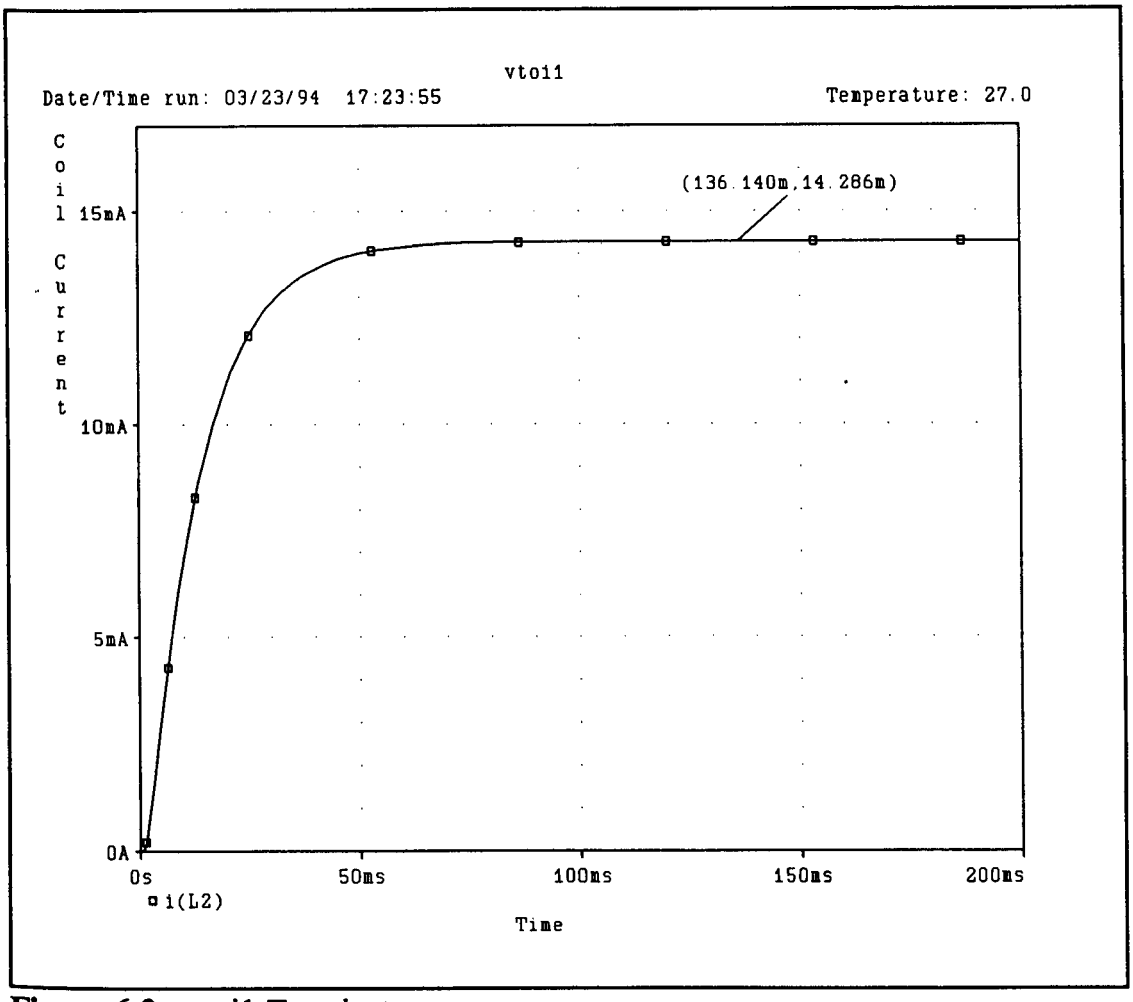


Figure 6.3 - vtoi1 Transient response

precision reference resistor, RS1, is also 1 volt. Since it has a value of 70 ohms (ignoring end effects as it is a 4 terminal device), the final current should be 1/70 = 14.2857 mA. According to the simulation this occurs at about 140 ms, however the results are rounded to the nearest micro-amp, whereas the minimum step in current is 0.14 μA (for 1 ppm). Considering that the current changes by 14 mA in 140 ms it is reasonable to assume that a further change of less than 1 μA will occur in less

than a few more milliseconds. Consequently, by inspection, an upper bound for the settling time due to a 1v step in the DACs is placed at 150 ms.

Figure 6.4 shows the gain of the circuit at various nodes. In particular the voltage at node 14 which corresponds with the voltage across the precision current-sensing resistor RS1, shows a progressive attenuation from 10 Hz to 400 Hz at 10dB/decade and thereafter at 80dB/decade. This suggests that noise above 1 kHz at the input,

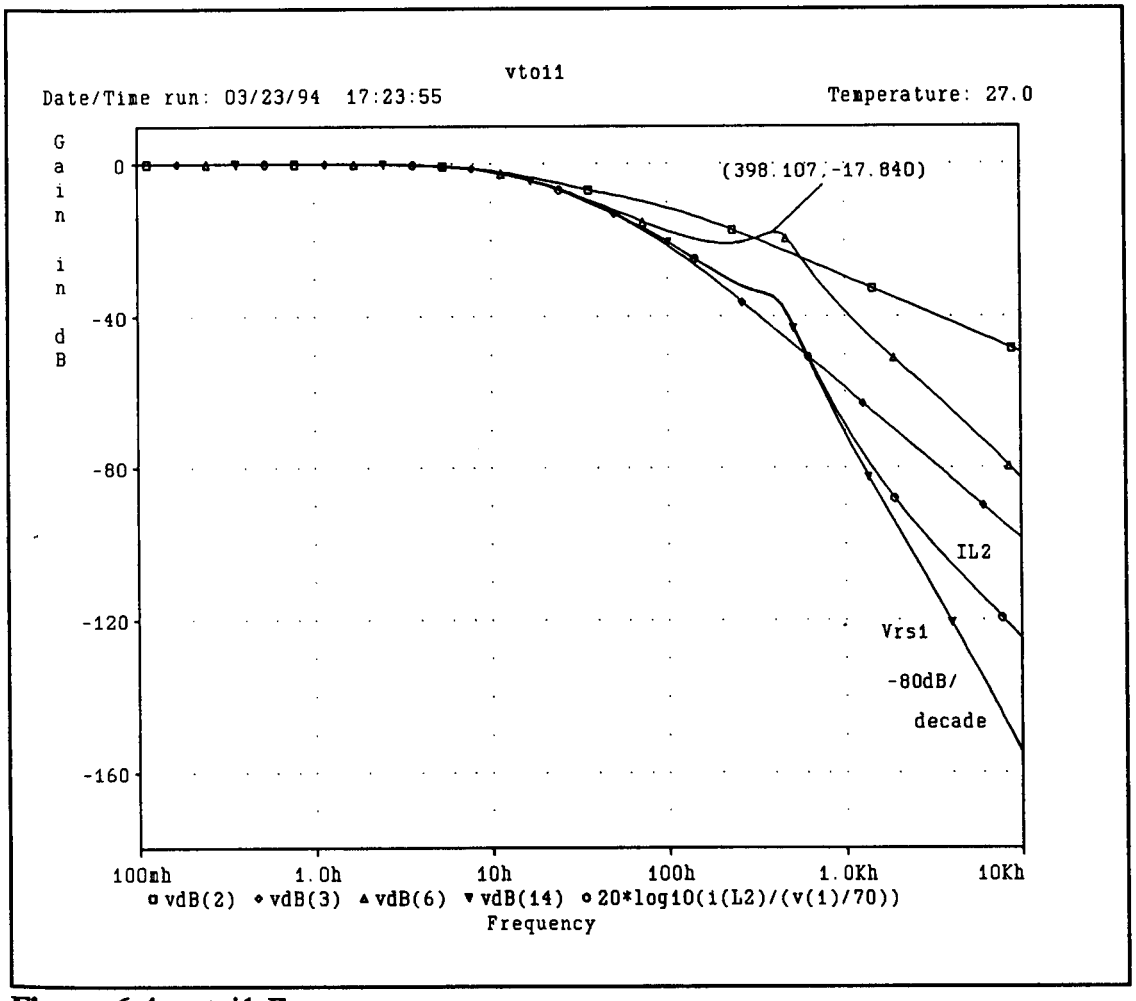


Figure 6.4 - vtoi1 Frequency response

which comes from the DACs in the real circuit, is of insignificant contribution to the output current. The trace labelled 'IL2' is included to show that the response of the current in the coil follows the same shape as the voltage across RS1 until about 1kHz when it does not fall as quickly. This occurs because the impedance of C36 falls and a more significant proportion of the current which should flow through RS1 flows through C36.

Using circuits similar to the one already described, the effects of noise in various parts of the design have been studied extensively. In particular further voltage sources were added to the circuit in Figure 6.2 to investigate the effects of noise at the input

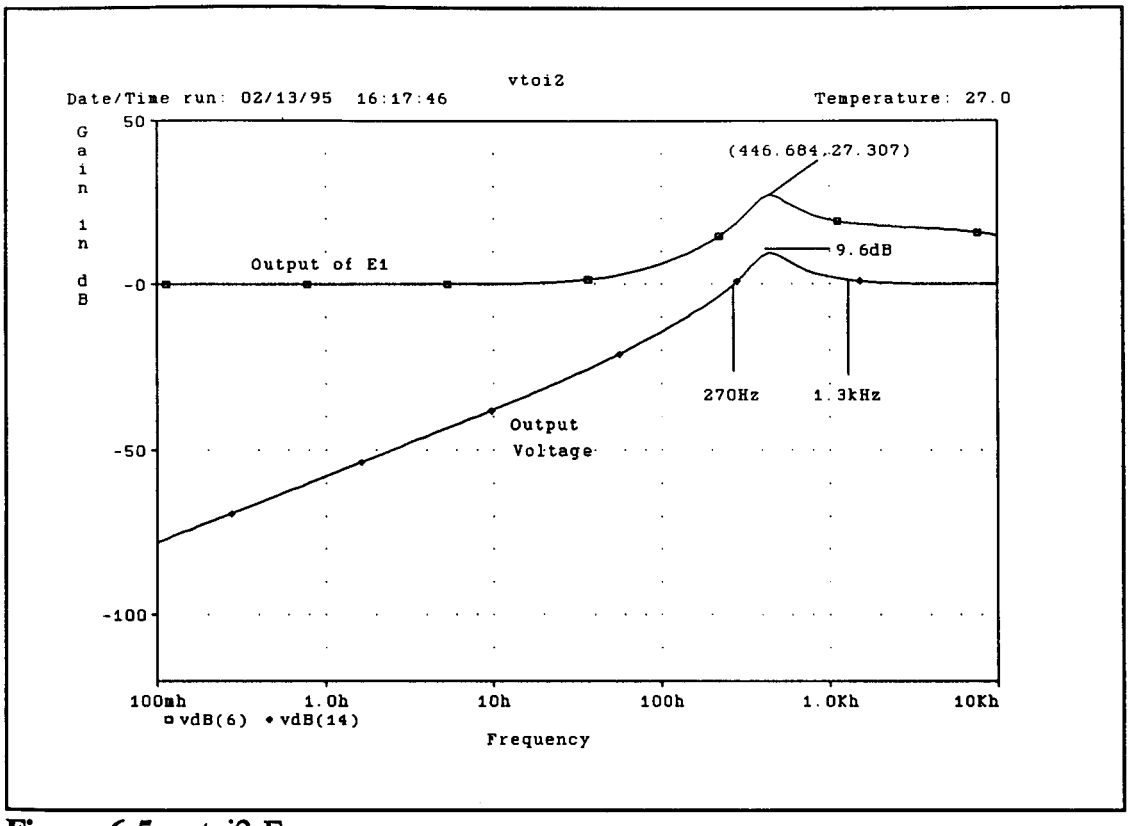


Figure 6.5 - vtoi2 Frequency response

of U13 and the output of U12. The resulting circuits appear in Annex [5] as vtoi2 and vto3 respectively. One of the graphs associated with vtoi2 is shown in Figure 6.5. It indicates that for frequencies below 100 Hz the servo action of U12, as modelled by E1, provides considerable attenuation, but noise generated at the input of amplifier U13, between 270 Hz and 1.3 kHz will be amplified with a peak gain of 9.6 dB at 446 Hz. This is potentially undesirable and may be significant depending on the noise in this band for the real device. The data sheet for the OPA654 [3.6] provides only a wide-band noise figure but does include a plot of input voltage noise spectral density versus frequency. Using this plot, which shows a progression from flicker-like noise at 0.01 Hz to white noise at about 1 kHz, it was possible to determine an explicit relationship between voltage noise spectral density (y) and frequency (x) for the region between 0.01 Hz and 100 Hz. This relationship was expected to be of the form $y=A/x^n$, and non-linear regression analysis shows there to be an excellent 'fit' when $A=393.7$ and $n=0.6309$. According to [6.1] many semiconductor devices exhibit noise with $n=0.5$ to 2, so $n=0.6309$ is not unexpected. For 1/f, or pink noise $n=1$. The rms value in a narrower region is given by the square-root of the definite integral of y^2 over that region. The noise in several bands was calculated and is shown in Figure 6.6

Frequency Hz			pk to pk (μV)=
from	to	rms (nV)	5xrms @ 99% confidence
0.01	0.10	946.4	4.73
0.10	1.00	700.0	3.50
1.00	10.00	517.8	2.58
10.00	100.00	383.0	1.91
100.00	1000.00	283.3	1.42

Figure 6.6 - Noise of OPA654

Between 1 kHz and 10 kHz white noise is dominant and the rms noise is given by 12 x 10^{-9} x $(10^4 - 10^3)^{0.5}$ = 1.14μV, which suggests a pk-pk value of 5.7μV with 99% confidence. When all these noise figures are considered in conjunction with the simulated gain vs frequency plot of vtoi2, there is no region where the noise in the coil will exceed about 1.1 ppm due to noise generated within the power amplifier, U13.

The real chopper-stabilised amplifier used for U12 has a very low input offset voltage of 5 μV, and an input voltage noise of 1.5 μV pk-pk from 0 Hz to 10 Hz. However it also has a very narrow-band noise output at its internal clocking frequency of 9.795 kHz. Consequently vtoi3 sets out to investigate the effects of signals added to the output of U12. The results show a flat response from dc to 100 Hz and an attenuation of about 70 dB at 10 kHz. This means that noise spikes of up to 15 mV at the chopping frequency will be attenuated to below 5 μV across RS1, and represent less than 1 ppm of full range.

6.1.3 Precision current sensing resistor RS1

The stability of the voltage-to-current converter relies upon the stability of the current-sensing resistor RS1. This is a power resistor made by Vishay Resistive Systems, and is capable of dissipating 10 watts. However in this circuit it dissipates a maximum of 0.35 Watts and is bolted to a heatsink which has a forced air flow over it. There is negligible local heating and the device remains essentially at ambient temperature. According to the data sheet for the resistor shown in Appendix [B], the maximum temperature coefficient at +25°C is $\pm$2.5 ppm, with a nominal of 0 ppm! Assuming the drives are operated in a precision metrology laboratory with a temperature stability of $\pm$0.1°C the drift of RS1 is negligible. Actually it is the resistance across the sensing elements that exhibits the quoted stability and end-effects may cause greater drift. Although the analogue servo-loop cannot remove the end

effects, the calibration procedure, together with the closed-loop tracking behaviour of the control algorithm, eliminates them.

6.2 Measured responses

The electrical behaviour of the drives was assessed by making measurements of various selected parameters under a number of operating conditions. The sections that follow describe these measurements in detail and show that the behaviour is as expected in all but one aspect; this being the existence of a (large) transient current in the coil when mains power is removed.

6.2.1 Precision ADC

The stability of the drive was first assessed by setting the digital control codes of the primary and secondary DACs to zero and recording the readings made by the precision ADC. Command Q in the embedded software outputs 20 sequential readings via the serial link in Hexadecimal notation. A typical list appears in Figure 6.7 and shows there is dc offset, which is made up from offsets in the DACs and precision ADC, as well as some variation due to noise. Also shown is the computed standard deviation of the readings from their mean value. Assuming the distribution to be Gaussian in nature, about 95% of readings may be expected to fall within 3 standard deviations, or ± 8.1. This means that 4 bits of the 22 may vary, and that if a particular reading is used in any simple control strategy the resolution may only be 1 part in 2^{18} (262144_{10}), which does not meet the required specification. Fortunately if a moving average is taken of 16 readings the variation in the mean value is much lower and may be used in the closed loop control strategy. The standard deviation of these moving averages is about 0.12 with a corresponding confidence interval of less than ± 1 bit. However this statistic is based on only 5 samples and therefore must be interpreted with care. The variation in readings comes from noise associated with all components including the precision ADC itself. Close inspection of the data sheet for this device reveals a specification of noise in the voltage reference, in the band from 0.01 Hz to 10 Hz, of 1 ppm pk-pk, or equivalent to ± 2 counts in 22 bits. After inspection of these results a moving average window filter was implemented in the embedded software, and the output of command X gives the average of the most recent 16 readings.

Reading in Hex	Decimal - Bias	Deviation	Moving Averages				
40005F	95	-2.8	95				
400062	98	.2	98	98			
400060	96	-1.8	96	96	96		
400062	98	.2	98	98	98	98	
40005F	95	-2.8	95	95	95	95	95
400060	96	-1.8	96	96	96	96	96
400064	100	2.2	100	100	100	100	100
40005F	95	-2.8	95	95	95	95	95
400066	102	4.2	102	102	102	102	102
40005E	94	-3.8	94	94	94	94	94
400064	100	2.2	100	100	100	100	100
400064	100	2.2	100	100	100	100	100
400061	97	-.8	97	97	97	97	97
400067	103	5.2	103	103	103	103	103
400060	96	-1.8	96	96	96	96	96
400064	100	2.2	100	100	100	100	100
400062	98	.2		98	98	98	98
400062	98	.2			98	98	98
400063	99	1.2				99	99
400060	96	-1.8					96
Mean	97.8		97.81	98.00	98.00	98.19	98.06
Std. Dev.		2.42					

Figure 6.7 - Precision ADC readings

Using a simple terminal emulation program on the slave PC with the ability to log received characters to a file, command Q was repeatedly executed and the readings of the precision ADC recorded, with the primary DAC set to half range. A list of hexadecimal characters was obtained that was changed into scaled floating-point format before being imported to a 'spread-sheet' package [6.3] for analysis. Figure 6.8 shows a polynomial fit to the readings and the moving mean of the most recent 16 readings, together with the expected output of the X command, which is truncated to ± 21 bits. The vertical axis is biased by the mean of all the readings which is 2.5198245, and expressed in parts per million. The deviation of the truncated mean is within ± 0.5 ppm for the duration of these readings but extended monitoring of the readings provided by the X command during normal operation over tens of minutes shows the peak deviation is ± 3 counts in 22 bits or about 1.5 ppm. Software in the slave PC averages these readings so as to obtain an even better estimate of the mean voltage across the precision resistor. It is this estimate that is used by the closed-loop control algorithm. Figure 6.9 shows the spectrum of the low frequency noise with the control algorithm functioning.

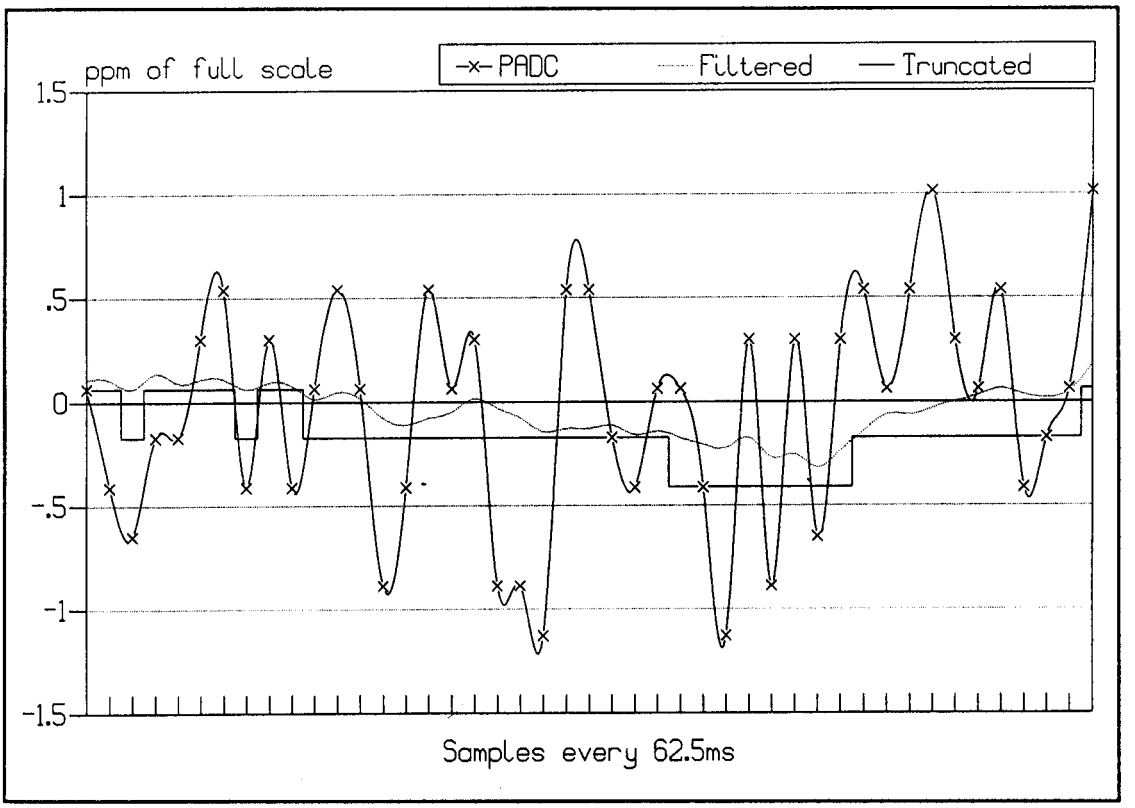


Figure 6.8 - Precision ADC readings and filtered values

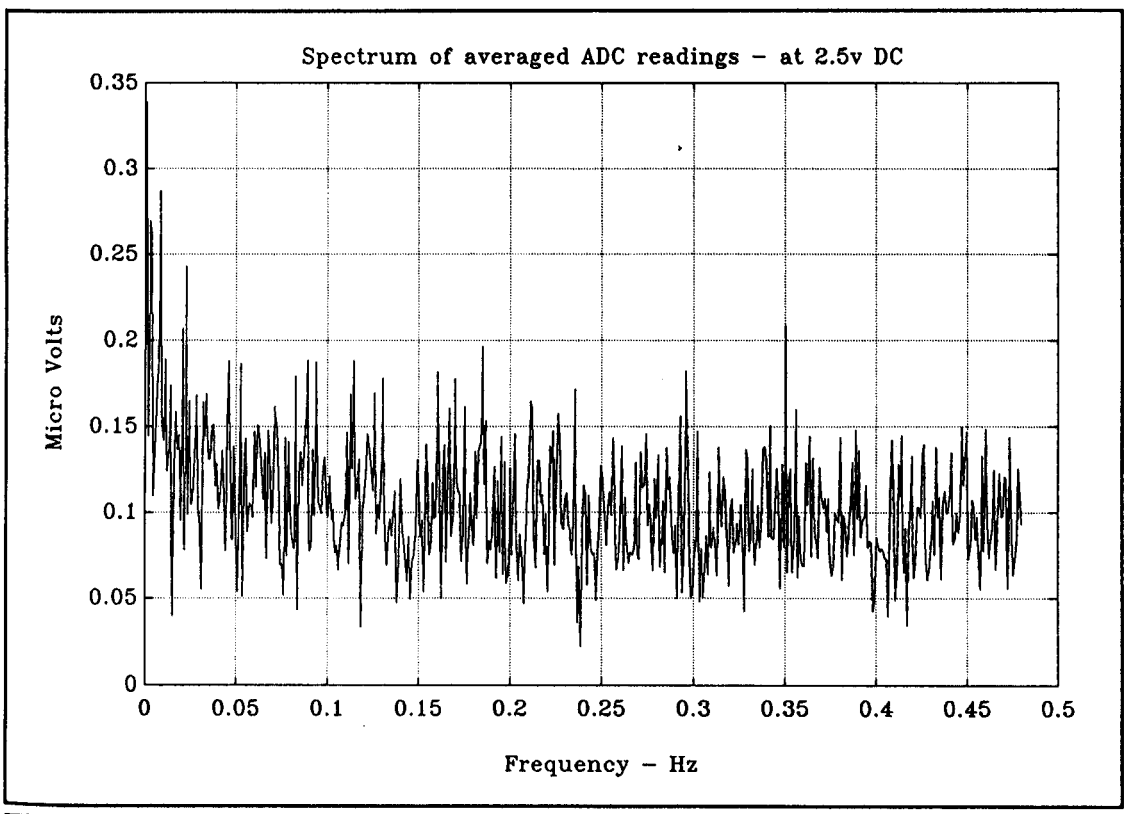


Figure 6.9 - Low frequency noise of filtered ADC readings

6.2.2 Power-on/off transients

The voltage-to-current converter circuit contains resistors R34 and R35 which protect the output amplifier from a short-circuit coil or accidental connection of the coil output to analogue ground. (see circuit diagram 5 in Annex [1]). Consequently the voltage at the connector on the rear panel corresponding to pin 2 of JP7 on the circuit is larger than the control voltage by a factor of (10+70)/70. Using a data acquisition system [6.5], samples of the output voltage with respect to analogue ground were taken every 1 ms for about 2.5 seconds for both coils. During this time mains power was applied and then removed from both drives. Figure 6.10 shows the output expressed as a percentage of full scale, accounting for the voltage drop across R35.

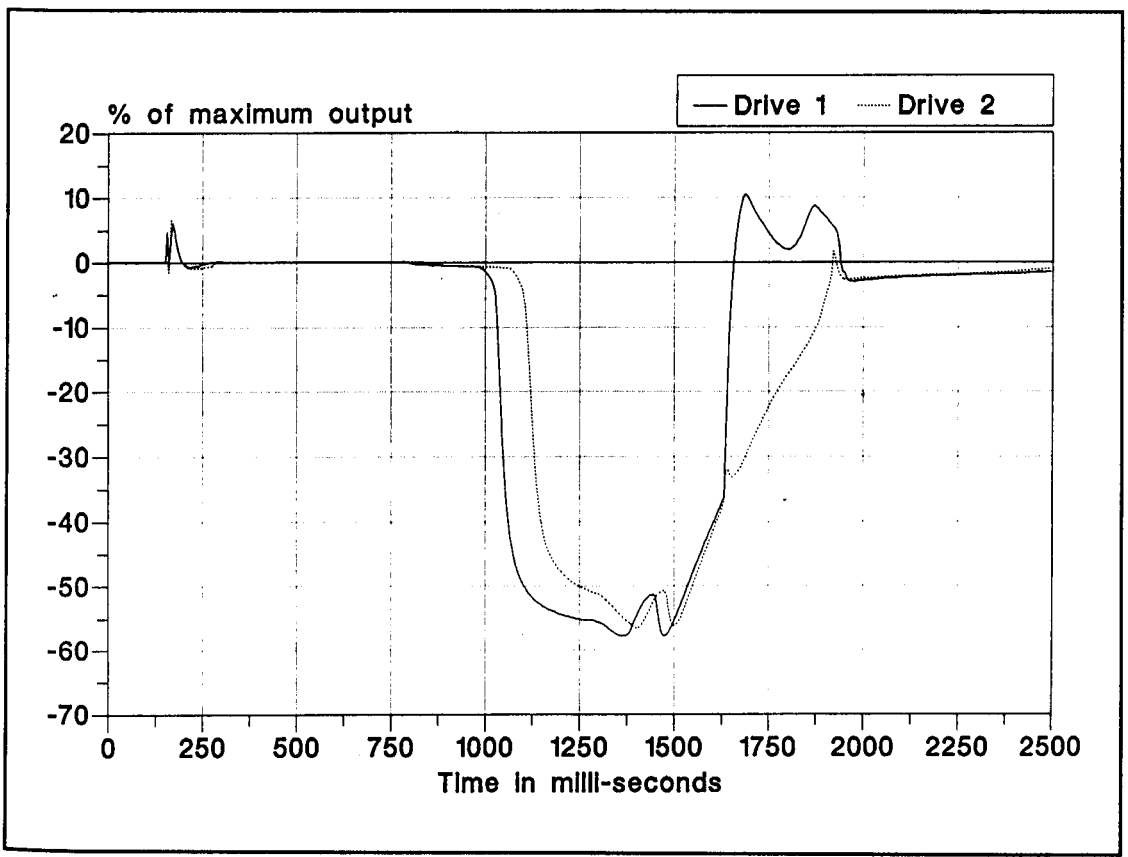


Figure 6.10 - Power on and off transients

Both drives have similar outputs, showing a small transient when power is applied at time=130ms, but much larger transients when power is removed at about time=750ms. Care was taken when selecting the DACs that they power-on at 0volts so the relatively small power-on transient was expected, but no consideration was given to power-off conditions. The transients are no larger than could be applied under program control and from this point of view they are acceptable, however if the transients had been in opposite directions a torque would be applied to the monolith which is far larger than any desired under program control. In hindsight

additional circuitry should have been included to prevent the power-off transient.

6.2.3 Step response

Using the keyboard entry option of the controlling software running on the slave PC the set-point for both drives was changed from 0v to +4v, representing a step of 80% of +full scale output. Monitoring the output voltage as described in 6.2.2, allowed the plot shown in Figure 6.11 to be produced. It shows the output of both drives settling to 4*80/70=4.57v in about 100 ms, and that there is excellent agreement between the simulated and actual responses; indeed two curves lay almost on top of each other. The output of drive 1 lags drive 2 because of the time taken to communicate messages serially from/to the slave PC where the interrupt structure gives priority to drive 2. Specifically drive 2 is on channel COM2 with interrupt request level 3, while drive 1 is on channel COM1 with interrupt request level 4, and level 3 has a higher priority than 4 [6.6, 6.7]. The delay results in an undesirable short-lived twist of the monolith which could reduce contrast of X-Ray fringes, but this has never been noticed.

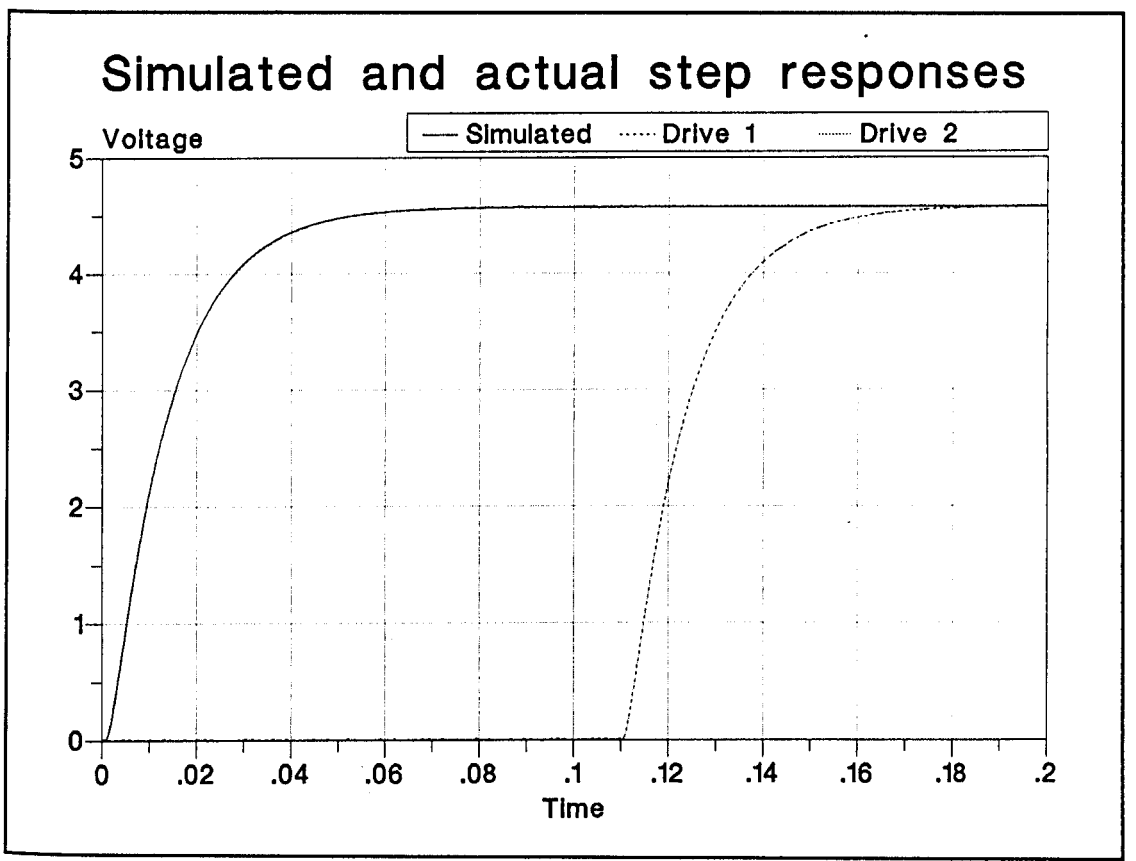


Figure 6.11 - Step response

6.2.4 Noise in the coil

It is to be expected that any silicon monolith will have mechanical resonances in the range of a few hertz to a few kilohertz [6.8]. Consequently extensive effort was made to determine the current noise in the coil in this region. For frequencies up to about 1kHz the voltage on one side of the coil with respect to ground is closely related to the current flowing through it, as shown in Figure 6.4 and associated text. However for higher frequencies this is not true and another means of measuring the current through the coil is needed. It was decided to place a 10 Ω resistor in series with the coil and monitor the voltage across it, and hence infer the current. Although a higher value of resistance would give a larger voltage drop, which in turn would be easier to measure, the value of 10 Ω was used so that it did not alter the normal operating conditions too much. An ac-coupled differential amplifier with high input impedance, low-noise and high-gain was needed to amplify the voltage across the resistor so that signals could be read via the sampled data acquisition system. Unfortunately no suitable amplifier was available in the laboratory and it was necessary to build one. In section 6.1.1 the predicted peak-peak noise across the current sensing resistor was about 15 μV. This corresponds to a current noise of 15/70 μA, and a voltage of 15/70 x 10 μV across the 10Ω resistor. Using a gain of about 500,000 produced signals which could be measured by the data acquisition system, and yet still allowed signals up to 5 times larger to be captured before causing an over-range condition.

The circuit diagram appears Figure 6.12. and shows 3 amplifiers in cascade. The first is an instrumentation amplifier with high impedance differential inputs, a single-ended output and a gain set at 100 by the particular choice of connections. The next two are low-noise operational amplifiers connected with voltage gains of 101 and 46.5 respectively as determined by fixed resistors. The combined gain is 100 x 101 x 46.5 = 469000. It was built on a square pad prototyping circuit board and operated by two 9 V batteries inside a aluminium box.

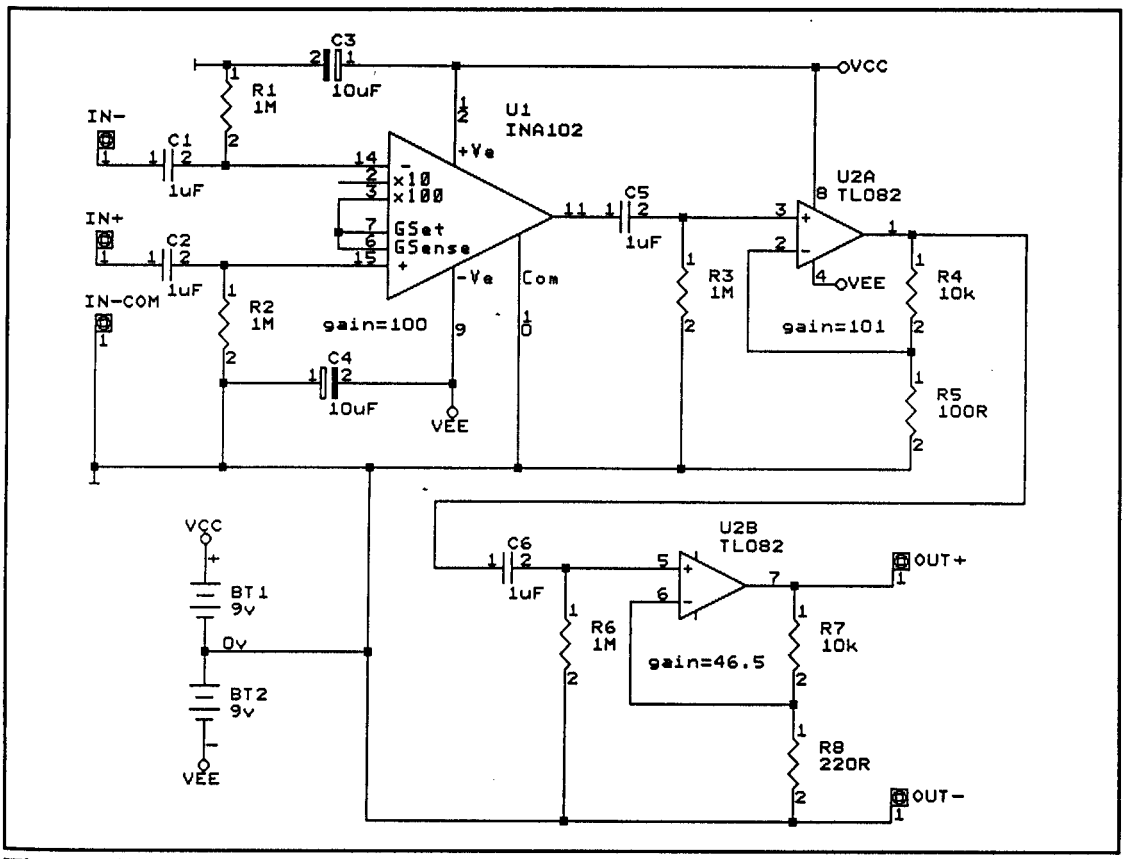


Figure 6.12 - High gain test amplifier

The output was sampled every 100μs for 7 x 1024 readings and then analyzed using MATLAB [6.4]. The sampling interval of 100μs was chosen because it enabled frequencies up to 5 kHz to be studied, and 7 blocks of 1024 samples allowed averaging of the output of successive Fast Fourier Transforms, while keeping within the matrix limits imposed by the PC version of MATLAB. In order to present the spectra in terms of parts per million FSO a correction factor is needed, and this was calculated as follows.

R is a 10Ω *resistor in series with the coil.*

Let V_R=*voltage across R,* V_a=*amplified voltage,*
g=*gain*=469000, I_c=*coil current*

$$V_R = I_c R \quad \therefore \quad I_c = \frac{V_R}{10} = \frac{V_a}{10g} \quad \text{also} \quad +I_{max} = \frac{5}{70} \Rightarrow I_{range} = 2 \times \frac{5}{70}$$

$$\therefore \text{fraction} = \frac{I_c}{I_{range}} = \frac{\frac{V_a}{10g}}{2 \times \frac{5}{70}} = \frac{V_a}{10g} \times \frac{70}{5 \times 2} = 0.7 \frac{V_a}{g}$$

$$\therefore ppm = 0.7 \times \frac{10^6}{469000} \times V_a$$

Averaging proved necessary because of the intrinsic noise in the amplifier which is shown in Figure 6.13a) and the low level of the signals of interest. The first few tests resulted in a spectrum with an unexpected peak of about 2 ppm at 351 Hz, see Figure 6.13b), but further investigation showed this was due to a temporary connecting cable with a poor shield being too close to the brushless cooling fan, which creates a rotating magnetic field. Using the cable specified for connection to

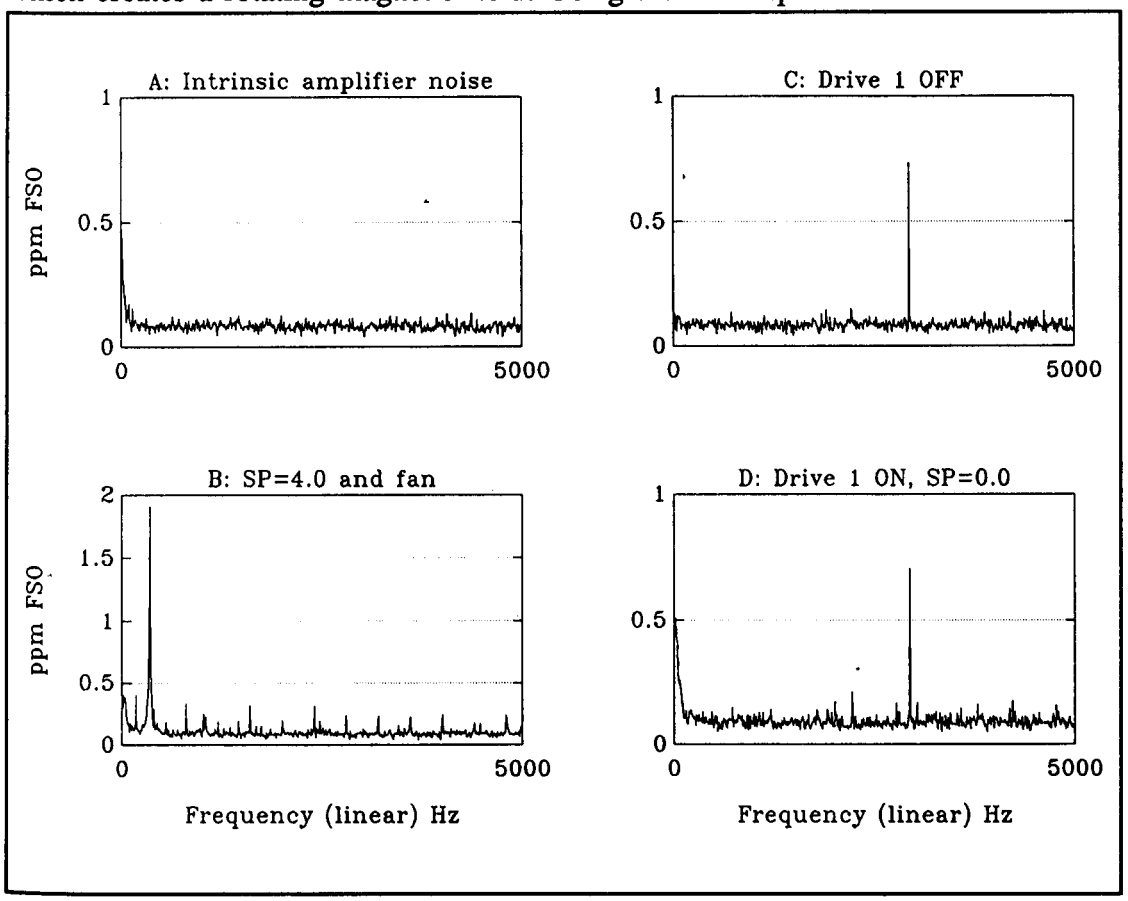


Figure 6.13 - Noise spectra of coil-current

the coil with a woven braid screen lapped with foil, and routed away from the fan this peak was no longer observed in the spectrum of either drive. It was possible to determine without doubt that the fan was the source of the interference by covering the air outlet and seeing a shift in the frequency of the peak as the fan partially stalls due to the impeded airflow. Spectra of the current in the coil for drive 1 appear Figure 6.13c) and Figure 6.13d). The peak at 2939 Hz is present even if the mains power to the drive is off, but is not present when the differential inputs of the amplifier are shorted together and to ground. The graphs for drive 1 and drive 2 are almost identical. This leads to the conclusion that the peak is due to the cabling between the amplifier and the coils, which was common during measurements of each drive. The peak is undesirable but of little practical significance because its amplitude is less than 1 ppm at a frequency above natural resonances.

Figure 6.14 shows the MATLAB script files that were used to create the plots in Figure 6.13.

```
file spec3.m
function z = spec3(x)
n=1024;                          % block size
s=zeros(n,1);
m=fix(max(size(x))/n);           % number of blocks
for i=1:m
  y=fft(x((i-1)*n+1:i*n));
  s=s+abs(y);                    % accumulate in s (sum)
end;
f=10000*(0:n-1)/n;               % actual frequencies with 100μs samples
z=[f',s./m/(n/2)];

file pnoise4.m
% Load files and compute spectra
load pcdni1 ; a=spec3(pcdni1);  clear pcdni1;  % amplifier noise
load pcdlnl6; b=spec3(pcdlnl6); clear pcdlnl6; % mains off
load pcdlnl7; c=spec3(pcdlnl7); clear pcdlnl7; % SP=0
load pcdlna1; d=spec3(pcdlna1); clear pcdlna1; % SP=4 and lead near %
                                fan
% remove dc component and scale to ppm of Full Scale Output (FSO)
g=0.7*1000000/469000; % scaling factor see section 6.2.5
f=a([2:512],1);   % frequencies
a=a([2:512],2)*g; % intrinsic noise
b=b([2:512],2)*g; % drive with mains power off
c=c([2:512],2)*g; % drive with mains power on SP=0.0
d=d([2:512],2)*g; % drive with mains power on SP=4.0
%                   and lead near fan
% Plot
clg;
subplot(221); axis([0,5000,0,1]); grid;
ylabel('ppm FSO'); title('1: Intrinsic amplifier noise'); plot(f,a,'w-');
subplot(222); grid;
title('3: Drive 1 OFF'); plot(f,b,'w-');
subplot(224); grid;
xlabel('Frequency(linear) Hz)'); title('4: Drive 1 ON, SP=0.0'); plot(f,c,'w-');
subplot(223); axis([0,5000,0,2]); grid;
xlabel('Frequency(linear) Hz'); ylabel('ppm FSO'); title('2: SP=4.0 and fan');
plot(f,d,'w-');
```

Figure 6.14 - MATLAB script files for spectral analysis

The frequency response of the high-gain test-amplifier was obtained with an Advance Spectrum Analyzer model R9211C, with an external resistive attenuator of 10/820k. Various plots are shown in Figure 6.15. The net gain was 469000x10/820000=5.72 or about 15 dB. Intrinsic noise means a smooth trace may not be obtained, even though the coherence function is nearly 1. The plots show the amplifier to have an almost flat response from 0.25 Hz to 2 kHz. At 5 kHz the gain is reduced by a factor of about 2, which means that although the trends in Figure 6.13 appear to be flat to 5 kHz, the original signal must rise and be complementary to the reduction of intrinsic gain. Even taking this effect into account the amplitude of noise in the coil is still only about 0.2 ppm at 5 kHz.

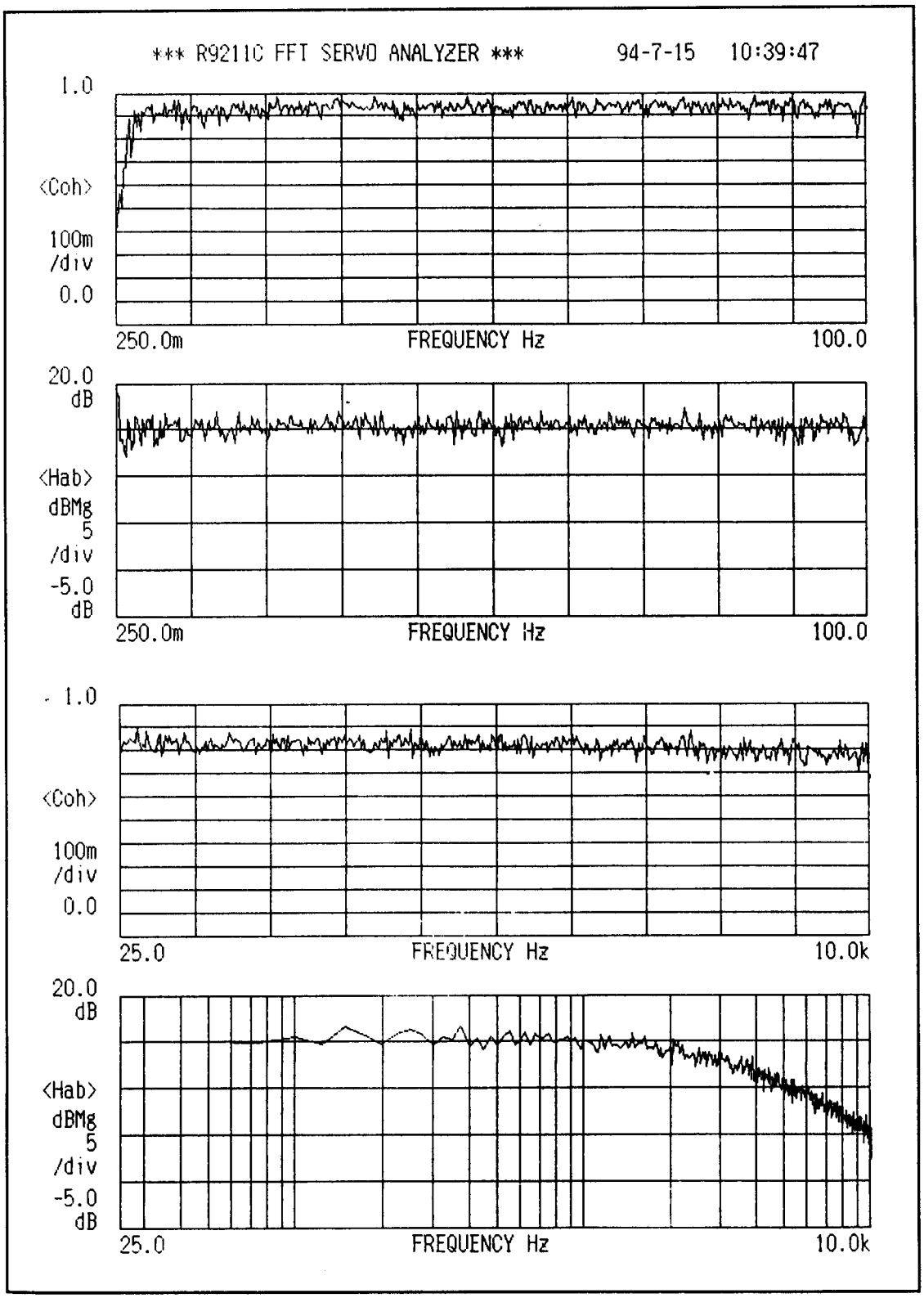


Figure 6.15 - Frequency response of test amplifier

6.2.5 Closed-loop response

When a particular set-point is defined, either by direct keyboard entry or by a message from the host computer, the control algorithm determines the most

appropriate value for the primary DAC by using a look-up table, and computes the necessary value for the secondary DAC. Unfortunately due to variations induced by fluctuations in temperature in the laboratory since the time of calibration, a large change in the set-point requires a small automatic trim of the secondary DAC. (For a full description of these operations see section 5.5). Also when the set-point crosses a boundary such that the primary DAC must be changed as well as the secondary DAC, it may be necessary to trim the secondary DAC again.

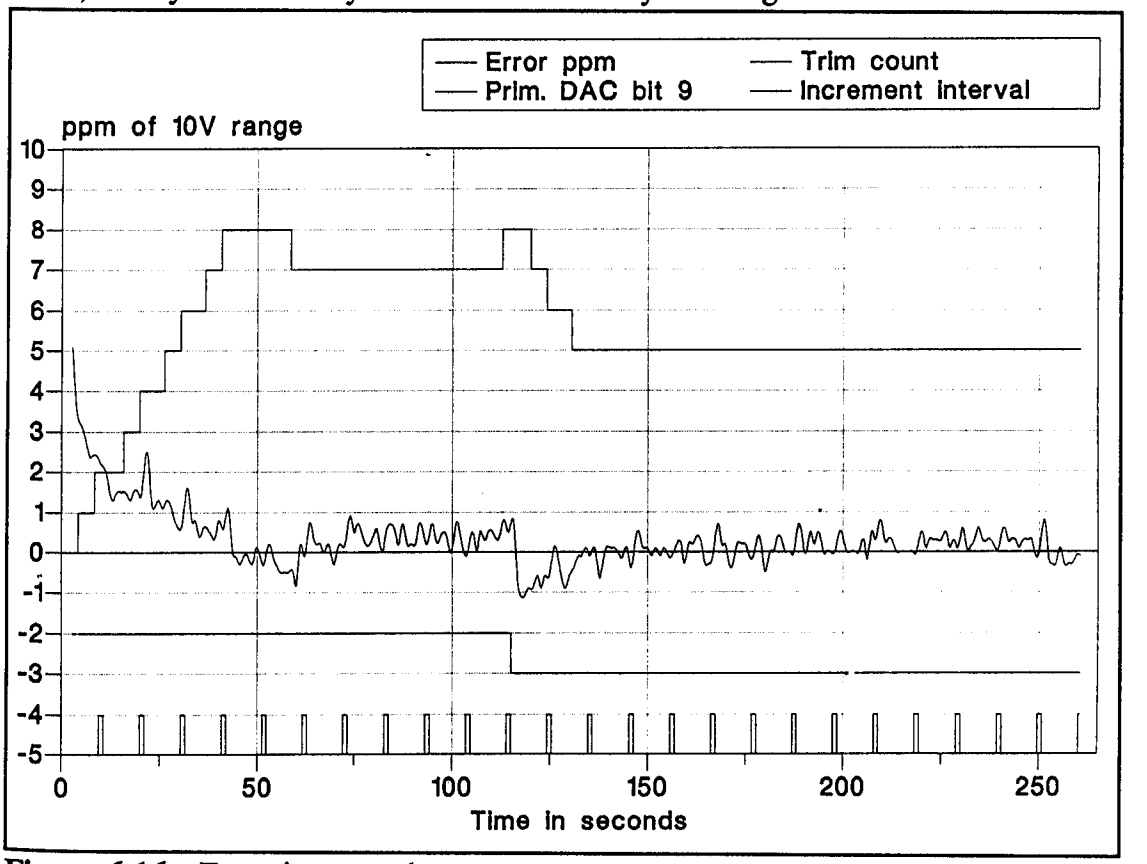


Figure 6.16 - Error in ramped output

The control software was modified to include an additional feature not accessible to the normal user which caused the set-point to be incremented from a given starting value in defined steps every 10 seconds. Figure 6.16 shows the error in the voltage read by the precision ADC and the trim value in counts. The initial set-point was chosen as 3.398665 volts so that the trim had time to settle before the primary DAC was changed; and the step size was 0.000005 V or 0.5 ppm of full scale output. Whenever the mean of the readings obtained from the precision ADC in the preceding 4 seconds is outside ± 0.5 ppm the trim value is altered by 1 in the direction appropriate to reduce the error. During the first 40 seconds the trim value is increased at a regular rate until the error is less than ± 0.5 ppm and then there are no further changes until time=115 s when the primary DAC changes from 22016 to 22528, and the secondary DAC from 16395 to -16377. The value predicted for the secondary DAC is not quite correct and a change in the trim value is required. This change is -

3 counts in 22 bits, which moves the error by about 0.5 ppm. The graph shows clearly that for these level changes, the drive output can track changes in the set-point as small as 0.5 ppm even when the secondary DAC experiences a major transition, but there is no reason to suppose that it will not function in a similar way for all levels. The behaviour of both drives is similar but the levels at which major changes of the secondary DACs occur are different, because of differences in components, and particularly offset voltages.

6.3 X-Ray interferometer fringes

All of the previous performance measures showed the drives to perform as desired, and within design requirements, but the greatest test was to use both drives together to control the position of a silicon monolith on an X-Ray interferometer. The slave PC and the drives are regarded as a subsystem by the host computer, which sends messages via a serial link to define the required operating voltages and hence currents. This host computer runs a commercial program that treats the two coils as 'linked-drives' and the operator can specify offset and gain factors for each of the drives to compensate for differences such as the strength of the permanent magnets, coil geometries and absolute or relative positioning. Once this has been done the coils may be considered as 'identical' and the user can work with 'equal' steps. A scan then consists of a ramp of equal step increments in sequence, and during each step the following occurs: 1) Alter drives, 2) pause for mechanical settling - typically 1 s, 3) photon count X-rays to get fringe intensity - 1 to 5 s. Overall rates are usually 1-10 seconds per step.

Tests on a large monolith, where the overall sensitivity of the drives was about 100nm/mA, show that contrast can be maintained across at least a ± 4.5 μm scan range without adjusting the twist compensation setting. This is excellent evidence of the drive stability and linearity as the two independent drives must be tracking to a very high degree.

The graphs in Figure 6.17 to Figure 6.20 show the fringe pattern under a number of different operating conditions. In each case one fringe corresponds to the <1 1 1> Si lattice parameter, 0.316..nm. The inference from these graphs (and others not presented) is that the drives provide stable control for scan speeds as low as 0.3nm/hour for a stiff monolith, and 1.4nm/hour for a large monolith. Mechanical resonances of the monolith can degrade or even destroy the contrast if the amplitude

approaches that of half the fringe-width. Resonances tend to be in the region of a few hundred hertz, and perhaps as low as 50 Hz for the lowest mode in the biggest monoliths. Figure 6.18 shows a noise spike on the basic fringe shape while traversing in one direction, which is attributed to a temporary loss of contrast induced by excessive external noise, including traffic and construction work. Noise is always present on the fringe pattern as the counting of x-rays follows a Poisson distribution. With 4000 counts per second (cps), a deviation of approximately $\pm$ 65 cps on a 1 s count can be expected. In no respect did the behaviour of the drives limit the performance of the X-Ray interferometer.

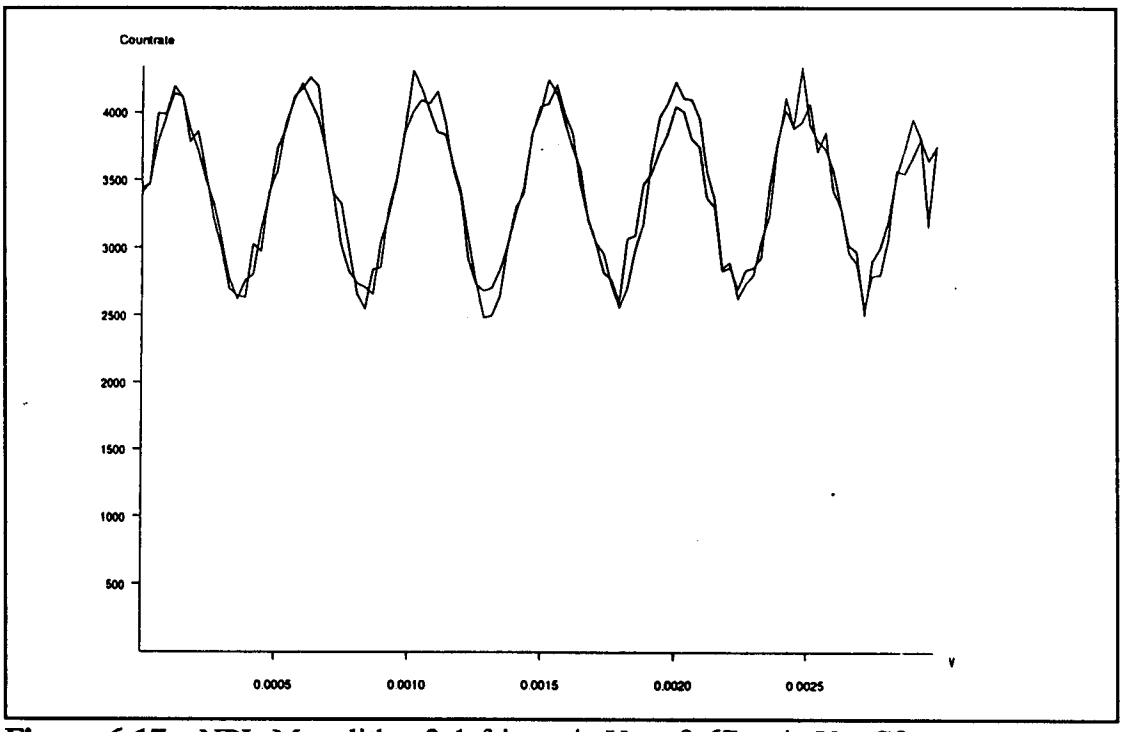


Figure 6.17 - NPL Monolith - 2.1 fringes/mV or 0.67nm/mV - G3

Offset/V	Settling/s	Gate/s	Step size	Comment
Front 4.50	1.0	1.0	30μV	Ramp and return of 100 steps
Rear 4.45			(~20pm)	

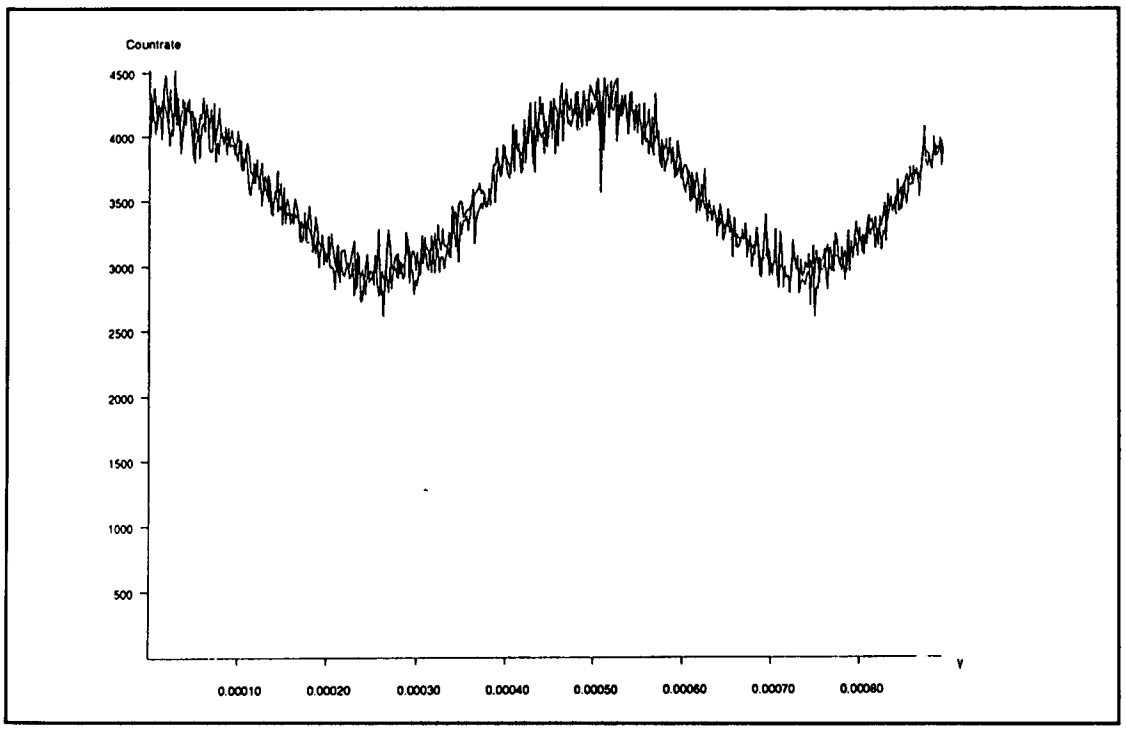


Figure 6.18 - NPL Monolith - 2.1 fringes/mV or 0.67nm/mV - G4

Offset/V	Settling/s	Gate/s	Step size	Comment
Front 4.50	1.0	1.0	2μV	Ramp and return in 30 minutes
Rear 4.45			(~1.5pm)	

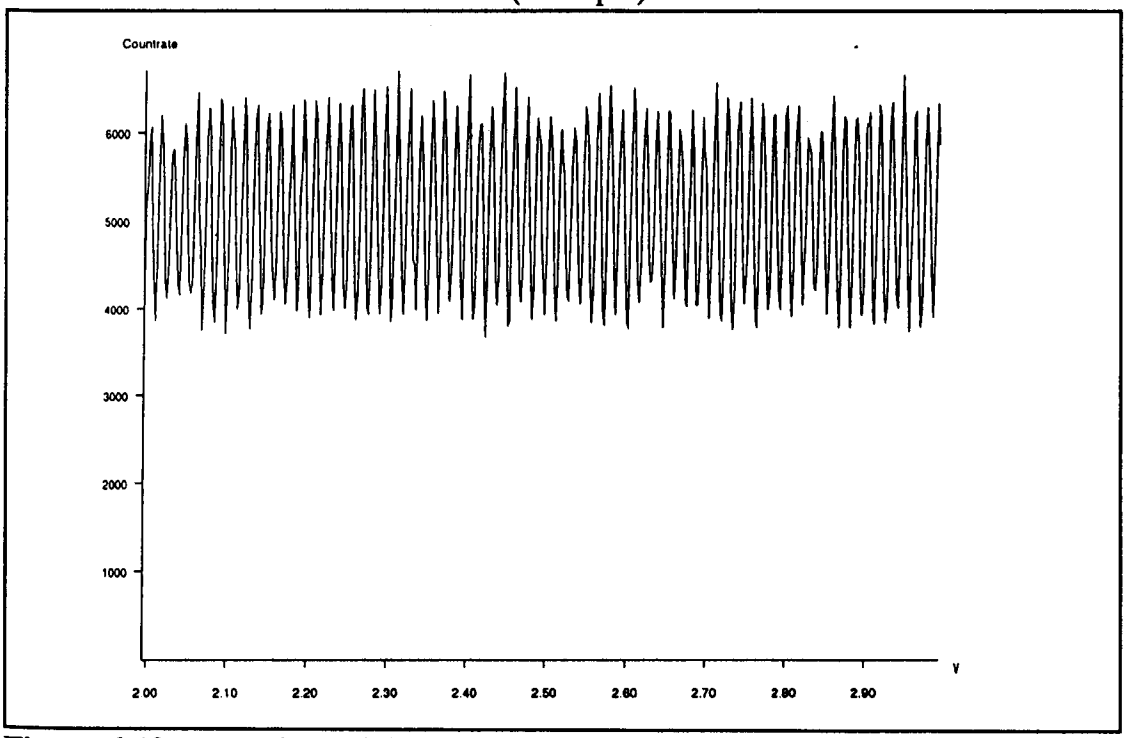


Figure 6.19 - Warwick (stiff) Monolith - 0.067 fringes/mV or 20pm/mV - G5

Offset/V	Settling/s	Gate/s	Step size	Comment
Front -2.00	5.0	0.5	2mV	Long range of 500 steps
Rear -2.65			(~40pm)	

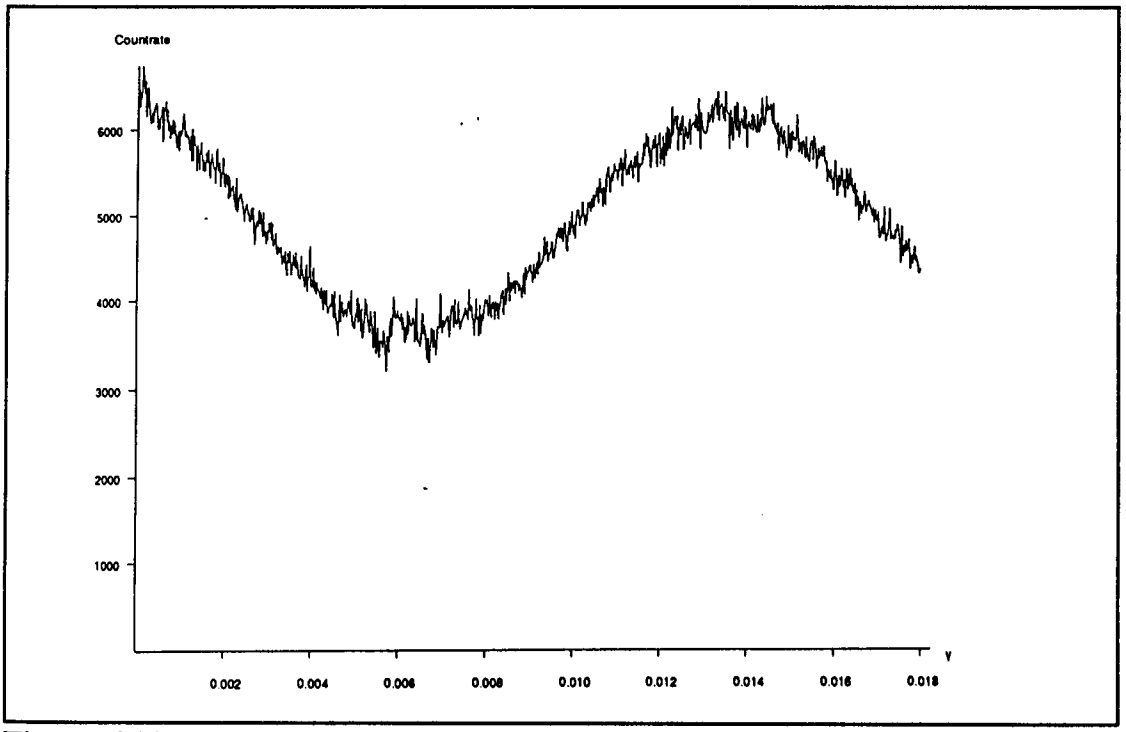


Figure 6.20 - Warwick (stiff) Monolith - 0.067 fringes/mV or 20pm/mV - G6

Offset/V	Settling/s	Gate/s	Step size	Comment
Front 2.02	5.0	0.5	2μV	Very slow scan of 900 steps
Rear 0.78			($\sim$0.4pm)	in 85 minutes

Chapter 7 Alternative approaches and new proposals

It is an inevitable consequence of the very rapid rate of development in integrated circuit design and technology, that by the time a state-of-the-art system design has been implemented and tested rigorously there will have been changes in the commercially available range of components used to make it. Since the changes usually represent improvements, 'better' components are always available at the end of a project. It would also be of some surprise were not the testing to lead to ideas for alternate ways of using new(er) components. This chapter addresses such developments, and although these occurred at irregular intervals over the time of the project, they are presented below in groups for the purpose of discussion. The intention is to highlight ways in which the existing design may be modified and/or extended, to produce a more effective implementation (in terms of eg. cost, precision and speed), and thereby increase the number of areas of application. Comments should be judged with regard to the components available when the design was committed (frozen) in January 1993.

The discussion is also motivated by the fact that the precision ADC module and the high-stability DAC, both made by Analogue Devices, were classified in 1993 as obsolete parts and no longer available. This is surprising because they were so good, but must be a reflection on the fact that there was, or is, insufficient commercial interest in such components for them to be profitable. No other DACs have been found which are as stable, nor ADC modules with integral non-linearity at 1 ppm.

7.1 Availability of improved components

It has been the case for many years that the complexity of digital integrated circuits doubles every few years [7.1]. For analogue integrated circuits the complexity does not necessarily increase in this way but the performance characteristics continue to improve steadily. Some developments are not relevant to the present work, such as high power and high frequency devices, but several developments are worthy of further discussion.

7.1.1 Voltage references at 1 ppm/°C

Precision voltage references which have drifts below 1 ppm/°C have been available

since about 1982 [7.2] but have usually been in the form of temperature controlled/compensated modules, often with significant power dissipation. During the present work a general design strategy was to select components with minimal power dissipation and use forced cooling to maintain the temperature of devices near to ambient conditions. For this reason the most appropriate voltage reference was the AD586LQ [3.5] which dissipated only 30 mW but with a potentially excessive drift of 5 ppm/°C . This was not actually critical because the use of an ADC with a stability of 1 ppm/°C, and a closed-loop servo-control algorithm, compensated for its (AD586) drift. However, if all the components involved with the generation of a reference voltage have negligible drift it would be possible to eliminate the need for an embedded ADC, since an external precision voltmeter could then be used during infrequent (re)calibration cycles. As of March 1995 Maxim Electronics offers a voltage reference in integrated circuit form which incorporates a proprietary method of temperature compensation to achieve a stability of < 1 ppm/°C [7.3]. Although not pin-compatible with the AD586, the circuit may easily be modified to accommodate the changes. The overall design would become less sensitive to local temperature changes and the forced air cooling may not be needed.

7.1.2 Delta-Sigma ADCs

The precision ADC module type AD1175K cost about £800 (1992) and has a resolution of 1 part in about 4700000, or 22-bits, with integral non-linearity of less than 1 ppm. Unfortunately it is no longer available, but cheap low-power integrated ADCs using delta-sigma modulation techniques are becoming much more accurate and may soon achieve a performance similar to the AD1175K but at a fraction of the cost. The CS5504-BP by Crystal Semiconductor Corporation [7.4] offers two differential inputs, self-calibration and 20-bit resolution with no missing codes, in a 20 pin dual-in-line package for £15 (1995). This provides monotonic performance but unfortunately integral non-linearity is (only) 0.0007% FSO. The noise and drift performance do not match the 20-bit resolution, but the device may be used to achieve 17-bit linearity which is still excellent for some applications. It seems likely that as processing improvements are made this kind of ADC will have reduced linearity errors. If this is so, the close-loop design considered here could become very cost-effective.

7.1.3 Linear operational amplifiers with low drift

Chopper-stabilised amplifiers offer very low drift with temperature and time but suffer from excessive noise when compared with other amplifiers. In particular there is usually significant noise at the internal chopping frequency. For most amplifiers this occurs between 250 and 450 Hz, and is undesirable because noise in this region may well cause resonances in associated mechanical components. The amplifier by Texas Instruments type TLC2654A currently used is unique in that its chopping frequency is at 10 kHz; but it requires external capacitors, supplies of ± 8 volts, and the external clock used to synchronise several devices must switch between the negative supply V_{DD-}, and $V_{DD-} + 5$ V.

It is much easier to design with non chopper-stabilised amplifiers but the drift of offset voltage with temperature, and limited gain, are usually severe limitations. However, Burr-Brown have introduced the OPA177E [7.5], which has some performance figures usually associated with chopper-stabilised amplifiers. Unfortunately its input bias current is still large when compared with FET input amplifiers and may limit its application. However FET amplifiers often have larger drifts with temperature, the OPA627 [7.6] being a notable exception. Both amplifiers may be appropriate for future designs, see Figure 7.1.

Parameter	Value(s)
OPA177E	
Voltage offset	4 µV (typ) 10 µV (max) at 25°C
Voltage offset drift	0.03 µV/°C (typ), 0.1 µV/°C (max)
Open loop gain	134 dB (min)
Input noise voltage	85 nV rms (typ), 1 Hz to 100 Hz
Long term drift	0.2 µV/month
Input bias current	0.5 nA (typ)
bias current drift	25 pA/°C (max)
Supply current	1.3 mA at ±15 V
OPA627BM	
Voltage offset	40 µV (typ) 100 µV (max) at 25°C
Voltage offset drift	0.4 µV/°C (typ), 0.8 µV/°C (max)
Open loop gain	112 dB (min)
Input noise voltage	0.6 µVp-p (typ), 0.1 Hz to 10 Hz
Long term drift	not specified
Input bias current	1 pA (typ) 5 pA (max) at 25°C
bias current drift	doubles every 10°C
Supply current	±7 mA at ±15 V

Figure 7.1 - OPA177E and OPA627BM specifications

7.1.4 Buffer amplifiers

The use of the TLC2654A, with supplies limited to ± 8 V, in the voltage-to-current converter meant that the output power amplifier had to provide voltage, as well as current, gain. See circuit diagram 5 in Annex [1]. This precluded the use of many buffer amplifiers which have a voltage gain of 1. If a design were based on the OPA177E discussed above with its increased output voltage swing, buffer amplifiers such as the BUF634 [7.7] could be used. This particular device, also by Burr-Brown, is available in a TO220 package, which is significantly smaller than the TO3 package used by the OPA654M [3.6]. The saving in PCB area is considerable and leads to reduced costs. The reduction in package size may result in increased local heating, and therefore voltage offsets, but this is of little consequence as it still lies within an analogue servo-loop whose performance is dominated by that of the OPA177E.

7.2 Use of an external voltmeter

If the programmable current source has to be a self-contained embedded system then the present design demands a local precision ADC. However with the introduction of a 'slave' computer the possibility exists for communication with an external precision d.c. voltmeter via a suitable digital interface. Obviously the introduction of a computer violates the idea of self-contained sources, but this may not be of much import in some applications. For example the Schlumberger voltmeter type 7081 [7.8] has a resolution better than 1 part in 10^7, linearity of <0.2 ppm of full scale, and a temperature coefficient of less than 0.3 ppm/°C, as well as IEEE488 and RS232 interfaces. This makes it ideally suitable for measuring the voltage across the precision reference resistor, although at £5000 (1993) it is more expensive than the cost of all other components. However using newer and smaller components, it seems possible to build 2 or 4 programmable current sources on one PCB in a single chassis, and multiplex the sensing terminals of the precision resistors to the external voltmeter. The cost of the voltmeter is therefore shared across a number of programmable sources and may be cost-effective. The proposed arrangement is shown in Figure 7.2a). It is essential for the analogue multiplexer to have negligible leakage currents on unselected channels and for the cross-talk between channels for be lower than -120 dB, if overall performance is to reach 1 ppm, and a number of multiplexers exist which are suitable for this purpose. Another requirement is to include unity gain buffer amplifiers on the PCB to isolate the precision resistors from the effects of capacitive loading in the cable connected to the voltmeter.

In order to achieve closed-loop control of several multiplexed drives it will be necessary to change the select code at a regular rate. It is a characteristic of all analogue multiplexers that small charges are injected at the selected input, as the select code changes. These charges may be of the order of pico-coulombs every 0.1 seconds, and may disrupt the current in the coil in a significant way. To alleviate this problem additional capacitive decoupling may be included as shown in Figure 7.2b). The additional components will introduce undesirable parasitic thermo-electric effects but the differential characteristics remain unchanged. Unfortunately even a charge injection as low as 5 pC, and a capacitor of 0.1 μF, will give a voltage change of 50 μV, which on a range of 5 volts represents 10 ppm. Consequently it may be necessary to buffer each connection to the reference resistors before multiplexing, as shown in Figure 7.2c). The low impedance output of each buffer amplifier will be able to supply current to compensate for the effects of transitory charge injection, but the input impedance must be high enough not to load the reference resistors unduly. Since the analogue multiplexer is unlikely to be ideal and has an internal resistance between a selected input and the output, it is still necessary to include a buffer amplifier at its output so that negligible current flows across it, and there are no parasitic voltage drops. If the internal resistance is independent of temperature and absolute voltage level it may be possible to compensate for parasitic losses but this is improbable.

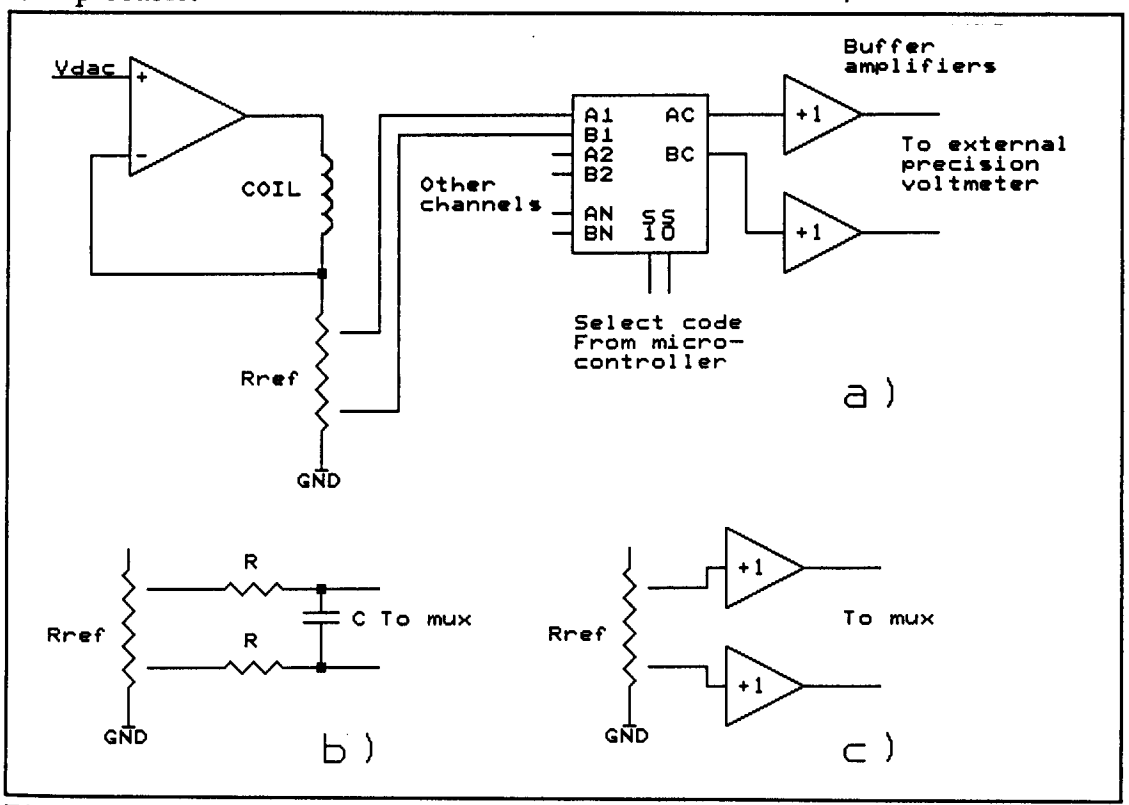


Figure 7.2 - Use of external voltmeter

Another aspect to the use of an external voltmeter is to consider a system where a the performance of a multiple-output current source is so stable and predictable that a voltmeter is needed only for infrequent calibration cycles. For example if re-calibration is needed only once every few days and takes two hours, the meter may be used for other measurements most of the time. It is therefore worth considering designs which may not exhibit excellent linearity but are very stable.

7.3 Application of switch capacitor building blocks

A novel integrated circuit type LTC1043, is available from Linear Technology, and is described as a "Dual precision instrumentation switched-capacitor building block" [7.9]. It consists of a number of analogue switches which act together to connect a capacitor alternately between different pairs of pins under the control of an internal or external clock. Proprietary circuits to compensate for charge injection are incorporated, and suggested circuits indicate that some signal processing may be performed to within ± 1 ppm. A number of useful applications for this device have been identified and are presented below.

7.3.1 For voltage halving and inversion

Digital-to-analogue converters invariably create a unipolar output which must be biased by half-range to produce a bipolar output. This is usually done with matched on-chip resistors connected to a summing amplifier and the reference voltage. However it may be preferable to bias the amplifiers directly, and this may be done by a combination of voltage halving and voltage inversion. The circuits in Figure 7.3 showing how these functions may be achieved, and were taken from [7.9]

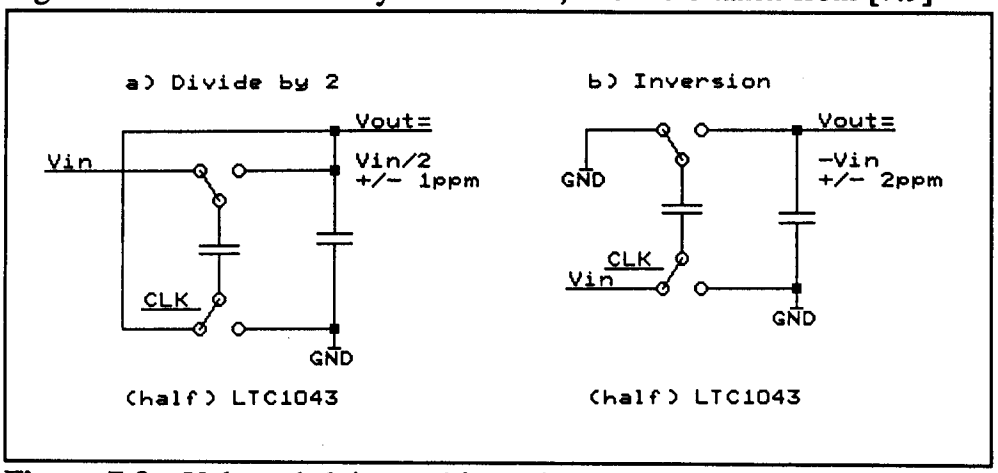


Figure 7.3 - Voltage halving and inversion

The quoted tolerances on the outputs assume they are unloaded and that no currents flow once equilibrium levels have been established. Care should be taken to buffer the outputs if they are to drive other circuits.

7.3.2 In voltage-to-current conversion

The end effects of all 2-terminal resistors mean that for precision voltage-sensing applications 4-terminal devices must be used where current forcing pins are separated from voltage-sensing pins. This necessarily elevates the voltage of both pins from ground and creates a differential output. The general arrangement of using part of an LTC1043 to convert a differential voltage to a single-ended signal with respect to ground has been copied from the manufacturers data sheet and is shown in Figure 7.4a). Applying this technique to create a precision voltage-to-current converter is shown in Figure 7.4b), where the differential voltage across the sensing terminals of a precision resistor is reference to ground and used to provide negative feedback around a very high gain power amplifier. Assuming the input current of the amplifier is negligible, the voltage across R will be zero, and equilibrium will be achieved when, for all practical purposes, $V_h = V_{dac}$. Since V_h=voltage across sensing pins of R_{ref}, and the coil and R_{ref} are in series, the current in the coil is proportional to V_{dac}. C reduces the high frequency response and provides closed-loop stability.

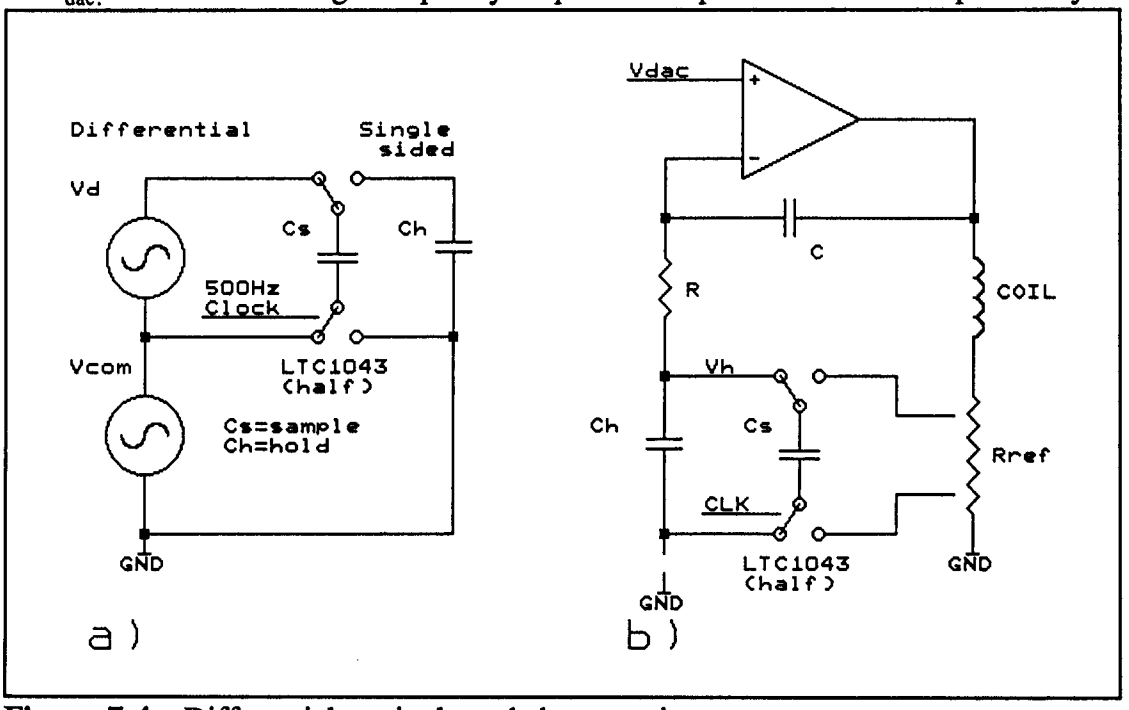


Figure 7.4 - Differential to single-ended conversion

If the gain of the power amplifier is large enough, as should be the case using a combination of the OPA177E and BUF634, it may be possible to show that the

voltage-to-current converter subsystem tracks V_{dac} voltage so well that measurement, or calibration, of this alone is enough to characterise the changes of current in the coil with respect to DAC levels. This is of benefit in two respects, firstly, the measurement is made outside the voltage-to-current servo-loop and therefore cannot interfere with its operation, and secondly the DAC voltage is single-ended, and easier to multiplex than the differential voltage across the sensing pins of R_{ref}.

Although the LTC1043 incorporates elements to compensate for charge injection, it may still present a dynamic load to R_{ref} at the beginning of each sampling interval, consequently a decoupling capacitor may be advantageous. It is also beneficial to bias the 'hold' capacitor, C_h, to creates a bipolar output current for a unipolar input voltage. These additions are shown in Figure 7.5, where the LTC1043 has been replaced with 4 single pole switches and the general format has been modified to aid the preparation of a circuit definition ready for simulation by PSPICE [6.2].

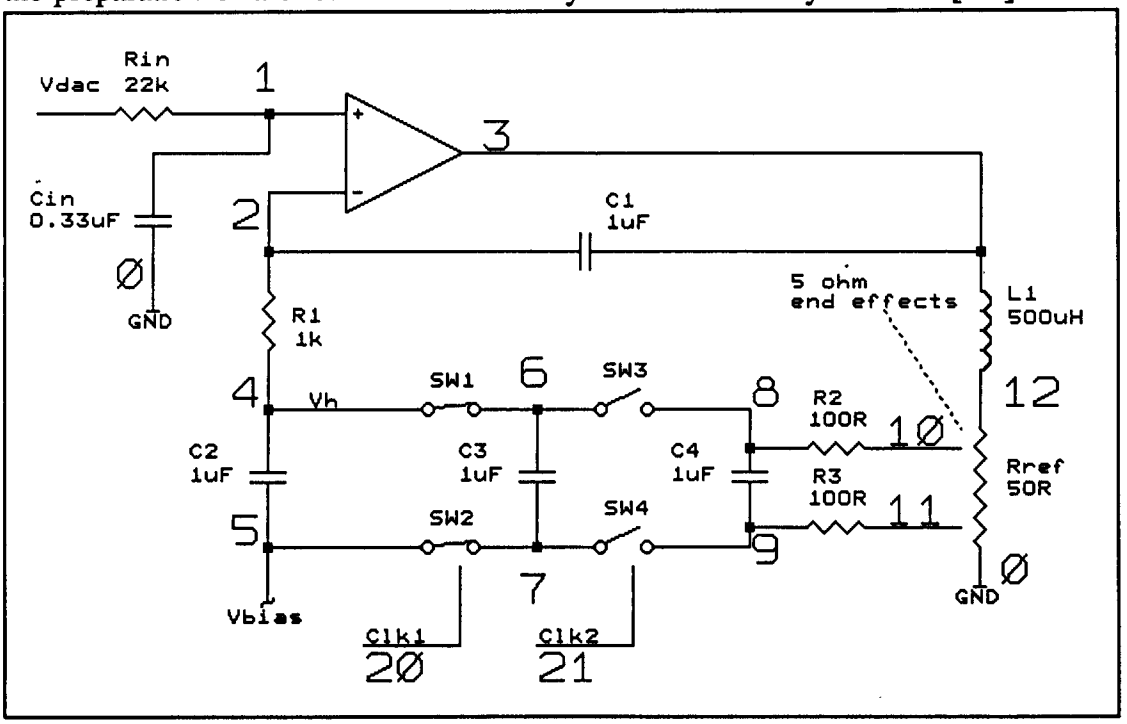


Figure 7.5 - Switched capacitor V-to-I with bias

In order to minimise the effects of charge injection the sample and hold capacitors should be of the order of 1 μF. R2,R3 and C4 should be chosen to provide added decoupling (filtering) but their time-constant must not be so large that a significant phase shift arises around the whole circuit, as this will result in unwanted oscillation.

Using the component values given in Figure 7.5 several simulations were performed, for different values of the input filter capacitor C_{in}. It is important to limit the rate of rise at the non-inverting input of the amplifier, in order to avoid overshoot in the current through the coil, as this may have adverse effects in a system with mechanical

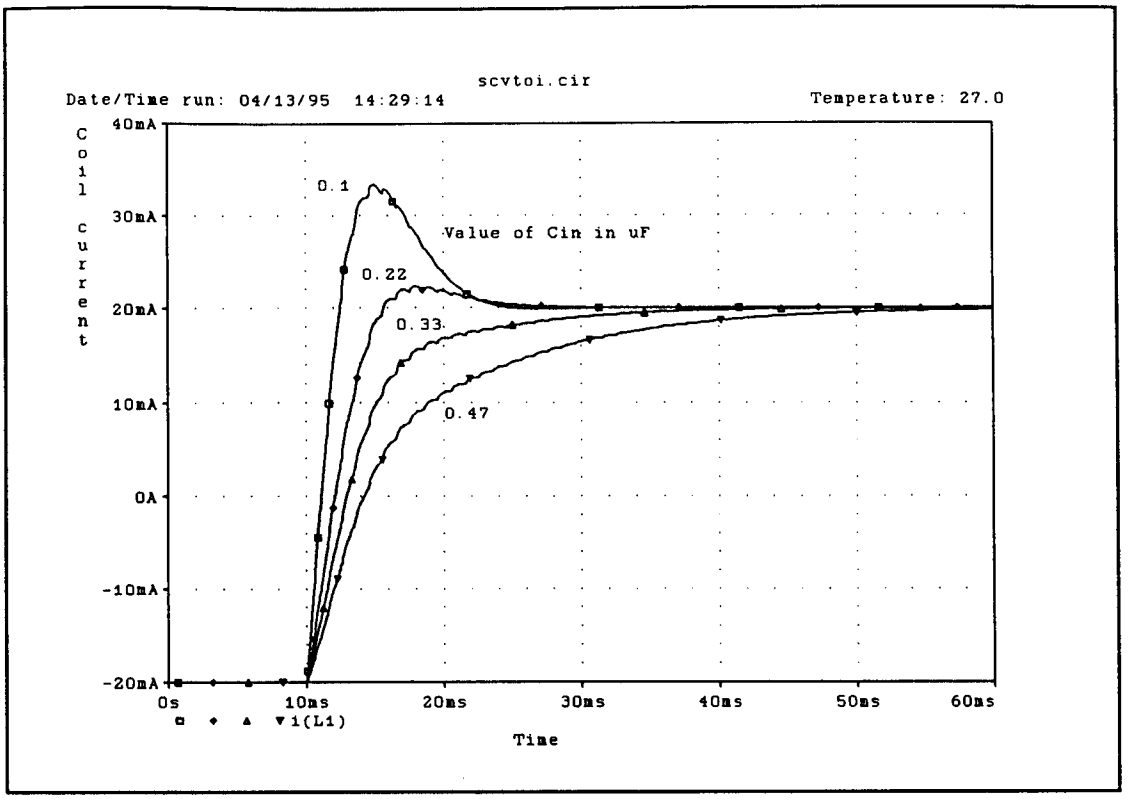


Figure 7.6 - Response of switched capacitor V-to-I

hysteresis. The curves in Figure 7.6 show the coil current with respect to time for various values of C_{in}, when the input voltage changes from 1.5 V to 3.5 V at time= t_c=10 ms. Before t_c the current is stable at (1.5-2.5)/R_{ref} = -1/50 = -20 mA and thereafter it rises to +20 mA. Study of the simulation source-file called *SCVTOI* at the end of Annex [5] shows how non-overlapping clocks at 1 kHz were created for the switches, and how bias conditions were defined to give bipolar current output.

That part of the response from 20 ms to 40 ms when C_{in} = 0.33 μF is shown in Figure 7.7, together with a scaled clock to make clear the synchronous nature of the sampling. As time passes the current difference of successive levels is reduced, and, although not shown, equilibrium is achieved when the voltage across the sensing pins of R_{ref} equals (V_{in} - V_{bias}) = 1 volt, and the current becomes constant at +20 mA.

The simulated performance of a voltage-to-current converter based on a switched capacitor looks promising but further experimental work is needed to establish the exact behaviour of the LTC1043 and the overall dependence on temperature. It is also appropriate to discover if the clocks for multiple LTC1043's in the same system are best synchronised (or left to free-run), bearing in mind that, for a particular mechanical system, there may be a resonant frequency which is the same as that used to clock the switches. It a real design perhaps there should be a way of setting the clock frequency.

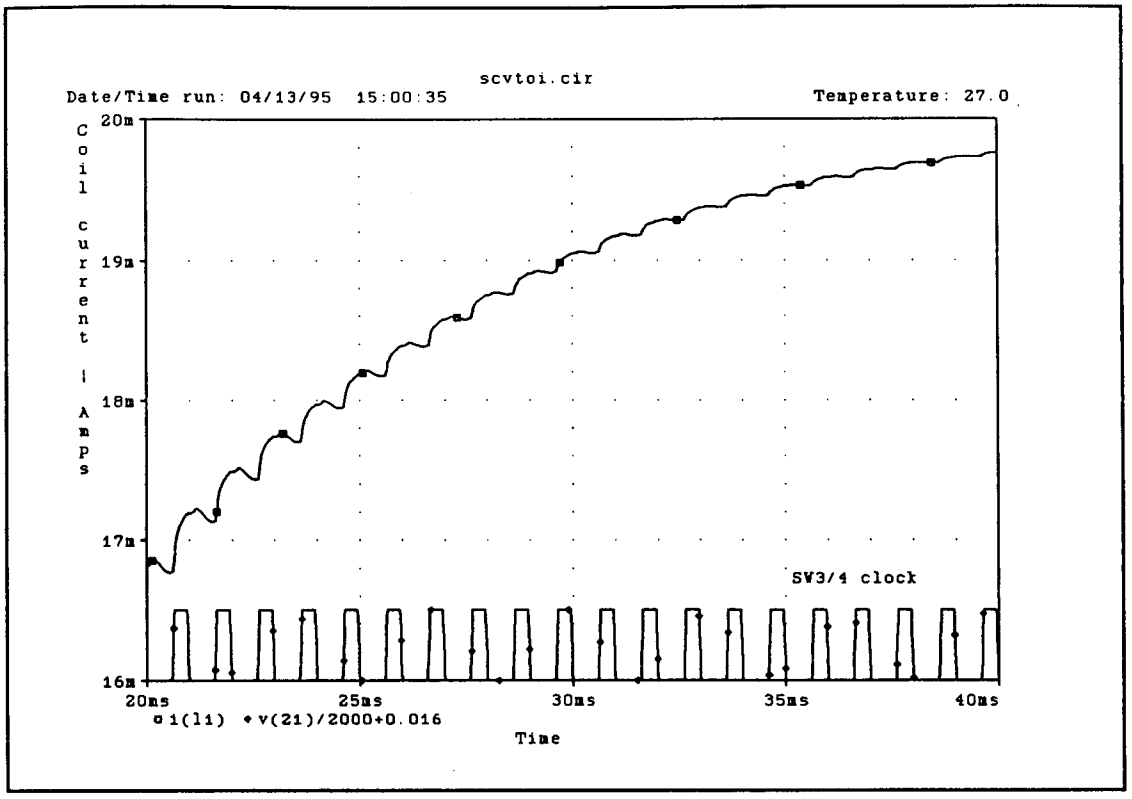


Figure 7.7 - Response of switched capacitor V-to-I

7.3.3 Deglitching and demultiplexing a DAC

A sample and hold circuit is often used to de-glitch the output of a digital-to-analogue converter. The idea is to remove the transient voltage signal, which occurs at the DAC output when the digital control code is changed, by isolating a 'holding' capacitor until the transient subsides. An analogue switch is usually employed but these are prone to charge injection which offsets the level being stored. However the LTC1043 has internal charge compensation circuits, and charge injection is significantly reduced. This suggests it may be used as a deglitching element by synchronising its switching action with changes in the DAC output as shown in Figure 7.8a), where the DAC code is changed only when C_s is connected to C_h. Most of the time the DAC level will not change between samples, but there would be no interference with C_s if the DAC *did* change while C_s is connected to C_h. This immediately introduces the idea of time demultiplexing the DAC between two channels, as shown in Figure 7.8b). So long as the time for the glitch is short compared with the sampling time, each C_s will acquire the correct level by the end of the sampling interval just before it is connected to the corresponding C_h.

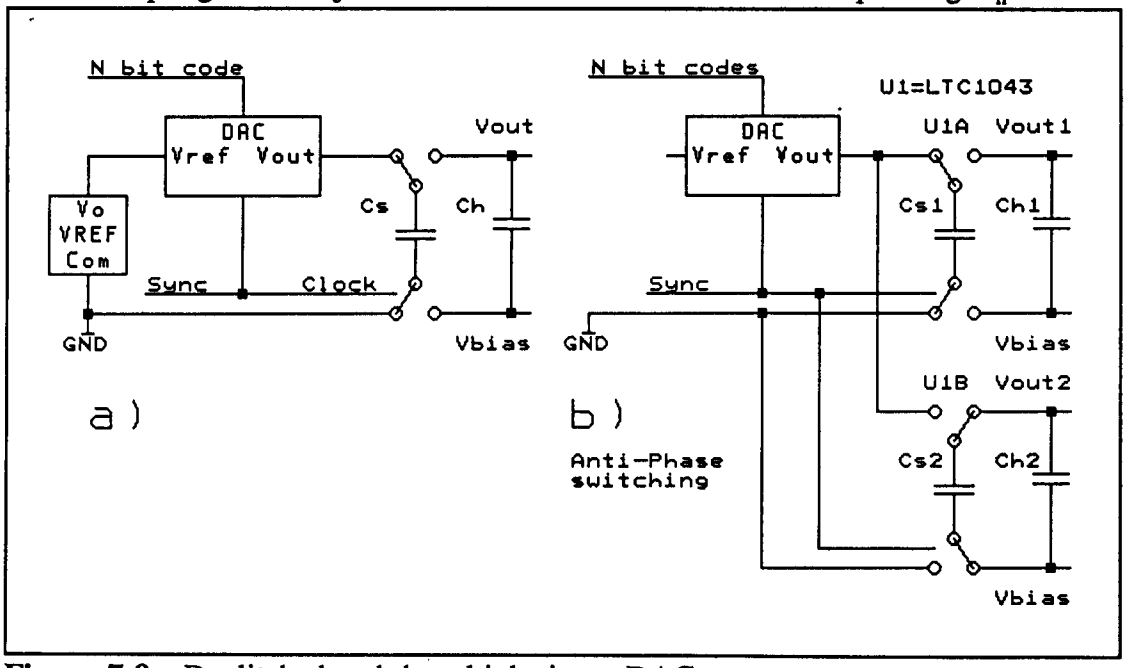


Figure 7.8 - Deglitched and demultiplexing a DAC

The behaviour of Figure 7.8b) is best explained with the aid of the waveforms presented in Figure 7.9. At the beginning of each sampling interval an ideal DAC would respond by a simple linear change from one level to the next after a fixed delay, however a real DAC will output a code-dependent 'glitch' after an additional delay. Suppose DAC code m produces level M and is associated with C_{s1}, and code n produces level N and is associated with C_{c2}. In the case of an interval of type-m,

C_{s1} is briefly subject to the previous level N, then a glitch, before settling to the desired level M. The value of C_{s1} has to be chosen after consideration of the internal resistance of the analogue switches in the LTC1043, and the drive capability of the DAC output, such that it will always be able to *acquire* the expected voltage at the end of each interval. Strictly this condition must be satisfied only when the channel is in equilibrium, for when a large change occurs as M becomes M', C_{h1} will not follow immediately, and many samples will be necessary before charge redistribution is completed. While the charge on C_{h1} is in flux the voltage on C_{s1} need not settle to the expected level by the end of its sampling interval. The behaviour of C_{s2} and C_{h2} is similar but C_{s2} *acquires* level N by the end of type-n intervals.

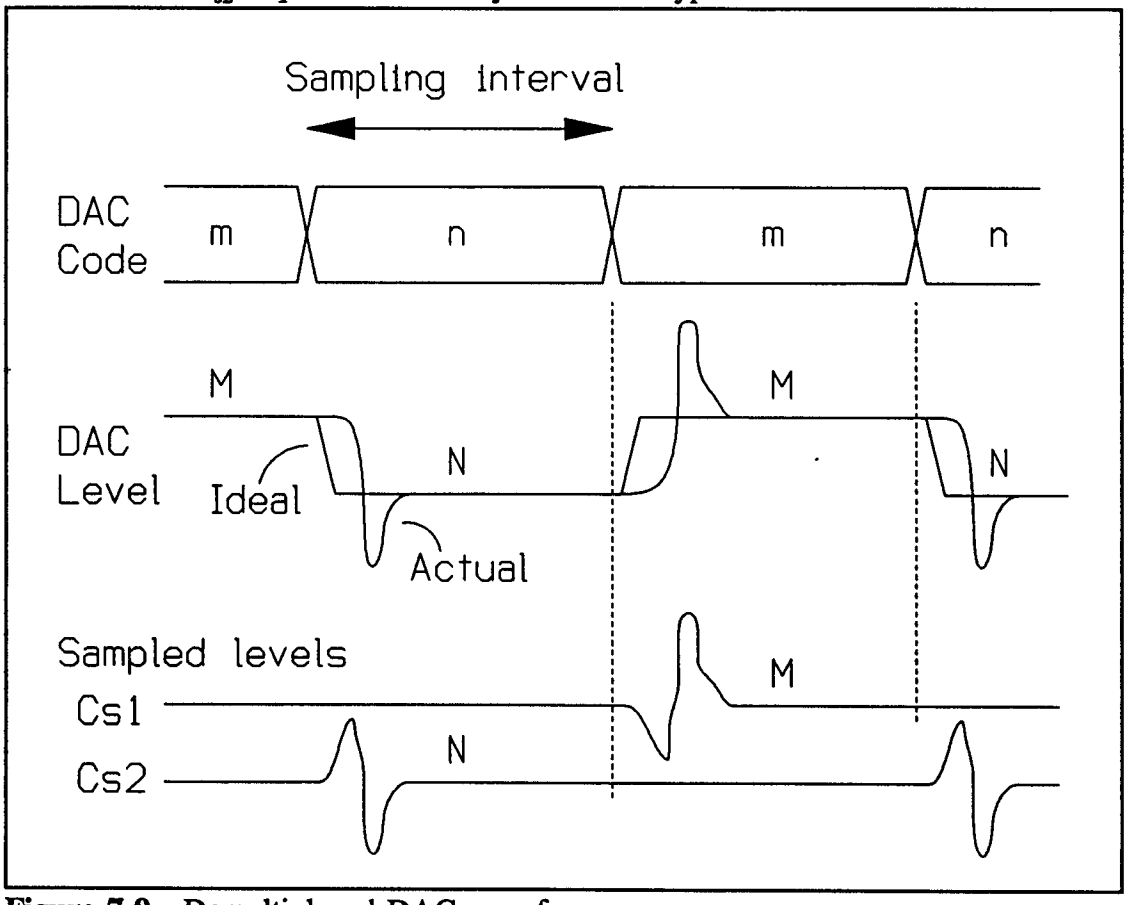


Figure 7.9 - Demultiplexed DAC waveforms

The LTC1043 contains two sampling circuits that operate in anti-phase and this leads directly to the implementation in Figure 7.8b), but its operation is not optimal because each sampling capacitor still 'sees' the DAC glitch. A better arrangement is shown in Figure 7.10, where two LTC1043's (with only half of each being used) are controlled by independent clocks both synchronised to changes in the DAC output. In this way each C_s is connected to the DAC output after the glitch has died away. The duty cycle is necessarily no longer 1:1, but need not be as bad as the waveforms suggest, as the duration of each glitch appears exaggerated. Many DAC have a glitch

which lasts for only a few microseconds, and with a nominal sampling time of 0.5ms, the duty cycle need not be far from 1:1. If the DAC includes an amplifier with limited slew rate, it may be this, and not the glitch-time, which limits the time for the DAC to change from level M to level N. In which case the clocks may have a duty cycle of 1:2 or more, and the effect of this remains to be investigated.

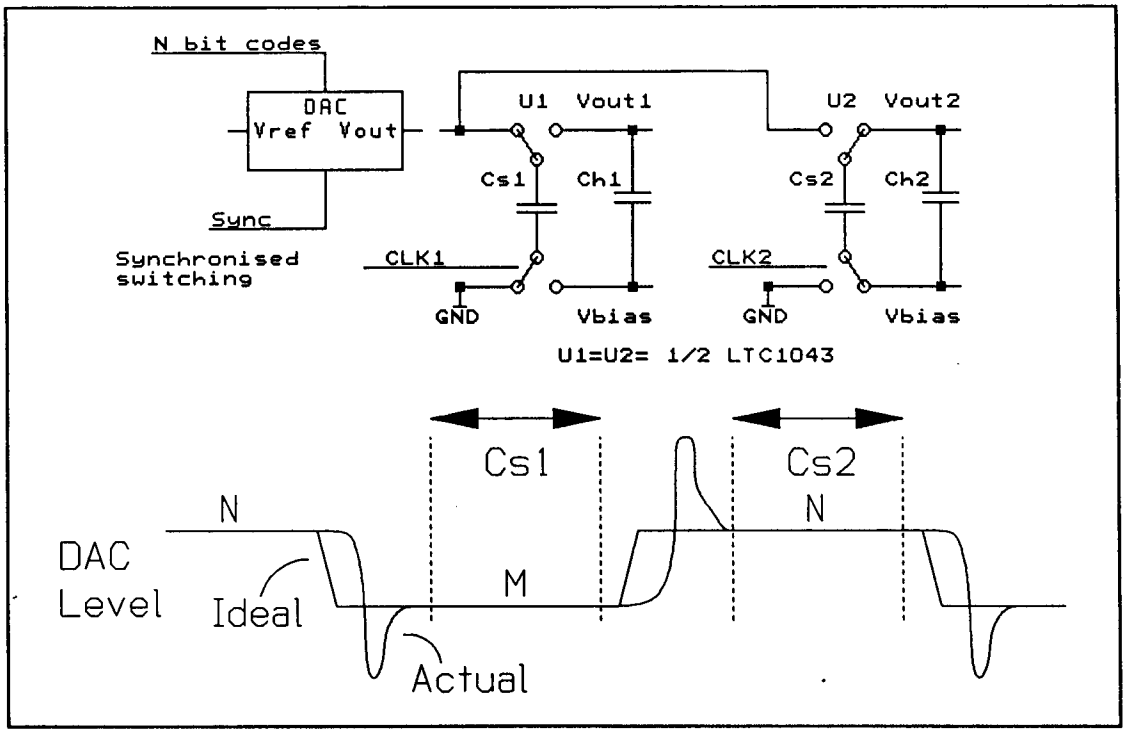


Figure 7.10 - Better demultiplexing

It is anathema to electronic design engineers for circuits to incorporated into a design and not be fully utilised, yet this would be the case by using only half of two LTC1043's. Fortunately the other half may be used in the voltage-to-current converter, as simulations show its performance is acceptable even when the duty-cycle of the switched-capacitor is not 1:1.

7.3.4 Interpolation in a time demultiplexed DAC

As a further modification to the previous design, the level presented to the sampling capacitor during each of its sampling intervals need not be the same. By including additional filters the pulse level modulated signal thus generated may be averaged to provide interpolation between DAC levels. Further experimental work is needed to establish the scope for this technique and the optimal design of the filters. The extended circuit is shown in Figure 7.11.

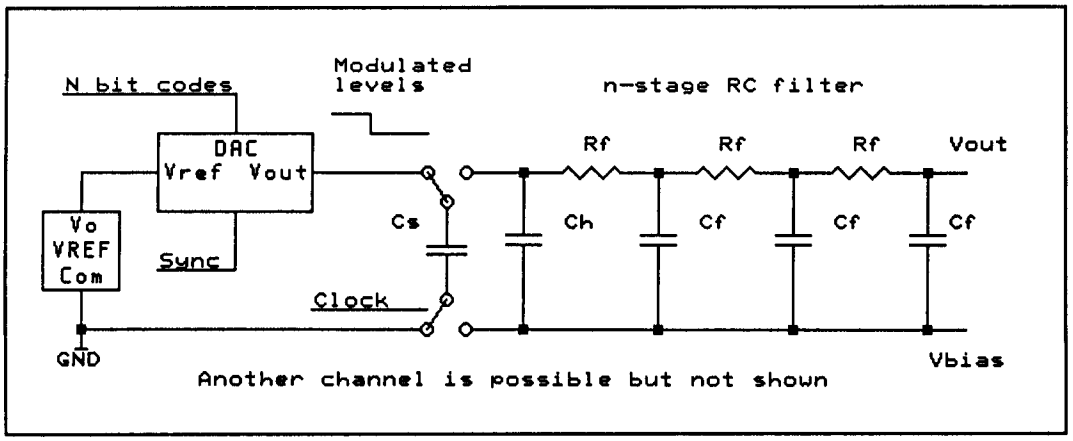


Figure 7.11 - Interpolation in a time demultiplexed DAC

As was discussed in section 2.3.2 the integral non-linearity (INL) of this approach will be limited to the INL of the DAC. But so long as it is monotonic it should be possible to provide an ideal linear scale by calibration using a precision ADC or voltmeter.

7.4 PWM DACs and Field Programmable Gate Arrays (FPGA)

The difficulties of creating a precision pulse-width-modulated DAC using a 20-bit counter were discussed in section 2.3.1. The settling time was considered excessive and the technique dismissed. However it may be practical to produce a 10-bit PWM DAC, with a clock rate of about 1 MHz and corresponding cycle time of $2^{10} \times 10^{-6}$ = 1.024 ms, which requires a low pass filter to remove frequency components of about 1 kHz and above. A settling time of 100 ms is acceptable for many applications and it may be possible to achieve this with a combination of low drift active filters and/or passive RC filters. Although there are only 1024 levels, and such a DAC will still use an analogue switch with associated charge injection, its overall integral non-linearity may be small and approach 1 ppm for each level. This needs to be shown by experimentation but if it can be achieved it may be used as a coarse DAC in a system with coarse and fine DACs. The great advantage is that its performance may be accurately predicted and characterised by measurement of relatively few levels. Combining its output with an 'ordinary' monotonic 16 bit fine DAC in the ratio 1:64 ($1:2^6$) will provide a composite DAC with a resolution of 16+6 = 22 bits, as depicted in Figure 7.12. The characteristics of the fine DAC are less critical than those of the PWM DAC, and numerous devices with acceptable specifications are commercially available.

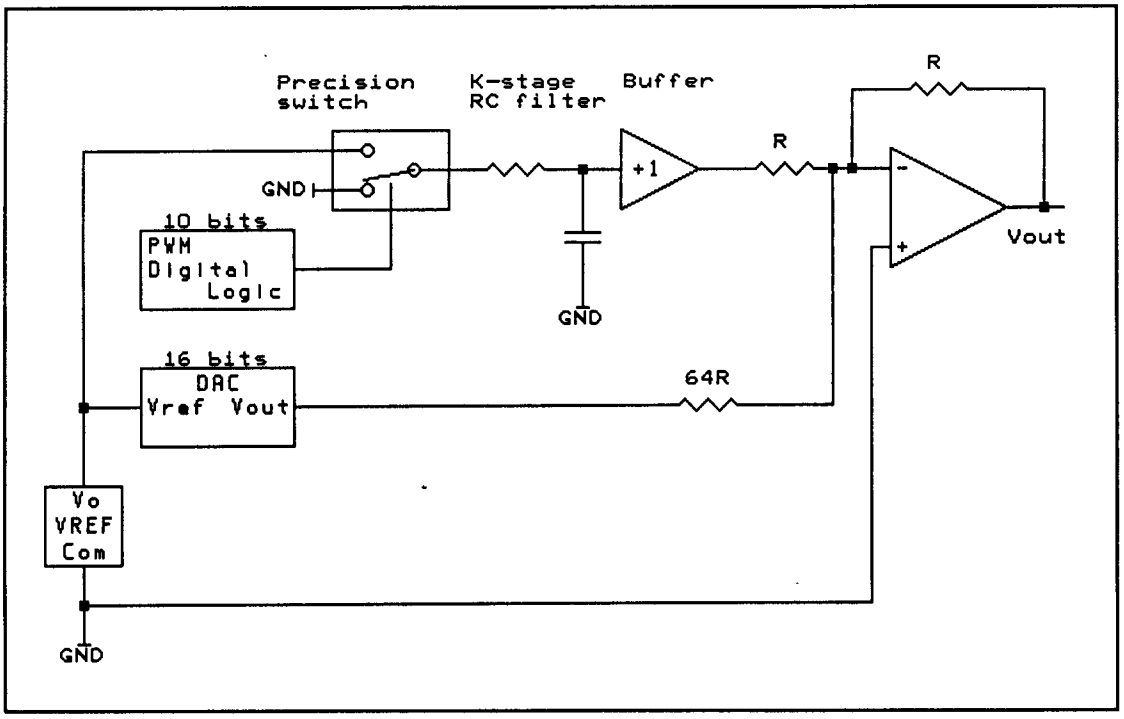


Figure 7.12 - Using a 10-bit PWM DAC as the coarse DAC

In a design with multiple composite converters of this form the number of digital integrated logic circuits 'on the analogue side' conflicts with the principle outlined in 2.1.4. However, Field Programmable Gate Arrays (FPGAs) [7.10,7.11] are readily available which have more than sufficient resources to implement several 10-bit PWM converters. Since all digital logic is confined to one integrated circuit, it may be possible to keep the effects of interference to an acceptably low level. It is also possible to use additional capacity in the FPGA to time-demultiplex the fine DAC.

7.5 Suppression of power on/off transients

The current in the coil during power-on, and power-off, conditions was measured and presented in Figure 6.10. Although the power-on transient is acceptably small, the power-off transient is large and unpredictable, and in making proposals for future designs this issue must be addressed. When the mains power is interrupted the behaviour of the power amplifier which drives the coil becomes undefined. The most reliable way to ensure there is no force in the actuator is to use a relay that shorts out the coil, thereby removing all current flow. 'Normally closed' contacts are used, and the relay is not energised unless correct power supply regulation is established. Some care must be taken in the selection of the relay, as it must have a response time of only a few milliseconds, and not subject to 'welding' of its contacts in the event of high transient currents. A miniature mercury-wetted reed-relay is most suitable. The design for a control circuit is shown in Figure 7.13, which is powered from an

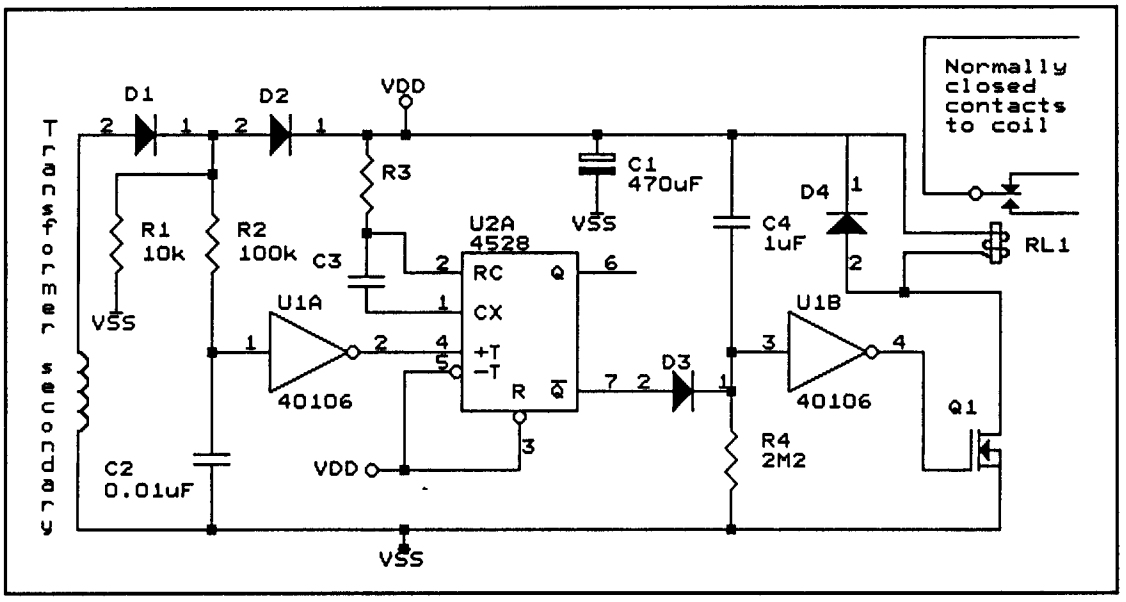


Figure 7.13 - Power on/off protection

independent mains transformer. Its operation is relatively simple but effective; at power-on there is a delay, determined by C4 and R4, before the relay is energised; while during normal operation the retriggerable monostable (U2A), with a time-out of about 0.1 s, detects an excessive number of missing mains half-cycles and causes the relay to de-energise before the power rails fail. There may be multiple sets of relays or contacts for designs with more than one coil.

7.6 Proposal for dual drive

Most of the ideas presented in this chapter have been incorporated into a design for a dual programmable source and appear in Figure 7.14. The circuit is more than a block diagram, but less than a complete design, since individual sections within this chapter have already dealt with specific sub-functions in detail. A number of signals and functions are highlighted by a * character, to indicate they are generated within one FPGA. What is not shown is an embedded microcontroller which accepts serial messages from a controlling computer and defines the operation of the FPGA accordingly. To understand the basic operation it is sufficient to describe the behaviour of one channel, as the other is identical. V_{ref} is a stable voltage reference for the DAC and the precision switch, both of which are controlled by the FPGA. PS1 is driven by a pulse width modulated signal using a 10-bit counter, while the DAC receives 16-bit code levels which are synchronised to the clock for U1. The PWM logic (running at 1 MHz) and the timing of the DAC (with sampling at 1 kHz) will be synchronised in some way by virtue of the fact that the FPGA has a single master clock frequency of several MHz, and it may be advantageous to ensure the

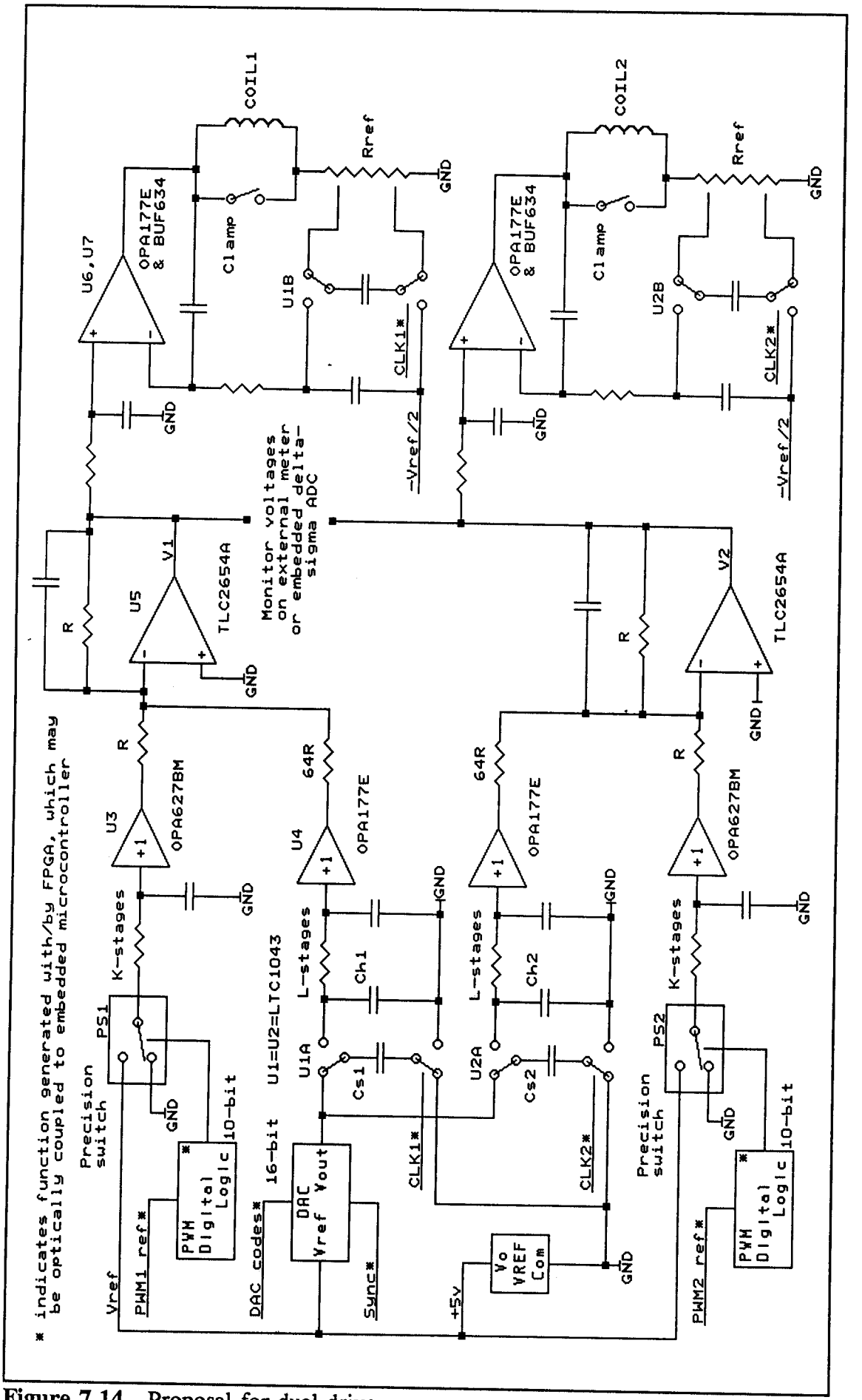


Figure 7.14 - Proposal for dual drive

cycle time for all signals are in a simple ratio. If this is not the case, undesirable 'beat' frequencies may be generated.

The analogue PWM signal from PS1, and the sampled signal from the DAC, are both filtered by multiple RC stages, however these are of different length and do not offer the same attenuation. For the PWM signal the K-stage filter must provide an attenuation of more than 120 dB at the fundamental frequency, so the output is stable to 1 ppm. Assuming a frequency of 1 kHz, 8 stages of 10 kΩ and 0.1μF will provide sufficient attenuation. The L-stage filter for the DAC output may not even be needed, but a value of L=1,2 is practical. Care must be taken to consider the effects of input bias current for U3 and U4. If U3 were type OPA177E (bipolar) the bias current of 0.5 nA results in an offset of 40 μV (for a source impedance of 80 kΩ). This, in itself, may not be too serious, as calibration will compensate for the offset, however its bias current changes at a rate of 25 pA/°C (max.) causing a voltage drift of 2μV/°C. U3 is specified as type OPA627BM (FET) because its bias current is only 5 pA, which results in a voltage loss of 0.4 μV across the filter resistors, and its voltage drift is only 0.8 μV/°C.

The summing amplifier, U5, needs to have low input bias current (<50 pA), low drift (<1 μV/°C) and high gain (>120 dB). These requirements are considered best satisfied with a chopper-stabilised amplifier type TLC2654A. In the original design, amplifiers of this type were used throughout, and there was concern that, by having unsynchronised clocks, the output of one amplifier may cause aliasing at the input of the next. Consequently additional circuitry was employed to provide synchronised clocks. In the design now being considered there is only one chopper stabilised amplifier (per channel) and synchronisation is not necessary. The TLC2654A must operate from supplies of no more than ±8 V, whereas driving amplifiers U3 and U4 may operate from ±15 V. However, using ±15 V introduces the possibility of latch-up at the input of U5, and it is therefore considered preferable to operate U3, U4 and U5 all at ±8 V.

The output of the summing amplifier provides the reference for the voltage-to-current converter, but compared with previous discussions and simulations it is negative. Consequently the bias is $-V_{ref}/2$ and the interpretation of the digital codes for the PWM logic and DAC must be inverted, to restore the expected sense of changes in the coil current. (Although not shown $-V_{ref}/2$ may be derived from V_{ref} by halving the inverted value, using circuit elements from Figure 7.3). The combination of an OPA177E with a BUF634 gives an amplifier which is capable of delivering up to

$\pm$50 mA at 12.5 V, or $\pm$100 mA at $\pm$12 volts, with other characteristics determined by the OPA177E. The value of R_{ref} should be chosen such that the maximum required coil current produces a voltage drop of $V_{ref}/2$ across it. For example, if V_{ref}=5 V and I_{max}=50 mA, the value of R_{ref}= 5/2/0.05 = 50Ω. Assuming the end effects of R_{ref} are small compared with its value the maximum coil resistance is (12.5-2.5)/0.05 = 200 Ω. This is larger in value than those that have been tried, but increasing the maximum allowed resistance is beneficial as it allows more turns of finer wire, which will produce a stronger magnetic field.

In chapter 1 some mention was made of the application of multiple actuators to complicated mechanical systems. For a long range X-ray interferometer 2 or possibly 3 drives are beneficial, but it is not impossible to consider systems that could use from 4 to 8 drives. It is therefore of interest to consider the design of a circuit with more than 2 outputs. Of course it is possible merely to replicate the design presented in Figure 7.14, and create a number of dual drives in a single chassis, but this would be wasteful of a FPGA used to create only 2 x 10-bit PWM DACs. However it seems altogether feasible to extend the idea of time demultiplexing the DAC (see Figure 7.10) from 2 to 4 drives, and make better use a FPGA to generate 4 PWM signals. The switched capacitor elements will necessarily have a duty cycle of at least 1:3, but may still operate as required. An adverse consequence of this approach is to lower the sampling rate to about 250 Hz, which may lead to unwanted mechanical resonances if the electrical signals are not filtered properly.

While some of the methods proposed need experimental evaluation to establish their potential for super-precision applications, there is no reason to believe any method is intrinsically unworkable. The methodologies employed for the present drives were appropriate given the constraints of the project for the DTI/NPL, however if progress is to be made towards more general research in precision programmable current sources, quantitative analysis of real circuits based on new proposals is needed. Additional simulations using idealised components will not prove the practicality of the techniques, and it will not be until circuit boards have been fabricated and instrumented that a full understanding of the performance will emerge. All this is left for future work.

Chapter 8 Final comments and conclusion

Throughout this thesis considerable effort has been made to present ideas in a logical order and in sensible groups. However, certain thoughts would have disrupted the flow, and/or had insufficient background, if they were to be included within certain sub-sections. Consequently these have been left to the end and are now discussed. They should be considered in the further development of programmable drives or systems that use them. Thereafter is a final section which concludes the work.

8.1 Settling time of DACs

In the present design the coarse and fine DACs are of the same type and each has a settling time of 6 μs to $\pm$0.5 LSB, which is very short compared with the time constant of the filter at the input of the voltage-to-current converter. Since the embedded software always changes both DACs simultaneously their weighted output also settles in a similar time. This is especially significant when a small overall change is required but the fine DAC is already at full scale. In this situation the coarse and fine DACs must both change, and it is desirable for the transition to the new level to occur smoothly. If the coarse DAC takes a long time to settle compared with that for the fine DAC the net output will not be a smooth transition.

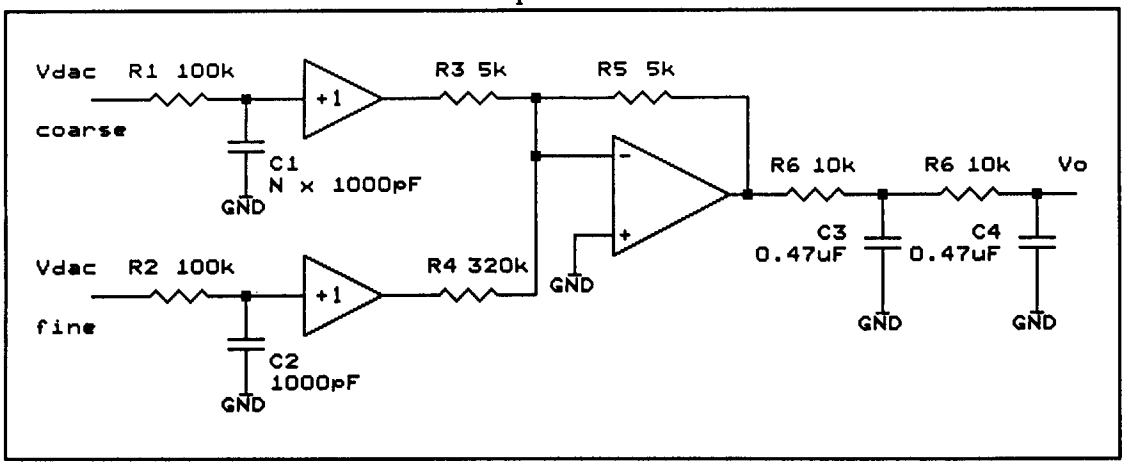


Figure 8.1 - Circuit depicting DACs with different time constants

Consider the circuit in Figure 8.1 which represents an idealised model of the present circuit as taken from diagram 4 in Annex [1]. It shows two voltage inputs from ideal coarse and fine DACs driving single stage RC networks. A simple RC network (with a nominal time constant of 10 μs) may be inappropriate for a real DAC but is acceptable for the purpose of discussion. The filtered outputs are buffered and summed in the ratio 1:1/64 and then filtered again. The output Vo is used to drive the voltage-to-current converter.

The simulated response is shown in Figure 8.2 when the time constant for the coarse DAC is 0.5, 1, 2 and 5 times that of the fine DAC. At time=5 ms the coarse and fine DACs change from -0.95 to -1.05, and -3.2 to +3.1968 volts respectively, which causes a net increase of 50 μV to the initial level of 1 volt. For N>1 there is change in the wrong direction, and, when N=5, a pulse whose amplitude is many times the desired step change. This is clearly undesirable but altogether likely for the composite DAC proposed in Figure 7.12, where the coarse DAC level is derived from a filtered PWM signal. Consequently future design(er)s should include components which balance the response time of DACs. Alternatively the fine DAC may be made to pass through a number of sub-levels which compensate for any delayed response of the coarse DAC, in such a way that the output pulse is reduced or removed.

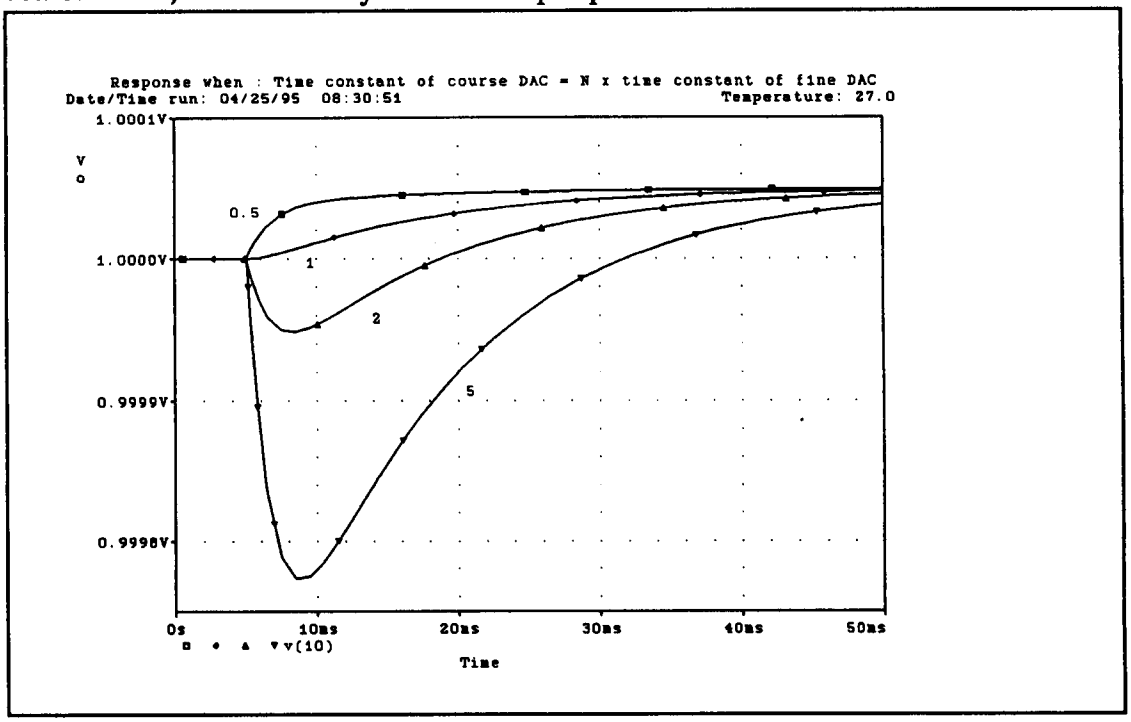


Figure 8.2 - Responses when DACs have different time constants

8.2 Additional auxiliary circuits

Several components are included in the current circuit which are not strictly needed during normal operation, but provide additional confidence that all is working well. For example, the auxiliary ADC channel which monitors the voltage level of the main power amplifier, and the sensor which monitors the temperature of the exhaust air. These are of trivial cost when compared with other components and their inclusion was certainly worthwhile. However, if the project were repeated, additional circuits should be included to provide even more knowledge about the performance. Three such examples are:-

a) It is very difficult to measure the noise current in the coil because it is so small, and external amplifiers are subject to unwanted interference. Consequently an a.c. coupled differential amplifier with a gain of about 1000 should be placed across the connections to the coil. A typical circuit appears in Figure 8.3. The output provides a signal of the order of milli-volts that can be taken to an oscilloscope or spectrum analyzer. The simple high-pass RC network has a -3dB point at 0.3 Hz, which is well below the resonant frequency of typical mechanical mechanisms. The response is almost flat above 1 Hz and hence all frequencies of interest will be passed through with negligible attenuation.

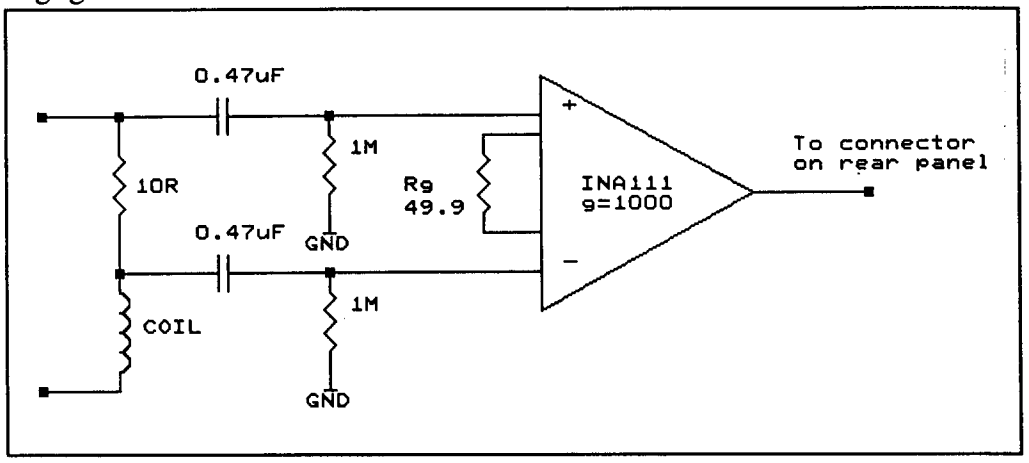


Figure 8.3 - Embedded diagnostic amplifier

b) Circuit diagram 6 in Annex [1] shows that two of the inputs to the analogue multiplexer connected to the auxiliary ADC are not used and are tied to ground. This represents an inefficient use of resources and the inputs should be brought to a connector on the rear panel so the user can have the benefit of reading two external voltage levels. Care should be taken not to compromise the isolated analogue ground signal.

c) In the same vain as b) some of the digital inputs associated with U22 are unused and may be of more general use if brought to a connector on the rear panel. Care should be taken not to introduce unwanted earth loops and/or noise.

Of course there comes a point where additional circuits may represent a significant cost and this should be considered, but the suggestions above are inexpensive and easy to implement.

8.3 Commercial considerations

Much has been said already about the alternative methods and/or improvements that can be made to the present design, but from some points of view these comments represent a rather narrow perspective of the possibilities, for they do not take into account potential modifications to the systems as a whole. If the x-ray interferometer, as developed for the DTI, is looked at from a commercial position a number of other thoughts for the future emerge.

The DTI project was essentially a 'one off' exercise, and involved little consideration for the sale of duplicate systems. The dominant cost was for the development of technology that exceeded existing capabilities; all of which was carried out on one unit. Hence its cost was high when compared with a possible production unit, but the methods of construction can be justified as they allowed the instrument to meet the specifications on time and within budget. The market for such precision instruments is small and there is little practical need to plan for vast sales. However, refinements may be made to the electronic control sub-system and these are worthy of discussion, since the precision current sources may be used on other systems which have the potential for commercial use in greater numbers in metrology laboratories.

The x-ray interferometer contains a number of expensive components, such as the x-ray source, detector, camera etc., all of which are interfaced to a controlling computer running commercially written software [8.1]. This software expected a remote programmable subsystem to respond to simple message packets, sent out via a serial port, and cause movement of the monolith. When a computer was included within the subsystem it was natural to describe the controlling computer as the *host* and the subsystem computer as the *slave*. However, no provision was made for anything other than the communication of set-point values from the host to the slave computers, using a proprietary software handshake mechanism. In particular no facility was provided for the host computer to 'learn' about the status of the subsystem circuits, and it was necessary to include a display for the slave computer. This display gave a great deal of diagnostic information which was useful during the development of the drives and was certainly cost effective for the DTI project but can be rendered unnecessary by modifications to the software in the host and slave. Using extended data exchange mechanisms will allow all data currently presented on the slave's screen to be presented at the host's screen upon request, as it is needed only infrequently. In order to remove the slave PC entirely it is necessary to embed the servo control algorithm within each drive, and there is sufficient capability for this

to be done using the floating point interpreter which has been written. Unfortunately the removal of the slave PC means that two serial ports are needed on the host for communication to each drive, as inferred from Figure 8.4a) which shows the present arrangement. This may be inconvenient for systems that require one port for other facilities, as a total of two ports is the usual maximum on many desk-top computers. Fortunately there are two ways around this problem. The first is to connect the drives in a 'daisy' chain and for extended messages to contain an address identifier so they may be relayed to the correct drive as depicted in Figure 8.4b). The second requires that the host computer and drives have hardware for multi-drop communications such as either RS485, or IEEE488, as shown in Figure 8.4c). The first arrangement is preferable because it is easy to preserve the isolation between different drives. Although RS485 tri-state drivers may be optically isolated the communications strategy is more complicated. The IEEE488 communications protocol, with some 16 signal lines, does not lend itself to isolation, and earth loops may be formed between the drives. The most effective arrangement is shown in Figure 8.4d) where a single RS232 communications channel connects the host to a multiple drive, which contains a common power supply, and microcontroller to support a number (say 2 or 4) programmable current sources.

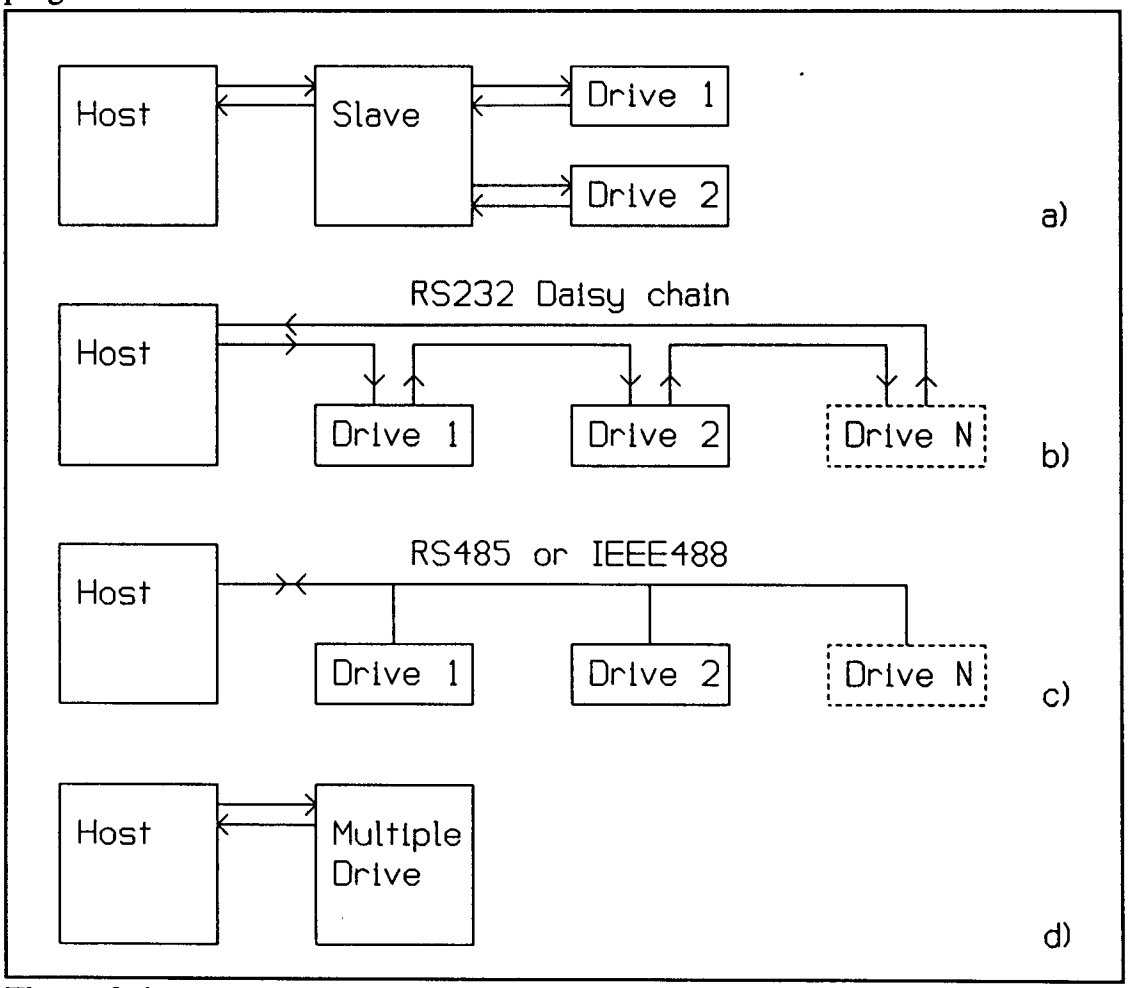


Figure 8.4 - Alternative communication schemes

8.4 Conclusion

The need to design a super-precision programmable current source to meet the needs of a long range x-ray interferometer was established in chapter 1, as was the fact that such a device has other applications in metrological environments. The chapters that followed presented a methodical development of ideas from initial concepts and constraints, to implementation and testing. Extensive use was made of computer aided design and simulation software and there are numerous suggestions for further work. Taken together they show that a programmable current source may be build with all performance parameters measured in parts per million. As a project it was deemed entirely successful for the immediate application. The exceptional stability and linearity of the current source(s) is over 100 times better than commercial systems and provides opportunities for new levels of control for current-driven devices.

References

[1.1] Bleaney, BI, B Bleaney: "Electricity and Magnetism", *Pub. Oxford University Press. 1976.* Chapter 4 page 98.

[1.2] Richtmyer, FK, E H Kennard, T Lauritsen: "Introduction to modern physics", *Pub. McGraw-Hill, fifth edition, 1955* p32-33.

[1.3] Montgomery, DB: "Solenoid magnet design", *Pub. Wiley Inter-science NY, 1969.*

[1.4] Smith, ST, DG Chetwynd: "An optimised magnet-coil force actuator and its application to precision elastic mechanisms", *Proc. Instn. Mech. Engrs.* **204,C4**, 243-53, 1990.

[1.5] Smith S T, D G Chetwynd D K Bowen: "Design and assessment of monolithic high precision translation mechanism", *J. Phys E : Sci. Instrum.* **(20)**, p977-983, 1987.

[1.6] Liu X, D G Chetwynd S T Smith W Wang: "Improvement of the fidelity of surface measurement by active damping control", *Meas. Sci. Technol.* **4** 1330-1340, 1993.

[1.7] Noltink B E (Editor): "Instrumentation reference book" ,*Pub. Butterworth-Heinemann,* **Pt.1 Ch.7**, 1988.

[1.8] Hicks T R, N K Reay and P D Anderton: "The application of capacitance micrometry to the control of Fabry-Perot etalons", *J Phys. E : Sci.Instrum.* **63**(2), p49-55. 1984

[1.9] Downs M J: "Optical metrology : The precision measurement of displacement using optical interferometry", *From instrumentation to nanotechnology Ed. JW Gardner and HT Hingle, Pub Gordon and Breach ISBN 2-88124-794-6,* Chapter 11, p213-226, 1990

[1.10] Chetwynd D g, X Liu and S T Smith: "Controlled-force stylus displacement probe", *8-IPES Universite de technologie de Compiegne,* May 15-19 1995.

[1.11] Becker P, P Seyfried and H Sigert: "The lattice parameter of highly pure silicon single crystals", *Z Phys B - Condensed matter,* **48**, p17-21, 1982.

[1.12] Bonse U, M Hart: "An X-ray Interferometer", *Appl. Phys. Lett,* **6** p155-156, 1965.

[1.13] Hart M: "An Ångstrom ruler", *J.Phys. D: Appl Phys,* **1**, p1405-1408, 1968.

[1.14] Chetwynd D G, S M Harb, N O Krylova and S T Smith: "The feasibility of extended range monolithic x-ray interferometric calibrators", *Nanotechnology,* **4**, p183-193, 1993.

[1.15] Chetwynd D G, D P Siddons and D K Bowen: "X-ray interferometer calibration of microdisplacement transducers", *J. Phys. E: Sci. instrum,* **16**, p871-4, 1983.

[1.16] Becker P, P Seyfried: "Calibration of optical and mechanical sensors by x-ray interferometry", *Proc. SPIE,* **1015**, p124-129, 1989.

[1.17] Bowen D K, D G Chetwynd, D R Schwartzenberger and S T Smith: "Sub-nanometre transducer characterisation by x-ray interferometry", *Precision Engineering,* **Vol. 12 no. 3**, July 1990

[1.18] Dyer D C: "High precision current drives for low speed actuators - Initial application on long-range x-ray interferometer", *4th Biennial joint Warwick/Tokyo nanotechnology symposium, at the University of Warwick,* 19th-23rd September 1994.

[1.19] Schwartzenburg D R et al: "Phase measurement x-ray interferometry", *J. x-ray Sci. Technol.*, **1**, 134-42

[1.20] Marconi Instruments (St. Albans England): "Instruments catalogue 1989/90", model 103A, pages 396-398.

[1.21] Keithly Instruments: "Catalogue and reference guide 1989-90", model 224 p102

[1.22] Fluke and Philips: "Test and measurement catalogue 1992", model PM2831 section 14, p14-17

[1.23] Hewlett Packard: "Test and measurement catalogue 1993", model 6625A, p469

[2.1] Horowitz and Hill: "The Art of Electronics", *Pub. Cambridge UP,* 1989

[2.2] Page 6 of [3.5]

[2.3] Schlumberger: "Maintenance Manual for Digital Voltmeter type 7081", *Section 5.3 (ref 2927g/0142/JWS) and circuit Reference 2 (ref 70817505 Sh2)*

[2.4] Page 20 of [3.4]

[2.5] Buckingham M J, J Michael: "Noise in electronic devices and systems", *Pub Chichester Horwood,* 1983

[2.6] Pages 130 and 133 of [6.2]

[2.7] BIDS - The Bath ISI Data Service, Bath University

[2.8] Gorkunov,ES, AV Kadrov, RP Petrov: "Programmable current supply for inductive load", *Instruments and Experimental Techniques*, **29/3**, 677-678, 1986

[2.9] Skyba, P: "Microcomputer-controlled, programmable current source for NMR measurements at very low temperatures", *Review of Scientific Instruments*, **62/11**, 2666-2670, 1991

[2.10] Chen, Z, G Yao, F Ling: "Development of a programmable current source", *Chinese Journal of Infrared and Millimetre Waves*, **12/3**, 243-247, 1993

[2.11] Communication with Mr. P. Cooke of Cooke Consulting, 16 Firle Road, North Lancing, West Sussex, England. Tel 01273 414620

[2.12] DTI contract "Traceable secondary standard displacement facility with sub-nanometre resolution - DTI reference MPU 8/0.13"

[3.1] OrCAD Systems Corporation: "Schematic Design Tools III", *Document Number 1871100A*, 1987

[3.2] See 4.2

[3.3] Analog Devices: "AD1145BG DAC data sheet"

[3.4] Texas Instruments: "TLC2654AC Chopper-stabilised amplifier data sheet", *sheet number D3174*, 1989

[3.5] Analog Devices: "AD586LQ High precision 5V reference data sheet"

[3.6] Burr-Brown: "OPA654M Wide bandwidth, high output current, DiFET operational amplifier data sheet" 1990

[3.7] Maxim Integrated Products: "MAX252ACHL Complete, +5v-Powered, Isolated, Dual RS-232 Transceiver Module data sheet"

[3.8] Xicor Inc.: "X28C256 5 volt, Byte alterable EEPROM data sheet",*sheet number 3855-1*, 1991

[3.9] Racal-Redac: "CADSTAR version 7.0, printed circuit board design software"

[3.10] Vishay resistive systems group: "Bulk metal foil fixed resistors", Vishay-mann, Wymondham, Norfolk.

[4.1] Motorola Inc.: "MC6805 Family Users Manual", *Publication 6805UM(AD) Ed 1*, 1980

[4.2] Motorola Inc.: "MC68HC705C8 Technical Data Book", *Publication MC68HC705C8/D REV1*, 1990

[4.3] Motorola Inc: "MC68HC05 Applications Guide", *Publication MC68HC05AG/AD*, 1989

[4.4] Maxim Integrated Products: "MAX132CNW - ±18 bit ADC with Serial Interface data sheet", *sheet number 19-0009; rev 1:7/92*

[4.5] Motorola Inc. Electronic Bulletin Board: Tel. USA 512-891-3733 at 2400 baud.

[4.6] Technical System Consultants: "Stack Oriented Arithmetic Processor for 6800 mpu", *TSC Box 2574 W. Lafayette. IN 47906 USA*. 1977. Supplied by Windrush MicroSystems. Worstead Laboratories, North Walsham, Norfolk NR28 9SA, England.

[4.7] Hewlett Packard: "HP 1000 Scientific Library Functions", 1979

[5.1] Borland International: "Turbo Pascal version 5.0", *1800 Green Hils Road, PO Box 660001, Scotts Valley, CA 95066-0001, USA*

[5.2] Blaise Computing Inc.: "Asynch Plus/5.0 - User Reference Manual", *Pub. Blaise Computing, 819 Bancroft Way, Berkeley, CA94710 USA.*

[6.1] Kennedy, EJ: "Operational Amplifier Circuits : Theory and Applications", *Pub. Holt, Rinehart and Winston, Inc. 1988.* Chapter 4. Noise analysis in Op-Amp Circuits.

[6.2] MicroSim Corporation: "PSPICE Circuit Analysis software", *20 Fairbanks, Irvine,California 92718, USA.*

[6.3] Computer Associates: "CA- SuperCalc for DOS", *Pub. Computer Associates International Inc.1991.*

[6.4] The MathWorks Inc.: "Matlab for MSDOS Computers", *Cochituate Place, 24 Prime Park Way, Natick, Massachusetts 01760 USA.*

[6.5] Keithley Instruments: Data acquisition system 'metrabyte μdas16' on IBM PS2

[6.6] Murray Sargent III, Richard L. Shoemaker: "The IBM PC from the Inside Out - revised edition", *Pub. Addison-Wesley 1986 ISBN 0-201-06918-0.* Section 6.4

[6.7] See page 34 of [5.2]

[6.8] Private communication with Dr. D.G. Chetwynd, Department of Engineering, University of Warwick

[7.1] Anderla G, A Dunning: "Computer Strategies 1990-9 : technologies-costs-markets", *Pub. John Wiley and Sons, ISBN 0-471-91585-8,* p177/8

[7.2] Codi Semiconductor: "Certavolt™ - PSV 10, 10 volt reference"

[7.3] Maxim Integrated Products Inc.: "Data sheets for MAX676,677 and 678 - Precision voltage references",120 San Gabriel Drive, Sunnyvale, CA, USA.

[7.4] Crystal Semiconductor Corporation: "Data sheet for CS5504 - Low power 20-bit A/D converter", publication number DS126PP1, July 1993.

[7.5] Burr-Brown Corporation: "Data sheet for OPA177E - Precision operational amplifier", publication number PDS-1081B, Linear products data book p 2.19-2.26, 1994

[7.6] Burr-Brown Corporation: "Data sheet for OPA627BM - Precision high-speed Difet Operational amplifier", publication number PDS-998E, Linear products data book p 2.181-2.192, 1994

[7.7] Burr-Brown Corporation: "Data sheet for BUF634 - 250 mA high speed buffer", publication number PDS-1206, Linear products data book p 3.18-3.26, 1994

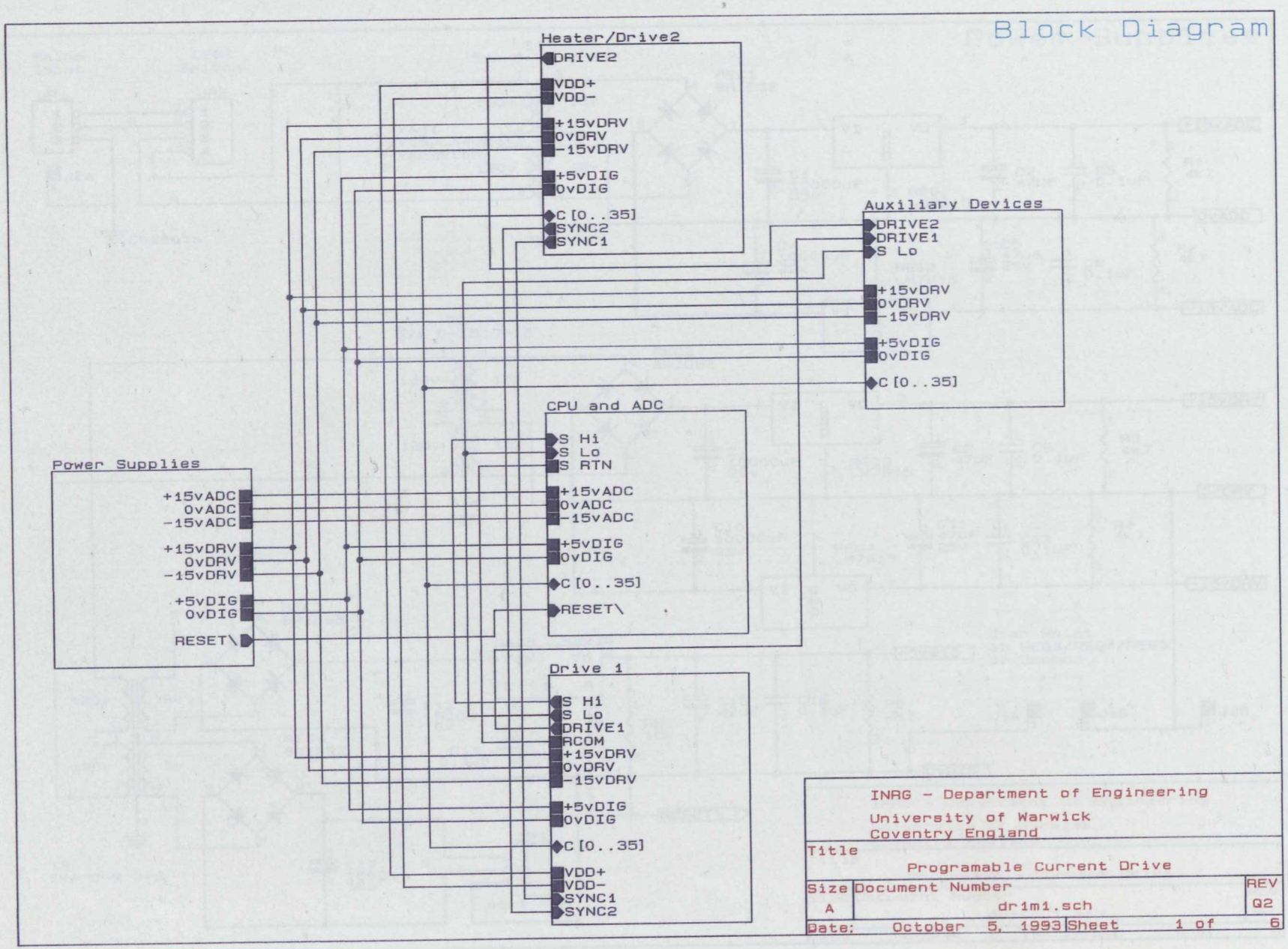
[7.8] Schlumberger Technologies; Instruments division : "7081 computing voltmeter", *Operating manual part 1,* p1.10

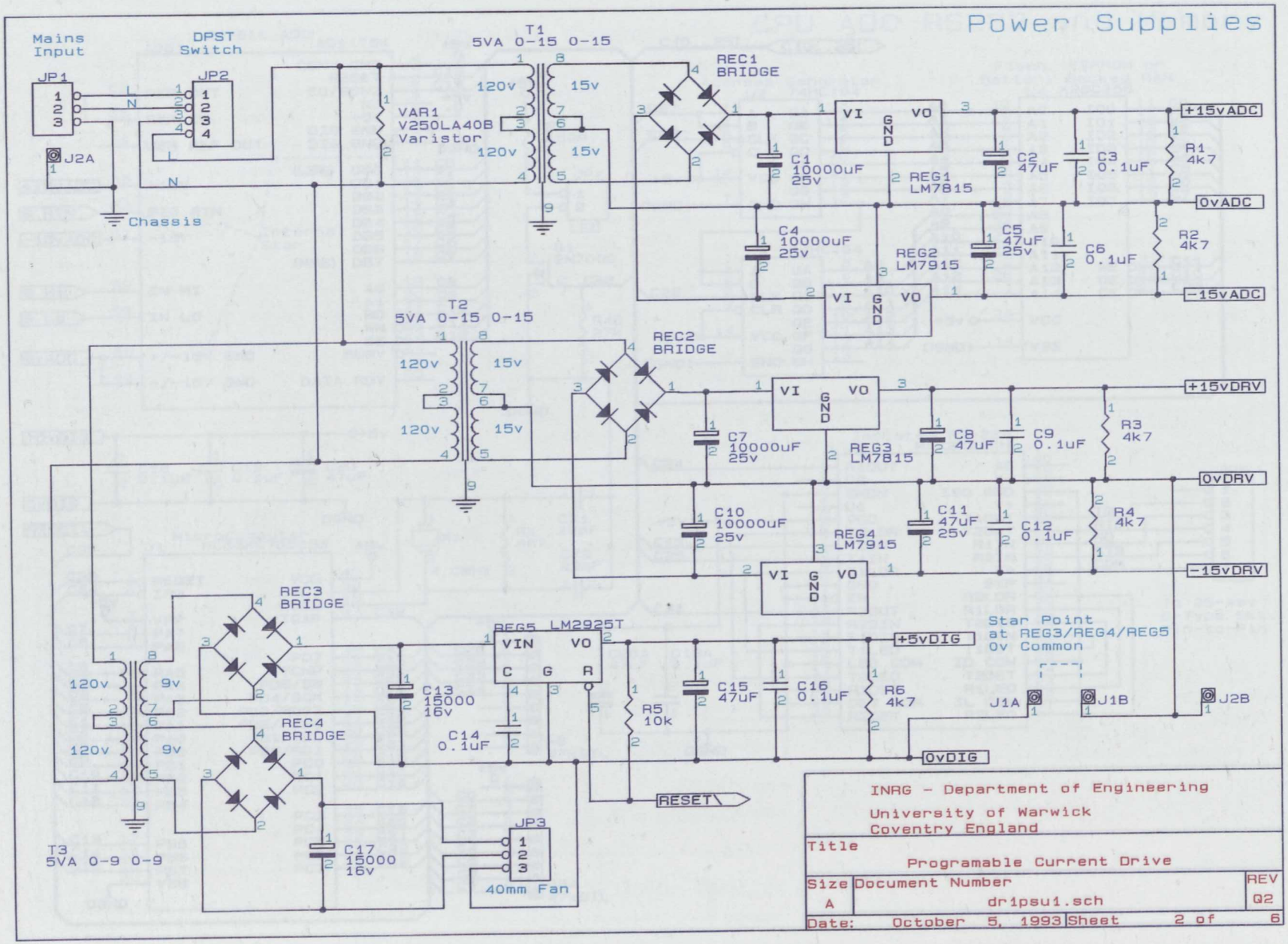
[7.9] Linear Technology Corporation: "Data sheet for LTC1043 - Dual precision instrumentation switched capacitor building block", p 11-15 to 11-30, 1990 data book.

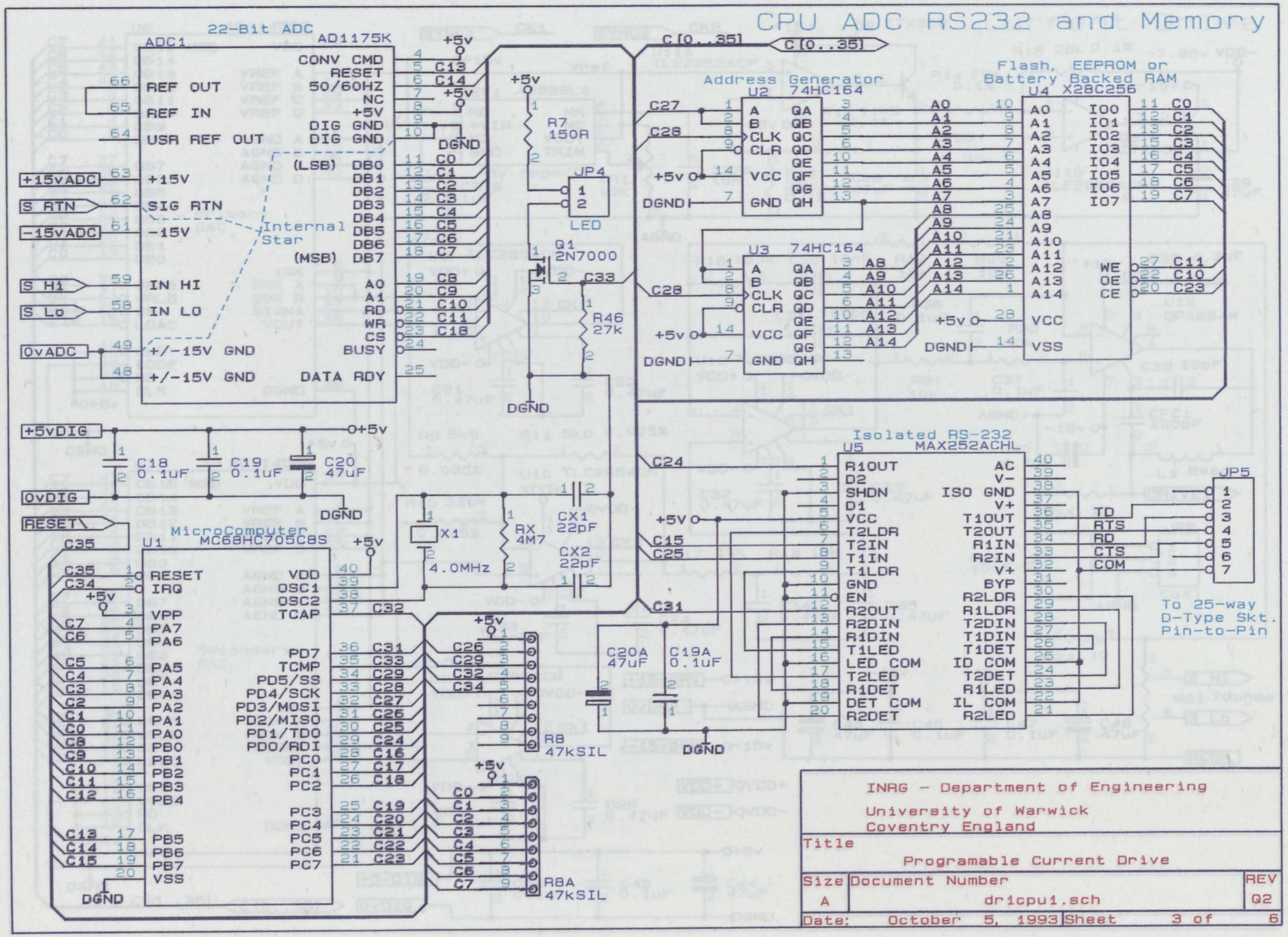
[7.10] Xilinx - The programmable gate array company. 2100 Logic Drive, San Jose, CA 95124.

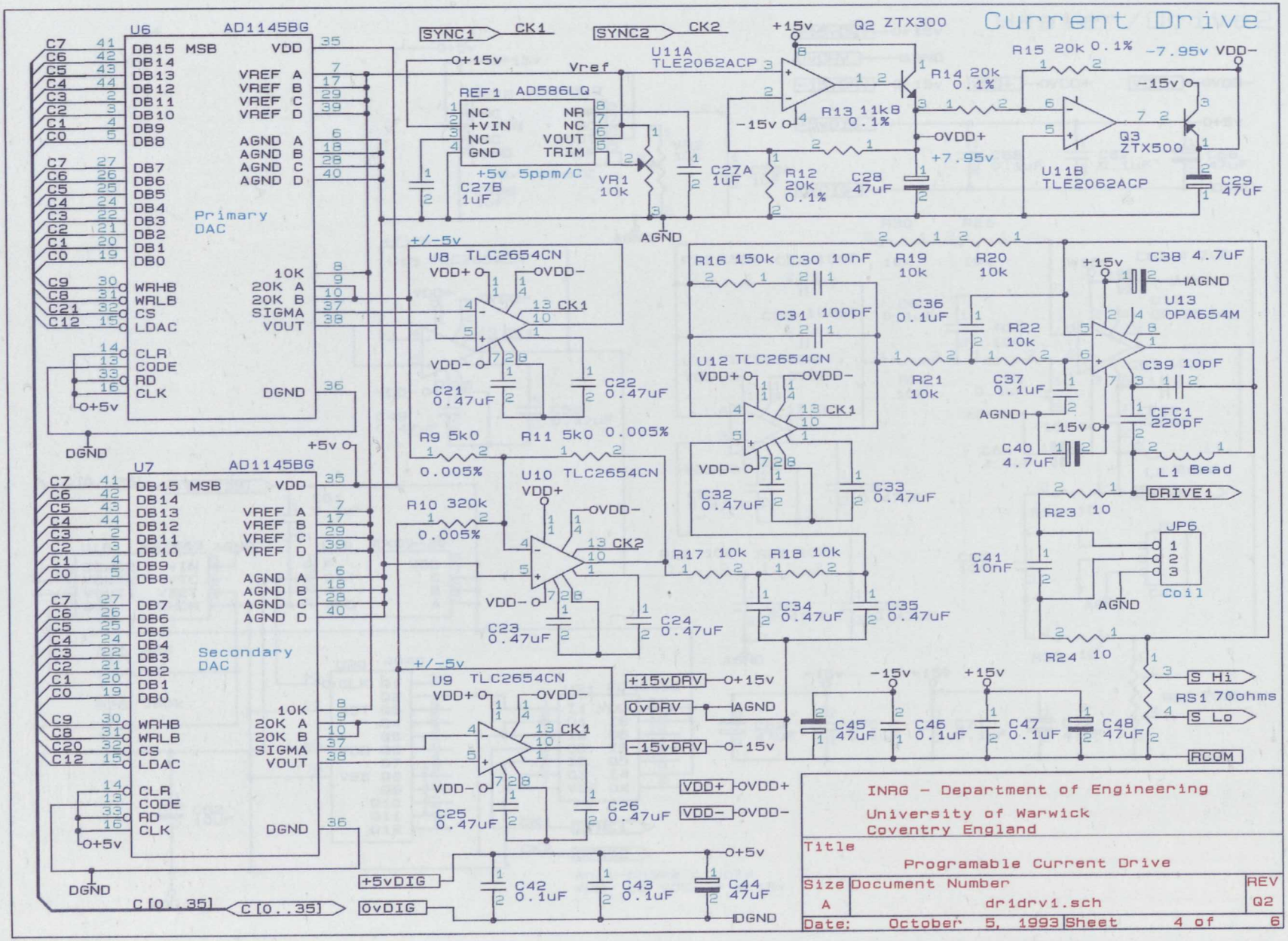
[7.11] Atmel corporation : "Field programmable gate arrays AT6000 series", *Configurable logic, design and application book, 1994/5,* p2-1 to 2-35.

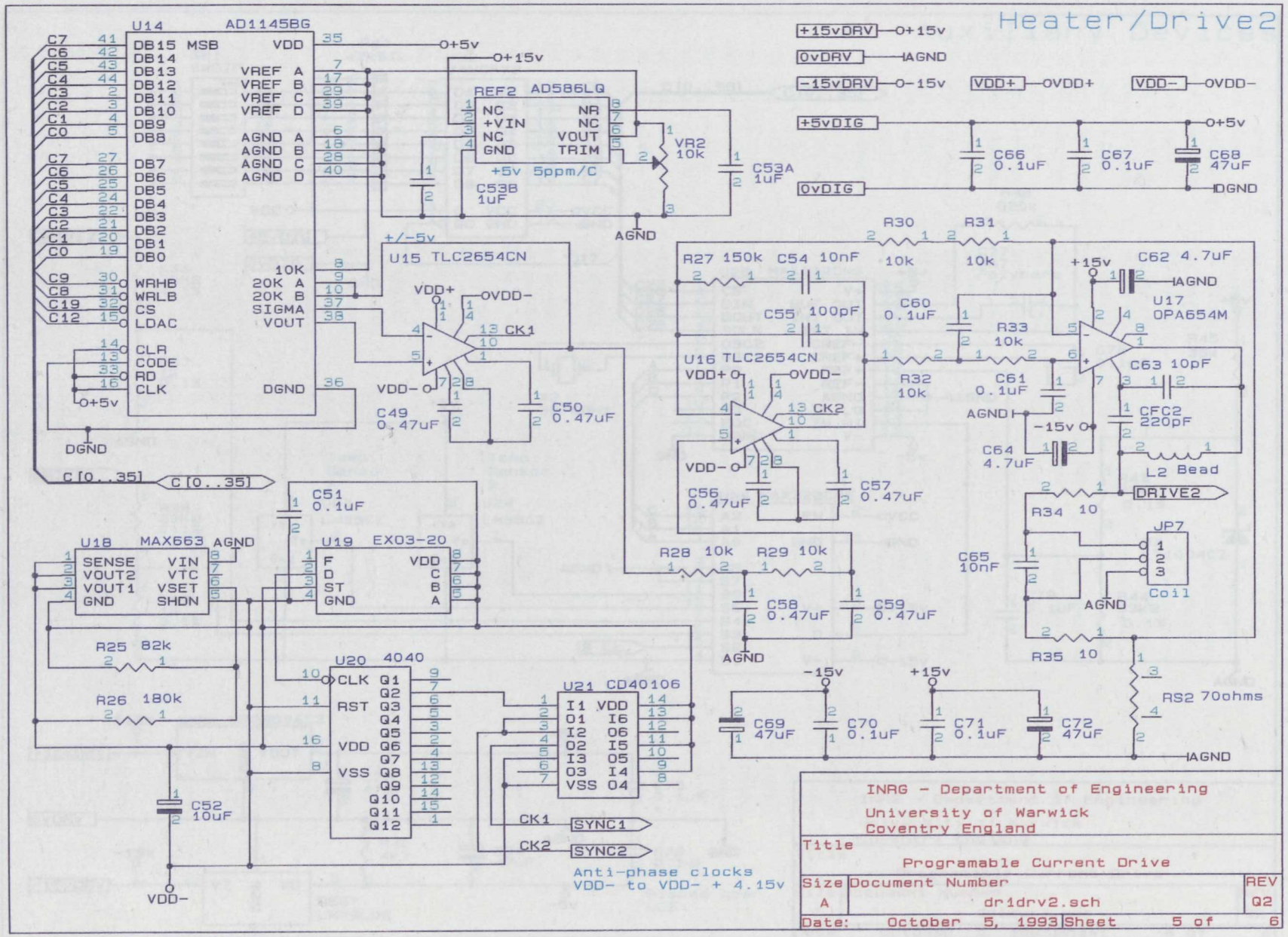
[8.1] Bede scientific instruments: "XIP control program", Linsey Park, Bowburn, Durham, England DH6 5PF

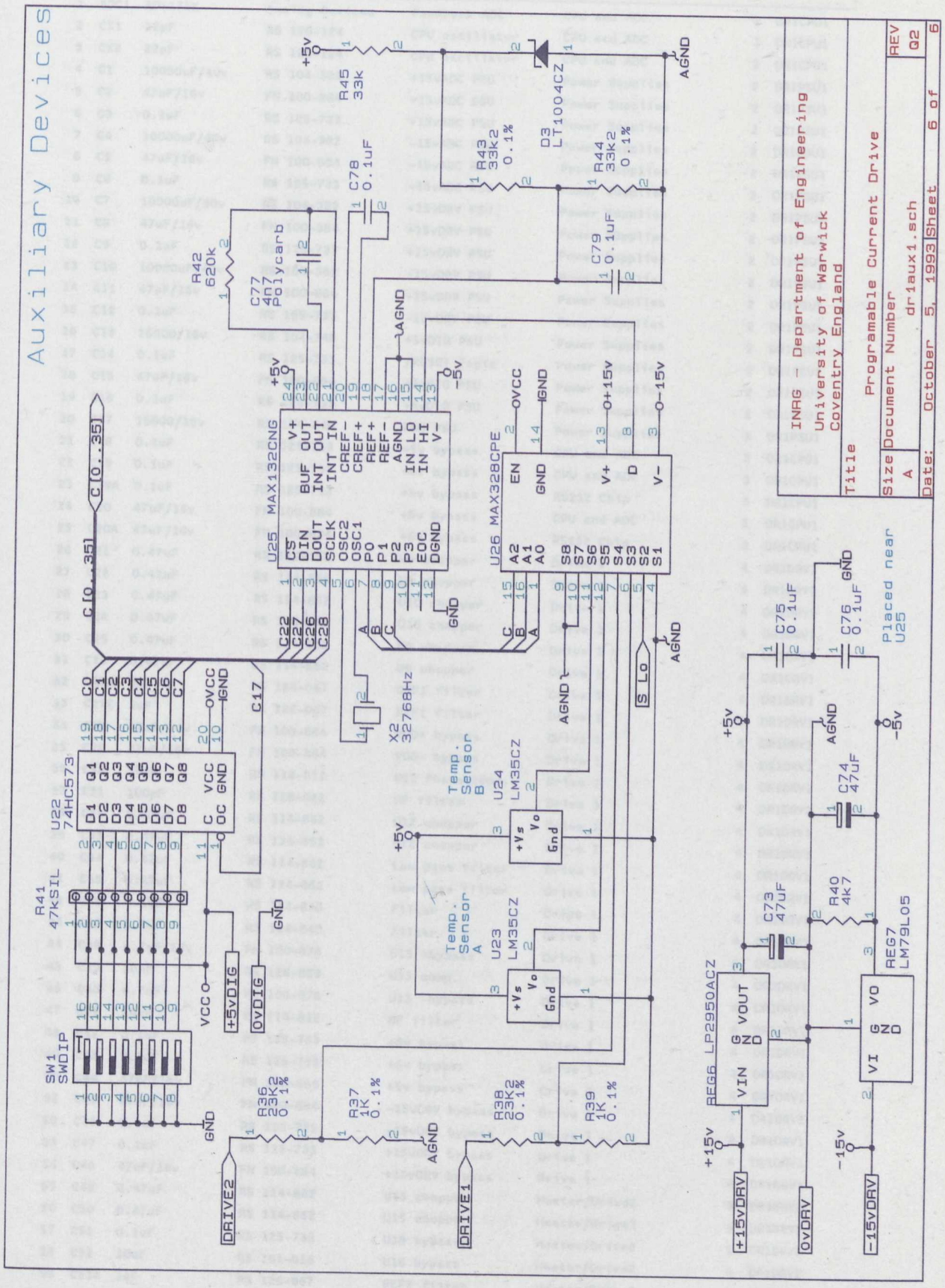

Block Diagram
Heater/Drive2
DRIVE2
VDD+
VDD-
+15vDRV
0vDRV
-15vDRV
+5vDIG
0vDIG
C[0..35]
SYNC2
SYNC1
Auxiliary Devices
DRIVE2
DRIVE1
S Lo
+15vDRV
0vDRV
-15vDRV
+5vDIG
0vDIG
C[0..35]
CPU and ADC
S Hi
S Lo
S RTN
+15vADC
0vADC
-15vADC
+5vDIG
0vDIG
C[0..35]
RESET\
Power Supplies
+15vADC
0vADC
-15vADC
+15vDRV
0vDRV
-15vDRV
+5vDIG
0vDIG
RESET\
Drive 1
S Hi
S Lo
DRIVE1
RCOM
+15vDRV
0vDRV
-15vDRV
+5vDIG
0vDIG
C[0..35]
VDD+
VDD-
SYNC1
SYNC2
INRG - Department of Engineering
University of Warwick
Coventry England
Title
Programable Current Drive
Size
A
Document Number
dr1m1.sch
REV
Q2
Date: October 5, 1993
Sheet 1 of 6


Power Supplies
Mains Input
DPST Switch
JP1
JP2
J2A
Chassis
T1
5VA 0-15 0-15
VAR1
V250LA40B
Varistor
REC1 BRIDGE
C1 10000uF 25v
REG1 LM7815
C2 47uF
C3 0.1uF
R1 4k7
+15vADC
0vADC
C4 10000uF 25v
REG2 LM7915
C5 47uF 25v
C6 0.1uF
R2 4k7
-15vADC
T2
5VA 0-15 0-15
REC2 BRIDGE
C7 10000uF 25v
REG3 LM7815
C8 47uF
C9 0.1uF
R3 4k7
+15vDRV
0vDRV
C10 10000uF 25v
REG4 LM7915
C11 47uF 25v
C12 0.1uF
R4 4k7
-15vDRV
REC3 BRIDGE
REC4 BRIDGE
REG5 LM2925T
VIN VO C G R
C13 15000 16v
C14 0.1uF
R5 10k
C15 47uF
C16 0.1uF
R6 4k7
+5vDIG
0vDIG
Star Point at REG3/REG4/REG5 0v Common
J1A
J1B
J2B
RESET
T3
5VA 0-9 0-9
C17 15000 16v
JP3
40mm Fan
INRG - Department of Engineering
University of Warwick
Coventry England
Title
Programable Current Drive
Size A
Document Number
dripsu1.sch
REV Q2
Date: October 5, 1993
Sheet 2 of 6


CPU ADC RS232 and Memory
22-Bit ADC
ADC1
AD1175K
CONV CMD
RESET
50/60HZ
NC
+5V
DIG GND
DIG GND
REF OUT
REF IN
USR REF OUT
(LSB) DB0
DB1
DB2
DB3
DB4
DB5
DB6
(MSB) DB7
+15V
SIG RTN
-15V
Internal Star
IN HI
IN LO
+/-15V GND
+/-15V GND
DATA RDY
BUSY
WR
RD
CS
A0
A1
+15vADC
S RTN
-15vADC
S Hi
S Lo
OvADC
DGND
+5V
R7 150R
JP4
LED
Q1 2N7000
C33
R46 27k
C[0..35]
Address Generator
U2 74HC164
U3 74HC164
Flash, EEPROM or Battery Backed RAM
U4 X28C256
IO0
IO1
IO2
IO3
IO4
IO5
IO6
IO7
WE
OE
CE
VCC
VSS
+5vDIG
OvDIG
C18 0.1uF
C19 0.1uF
C20 47uF
RESET
MicroComputer
U1
MC68HC705C8S
X1
4.0MHz
RX 4M7
CX1 22pF
CX2 22pF
C35
C32
Isolated RS-232
U5
MAX252ACHL
R1OUT
D2
SHDN
D1
VCC
T2LDR
T2IN
T1IN
T1LDR
GND
EN
R2OUT
R2DIN
R1DIN
T1LED
LED COM
T2LED
R1DET
DET COM
R2DET
AC
V-
ISO GND
V+
T1OUT
T2OUT
R1IN
R2IN
BYP
R2LDR
R1LDR
T2DIN
T1DIN
T1DET
ID COM
T2DET
R1LED
IL COM
R2LED
C20A 47uF
C19A 0.1uF
JP5
TD
RTS
RD
CTS
COM
To 25-way D-Type Skt. Pin-to-Pin
R8 47kSIL
R8A 47kSIL
INRG - Department of Engineering
University of Warwick
Coventry England
Title
Programable Current Drive
Size A
Document Number
dricpu1.sch
REV Q2
Date: October 5, 1993
Sheet 3 of 6


Current Drive
U6 AD1145BG
U7 AD1145BG
Primary DAC
Secondary DAC
REF1 AD586LQ
+5v 5ppm/C
U8 TLC2654CN
U9 TLC2654CN
U10 TLC2654CN
U11A TLE2062ACP
U11B TLE2062ACP
U12 TLC2654CN
U13 OPA654M
Q2 ZTX300
Q3 ZTX500
SYNC1
CK1
SYNC2
CK2
Vref
AGND
DGND
+15v
-15v
-7.95v VDD-
OVDD+
+7.95v
+/-5v
+5v
+15vDRV
OvDRV
-15vDRV
+5vDIG
OvDIG
VDD+
VDD-
C[0..35]
DRIVE1
S Hi
S Lo
RS1 70ohms
RCOM
Coil
INRG - Department of Engineering
University of Warwick
Coventry England
Title
Programable Current Drive
Size A
Document Number
dr1drv1.sch
REV Q2
Date: October 5, 1993
Sheet 4 of 6


Heater/Drive2
U14 AD1145BG
DB15 MSB
DB14
DB13
DB12
DB11
DB10
DB9
DB8
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0
WRHB
WRLB
CS
LDAC
CLR
CODE
RD
CLK
VDD
VREF A
VREF B
VREF C
VREF D
AGND A
AGND B
AGND C
AGND D
10K
20K A
20K B
SIGMA
VOUT
DGND
O+5v
O+15v
REF2 AD586LQ
NC
+VIN
NC
GND
NR
NC
VOUT
TRIM
+5v 5ppm/C
C53B 1uF
C53A 1uF
VR2 10k
AGND
+/-5v
U15 TLC2654CN
VDD+
OVDD-
CK1
VDD-
C49 0.47uF
C50 0.47uF
C [0..35]
C51 0.1uF
U18 MAX663 AGND
SENSE
VOUT2
VOUT1
GND
VIN
VTC
VSET
SHDN
U19 EX03-20
F
D
ST
GND
VDD
C
B
A
R25 82k
R26 180k
C52 10uF
U20 4040
CLK
RST
VDD
VSS
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q11
Q12
U21 CD40106
I1
O1
I2
O2
I3
O3
VSS
VDD
I6
O6
I5
O5
I4
O4
CK1
CK2
SYNC1
SYNC2
Anti-phase clocks
VDD- to VDD- + 4.15v
+15vDRV O+15v
OvDRV AGND
-15vDRV O-15v
VDD+ OVDD+
VDD- OVDD-
+5vDIG
OvDIG
DGND
C66 0.1uF
C67 0.1uF
C68 47uF
R27 150k
C54 10nF
C55 100pF
U16 TLC2654CN
R28 10k
R29 10k
C58 0.47uF
C59 0.47uF
R30 10k
R31 10k
C60 0.1uF
R32 10k
R33 10k
C61 0.1uF
C56 0.47uF
C57 0.47uF
+15v
C62 4.7uF
U17 OPA654M
C63 10pF
CFC2 220pF
L2 Bead
C64 4.7uF
-15v
DRIVE2
R34 10
JP7
C65 10nF
Coil
R35 10
RS2 70ohms
C69 47uF
C70 0.1uF
C71 0.1uF
C72 47uF
INRG - Department of Engineering
University of Warwick
Coventry England
Title
Programable Current Drive
Size A
Document Number dr1drv2.sch
REV Q2
Date: October 5, 1993 Sheet 5 of 6


Auxiliary Devices
INAG - Department of Engineering
University of Warwick
Coventry England
Programable Current Drive
Title
Size A
Document Number dr1aux1.sch
REV Q2
Date: October 5, 1993 Sheet 6 of 6
SW1 SWDIP
R41 47kSIL
U22 74HC573
C[0..35]
C17
U25 MAX132CNG
X2 32768Hz
C22 C27 C26 C28
R42 620k
C77 4n7 Polycarb
C78 0.1uF
R45 33k
R43 33k2 0.1%
R44 33k2 0.1%
D3 LT1004CZ
C79 0.1uF
U26 MAX328CPE
R36 23K2 0.1%
R37 1K 0.1%
R38 23K2 0.1%
R39 1K 0.1%
U23 LM35CZ
U24 LM35CZ
Temp. Sensor A
Temp. Sensor B
REG6 LP2950ACZ
REG7 LM79L05
C73 47uF
C74 47uF
C75 0.1uF
C76 0.1uF
R40 4k7
Placed near U25
DRIVE1
DRIVE2
+5vDIG
OvDIG
+15vDRV
OvDRV
-15vDRV
AGND
GND
+5v
-5v
+15v
-15v

Programable Current Drive Revised: 30 November 1993 From DR1M1Q1A.CAL

#	Ref	Part	Supplier	Function	Sheetname	No.	File.sch
1	ADC1	AD1175K	Analog Devices	Feedback ADC	CPU and ADC	3	DR1CPU1
2	CX1	22pF	RS 126-124	CPU oscillator	CPU and ADC	3	DR1CPU1
3	CX2	22pF	RS 126-124	CPU oscillator	CPU and ADC	3	DR1CPU1
4	C1	10000uF/40v	RS 104-382	+15vADC PSU	Power Supplies	2	DR1PSU1
5	C2	47uF/16v	FN 100-884	+15vADC PSU	Power Supplies	2	DR1PSU1
6	C3	0.1uF	RS 125-733	+15vADC PSU	Power Supplies	2	DR1PSU1
7	C4	10000uF/40v	RS 104-382	-15vADC PSU	Power Supplies	2	DR1PSU1
8	C5	47uF/16v	FN 100-884	-15vADC PSU	Power Supplies	2	DR1PSU1
9	C6	0.1uF	RS 125-733	-15vADC PSU	Power Supplies	2	DR1PSU1
10	C7	10000uF/40v	RS 104-382	+15vDRV PSU	Power Supplies	2	DR1PSU1
11	C8	47uF/16v	FN 100-884	+15vDRV PSU	Power Supplies	2	DR1PSU1
12	C9	0.1uF	RS 125-733	+15vDRV PSU	Power Supplies	2	DR1PSU1
13	C10	10000uF/40v	RS 104-382	-15vDRV PSU	Power Supplies	2	DR1PSU1
14	C11	47uF/16v	FN 100-884	-15vDRV PSU	Power Supplies	2	DR1PSU1
15	C12	0.1uF	RS 125-733	-15vDRV PSU	Power Supplies	2	DR1PSU1
16	C13	15000/16v	RS 104-348	+5vDIG PSU	Power Supplies	2	DR1PSU1
17	C14	0.1uF	RS 125-733	/RESET logic	Power Supplies	2	DR1PSU1
18	C15	47uF/16v	FN 100-884	+5vDIG PSU	Power Supplies	2	DR1PSU1
19	C16	0.1uF	RS 125-733	+5vDIG PSU	Power Supplies	2	DR1PSU1
20	C17	15000/16v	RS 104-348	FAN PSU	Power Supplies	2	DR1PSU1
21	C18	0.1uF	RS 125-733	+5v bypass	CPU and ADC	3	DR1CPU1
22	C19	0.1uF	RS 125-733	+5v bypass	CPU and ADC	3	DR1CPU1
23	C19A	0.1uF	RS 125-733	+5v bypass	RS232 Chip	3	DR1CPU1
24	C20	47uF/16v	FN 100-884	+5v bypass	CPU and ADC	3	DR1CPU1
25	C20A	47uF/16v	FN 100-884	+5v bypass	RS232 Chip	3	DR1CPU1
26	C21	0.47uF	RS 114-862	U8 chopper	Drive 1	4	DR1DRV1
27	C22	0.47uF	RS 114-862	U8 chopper	Drive 1	4	DR1DRV1
28	C23	0.47uF	RS 114-862	U10 chopper	Drive 1	4	DR1DRV1
29	C24	0.47uF	RS 114-862	U10 chopper	Drive 1	4	DR1DRV1
30	C25	0.47uF	RS 114-862	U9 chopper	Drive 1	4	DR1DRV1
31	C26	0.47uF	RS 114-862	U9 chopper	Drive 1	4	DR1DRV1
32	C27A	1uF	RS 126-067	REF1 filter	Drive 1	4	DR1DRV1
33	C27B	1uF	RS 126-067	REF1 filter	Drive 1	4	DR1DRV1
34	C28	47uF/16v	FN 100-884	VDD+ bypass	Drive 1	4	DR1DRV1
35	C29	47uF/16v	FN 100-884	VDD- bypass	Drive 1	4	DR1DRV1
36	C30	10nF	RS 114-812	U12 Phase comp.	Drive 1	4	DR1DRV1
37	C31	100pF	RS 126-922	HF filter	Drive 1	4	DR1DRV1
38	C32	0.47uF	RS 114-862	U12 chopper	Drive 1	4	DR1DRV1
39	C33	0.47uF	RS 114-862	U12 chopper	Drive 1	4	DR1DRV1
40	C34	0.47uF	RS 114-862	Low pass filter	Drive 1	4	DR1DRV1
41	C35	0.47uF	RS 114-862	Low pass filter	Drive 1	4	DR1DRV1
42	C36	0.1uF	RS 114-840	Filter	Drive 1	4	DR1DRV1
43	C37	0.1uF	RS 114-840	Filter	Drive 1	4	DR1DRV1
44	C38	4.7uF/16v	FN 100-878	U13 +bypass	Drive 1	4	DR1DRV1
45	C39	10pF	RS 126-809	U13 comp.	Drive 1	4	DR1DRV1
46	C40	4.7uF	FN 100-878	U13 -bypass	Drive 1	4	DR1DRV1
47	C41	10nF	RS 114-812	HF filter	Drive 1	4	DR1DRV1
48	C42	0.1uF	RS 125-733	+5v bypass	Drive 1	4	DR1DRV1
49	C43	0.1uF	RS 125-733	+5v bypass	Drive 1	4	DR1DRV1
50	C44	47uF/16v	FN 100-884	+5v bypass	Drive 1	4	DR1DRV1
51	C45	47uF/16v	FN 100-884	-15vDRV bypass	Drive 1	4	DR1DRV1
52	C46	0.1uF	RS 125-733	-15vDRV bypass	Drive 1	4	DR1DRV1
53	C47	0.1uF	RS 125-733	+15vDRV bypass	Drive 1	4	DR1DRV1
54	C48	47uF/16v	FN 100-884	+15vDRV bypass	Drive 1	4	DR1DRV1
55	C49	0.47uF	RS 114-862	U15 chopper	Heater/Drive2	5	DR1DRV2
56	C50	0.47uF	RS 114-862	U15 chopper	Heater/Drive2	5	DR1DRV2
57	C51	0.1uF	RS 125-733	U18 bypass	Heater/Drive2	5	DR1DRV2
58	C52	10uF	RS 101-816	U18 bypass	Heater/Drive2	5	DR1DRV2
59	C53A	1uF	RS 126-067	REF2 filter	Heater/Drive2	5	DR1DRV2
60	C53B	1uF	RS 126-067	REF2 filter	Heater/Drive2	5	DR1DRV2
61	C54	10nF	RS 114-812	Phase comp.	Heater/Drive2	5	DR1DRV2

Programable Current Drive Revised: 30 November 1993 From DR1M1Q1A.CAL

#	Ref	Part	Supplier	Function	Sheetname	No.	File.sch
62	C55	100pF	RS 126-922	HF filter	Heater/Drive2	5	DR1DRV2
63	C56	0.47uF	RS 114-862	U12 chopper	Heater/Drive2	5	DR1DRV2
64	C57	0.47uF	RS 114-862	U12 chopper	Heater/Drive2	5	DR1DRV2
65	C58	0.47uF	RS 114-862	Low pass filter	Heater/Drive2	5	DR1DRV2
66	C59	0.47uF	RS 114-862	Low pass filter	Heater/Drive2	5	DR1DRV2
67	C60	0.1uF	RS 114-840	Filter	Heater/Drive2	5	DR1DRV2
68	C61	0.1uF	RS 114-840	Filter	Heater/Drive2	5	DR1DRV2
69	C62	4.7uF/16v	FN 100-878	U13 +bypass	Heater/Drive2	5	DR1DRV2
70	C63	10pF	RS 126-809	U13 comp.	Heater/Drive2	5	DR1DRV2
71	C64	4.7uF/16v	FN 100-878	U13 -bypass	Heater/Drive2	5	DR1DRV2
72	C65	10nF	RS 114-812	HF filter	Heater/Drive2	5	DR1DRV2
73	C66	0.1uF	RS 125-733	+5v bypass	Heater/Drive2	5	DR1DRV2
74	C67	0.1uF	RS 125-733	+5v bypass	Heater/Drive2	5	DR1DRV2
75	C68	47uF/16v	FN 100-884	+5v bypass	Heater/Drive2	5	DR1DRV2
76	C69	47uF/16v	FN 100-884	-15vDRV bypass	Heater/Drive2	5	DR1DRV2
77	C70	0.1uF	RS 125-733	-15vDRV bypass	Heater/Drive2	5	DR1DRV2
78	C71	0.1uF	RS 125-733	+15vDRV bypass	Heater/Drive2	5	DR1DRV2
79	C72	47uF/16v	FN 100-884	+15vDRV bypass	Heater/Drive2	5	DR1DRV2
80	C73	47uF/16v	FN 100-884	U22 +bypass	Auxiliary Devices	6	DR1AUX1
81	C74	47uF/16v	FN 100-884	U22 -bypass	Auxiliary Devices	6	DR1AUX1
82	C75	0.1uF	RS 125-733	U22 +bypass	Auxiliary Devices	6	DR1AUX1
83	C76	0.1uF	RS 125-733	U22 -bypass	Auxiliary Devices	6	DR1AUX1
84	C77	4n7	RS 115-736	U22 Int. C.	Auxiliary Devices	6	DR1AUX1
85	C78	0.1uF	RS 114-840	U22 Ref. C	Auxiliary Devices	6	DR1AUX1
86	C79	0.1uF	RS 114-840	Vref filter	Auxiliary Devices	6	DR1AUX1
87	CFC1	220pF	RS 113-285	U13 freq. comp.	Drive 1	4	DR1DRV1
88	CFC2	220pF	RS 113-285	U17 freq. comp.	Drive 1	4	DR1DRV1
89	D3	LT1004CZ	Linear Technology	Bandgap Ref.	Auxiliary Devices	6	DR1AUX1
90	JP1	HEADER 3	3 pads for mains	Mains input	Power Supplies	2	DR1PSU1
91	JP2	HEADER 4	4 pads for mains switch	Mains switch	Power Supplies	2	DR1PSU1
92	JP3	HEADER 2	2 pin 0.1" pitch	Fan connector	Power Supplies	2	DR1PSU1
93	JP4	HEADER 2	2 pin 0.1" pitch	LED connector	CPU and ADC	3	DR1CPU1
94	JP5	HEADER 7	7 pin 0.1" pitch	Serial link	CPU and ADC	3	DR1CPU1
95	JP6	HEADER 3	3 pin 0.1" pitch	Drive 1	Drive 1	4	DR1DRV1
96	JP7	HEADER 3	3 pin 0.1" pitch	Drive 2	Heater/Drive2	5	DR1DRV2
97	J1A	TP	no component	Star point	Power Supplies	2	DR1PSU1
98	J1B	TP	no component	Star pooint	Power Supplies	2	DR1PSU1
99	L1	Bead	FN 108-269	U13 HF suppression	Drive 1	4	DR1DRV1
100	L2	Bead	FN 108-269	U17 HF suppression	Heater/Drive2	5	DR1DRV2
101	Q1	2N7000	RS 296-043	LED driver	CPU and ADC	3	DR1CPU1
102	Q2	ZTX300	RS 294-457	VDD+ booster	Drive 1	4	DR1DRV1
103	Q3	ZTX500	RS 294-463	VDD- booster	Drive 1	4	DR1DRV1
104	REC1	BRIDGE	FN 1KAB20E	ADC rectifiers	Power Supplies	2	DR1PSU1
105	REC2	BRIDGE	FN 1KAB20E	DRV rectifiers	Power Supplies	2	DR1PSU1
106	REC3	BRIDGE	FN 1KAB20E	DIG rectifiers	Power Supplies	2	DR1PSU1
107	REC4	BRIDGE	FN 1KAB20E	FAN rectifiers	Power Supplies	2	DR1PSU1
108	REF1	AD586LQ	Analog Devices	Drive1 Vref	Drive 1	4	DR1DRV1
109	REF2	AD586LQ	Analog Devices	Drive2 Vref	Heater/Drive2	5	DR1DRV2
110	REG1	LM7815CT	RS 648-444	+15vADC regulator	Power Supplies	2	DR1PSU1
111	REG2	LM7915CT	RS 648-472	-15vADC regulator	Power Supplies	2	DR1PSU1
112	REG3	LM7815CT	RS 648-444	+15vDRV regulator	Power Supplies	2	DR1PSU1
113	REG4	LM7915CT	RS 648-472	-15vDRV regulator	Power Supplies	2	DR1PSU1
114	REG5	LM2925T	RS 648-545	+5vDIG regulator	Power Supplies	2	DR1PSU1
115	REG6	LP2950ACZ	FN 2950ACZ5	U22 +5v regulator	Auxiliary Devices	6	DR1AUX1
116	REG7	LM79L05	FN 79L05ACZ	U22 -5v regulator	Auxiliary Devices	6	DR1AUX1
117	RS1	70ohms	N/A	External Coil	Drive 1	4	DR1DRV1
118	RS2	70ohms	N/A	External Coil	Heater/Drive2	5	DR1DRV2
119	RX	4M7	RS 135-645	CPU oscillator	CPU and ADC	3	DR1CPU1
120	R1	4k7	RS 148-663	+15vADC load	Power Supplies	2	DR1PSU1
121	R2	4k7	RS 148-663	-15vADC load	Power Supplies	2	DR1PSU1

Programable Current Drive Revised: 30 November 1993 From DR1M1Q1A.CAL

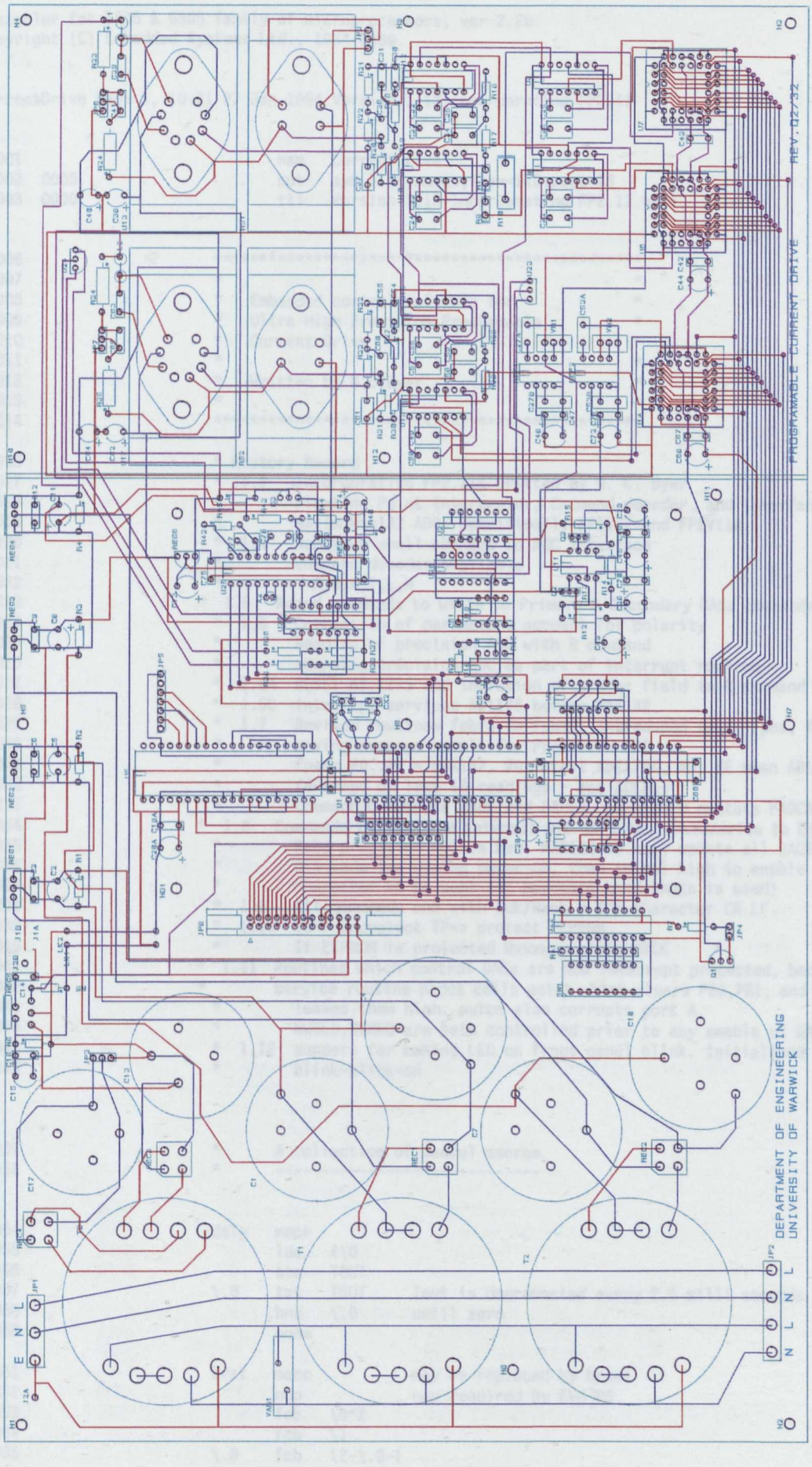
#	Ref	Part	Supplier	Function	Sheetname	No.	File.sch
122	R3	4k7	RS 148-663	+15vDRV load	Power Supplies	2	DR1PSU1
123	R4	4k7	RS 148-663	-15vDRV load	Power Supplies	2	DR1PSU1
124	R5	10k	RS 148-736	/RESET pullup	Power Supplies	2	DR1PSU1
125	R6	4k7	RS 148-663	+5vDIG load	Power Supplies	2	DR1PSU1
126	R7	150R	RS 148-304	LED I-limit	CPU and ADC	3	DR1CPU1
127	R8	47kSIL	RS 140-990	CPU pullups	CPU and ADC	3	DR1CPU1
128	R8A	47kSIL	RS 140-990	CPU pullups	CPU and ADC	3	DR1CPU1
129	R9	5k0	Vishay S102K	Primary DAC	Drive 1	4	DR1DRV1
		5k0 0.005%		weight			
130	R10	320k	Vishay S102K	Secondary DAC	Drive 1	4	DR1DRV1
		320k0 0.005%		weight			
131	R11	5k0	Vishay S102K	U10 feedback	Drive 1	4	DR1DRV1
		5k0 0.005%					
132	R12	20k	RS 167-018	VDD+ gain	Drive 1	4	DR1DRV1
133	R13	11k8	RS 166-790	VDD+ gain	Drive 1	4	DR1DRV1
134	R14	20k	RS 167-018	VDD- gain	Drive 1	4	DR1DRV1
135	R15	20k	RS 167-018	VDD- gain	Drive 1	4	DR1DRV1
136	R16	150k	RS 149-026	U12 Phase comp.	Drive 1	4	DR1DRV1
137	R17	10k	RS 148-736	Low pass filter	Drive 1	4	DR1DRV1
138	R18	10k	RS 148-736	Low pass filter	Drive 1	4	DR1DRV1
139	R19	10k	RS 148-736	DC feedback	Drive 1	4	DR1DRV1
140	R20	10k	RS 148-736	DC feedback	Drive 1	4	DR1DRV1
141	R21	10k	RS 148-736	Filter	Drive 1	4	DR1DRV1
142	R22	10k	RS 148-736	Filter	Drive 1	4	DR1DRV1
143	R23	10	RS 148-017	Current limit	Drive 1	4	DR1DRV1
144	R24	10	RS 148-017	Current limit	Drive 1	4	DR1DRV1
145	R25	82k	RS 148-950	Voltage set	Heater/Drive2	5	DR1DRV2
146	R26	180k	RS 149-048	Voltage set	Heater/Drive2	5	DR1DRV2
147	R27	150k	RS 149-026	U16 Phase comp.	Heater/Drive2	5	DR1DRV2
148	R28	10k	RS 148-736	Low pass filter	Heater/Drive2	5	DR1DRV2
149	R29	10k	RS 148-736	Low pass filter	Heater/Drive2	5	DR1DRV2
150	R30	10k	RS 148-736	DC feedback	Heater/Drive2	5	DR1DRV2
151	R31	10k	RS 148-736	DC feedback	Heater/Drive2	5	DR1DRV2
152	R32	10k	RS 148-736	Filter	Heater/Drive2	5	DR1DRV2
153	R33	10k	RS 148-736	Filter	Heater/Drive2	5	DR1DRV2
154	R34	10	RS 148-017	Current limit	Heater/Drive2	5	DR1DRV2
155	R35	10	RS 148-017	Current limit	Heater/Drive2	5	DR1DRV2
156	R36	23K2	RS 167-074	Voltage divider	Auxiliary Devices	6	DR1AUX1
157	R37	1K	RS 165-769	Voltage divider	Auxiliary Devices	6	DR1AUX1
158	R38	23K2	RS 167-074	Voltage divider	Auxiliary Devices	6	DR1AUX1
159	R39	1K	RS 165-769	Voltage divider	Auxiliary Devices	6	DR1AUX1
160	R40	4k7	RS 148-663	U22 +5v load	Auxiliary Devices	6	DR1AUX1
161	R41	47kSIL	RS 140-990	U19 pullup	Auxiliary Devices	6	DR1AUX1
162	R42	620k	RS 149-177	U22 Int. R	Auxiliary Devices	6	DR1AUX1
163	R43	33k2 0.1%	RS 167-226	Vref divider	Auxiliary Devices	6	DR1AUX1
164	R44	33k2 0.1%	RS 167-226	Vref divider	Auxiliary Devices	6	DR1AUX1
165	R45	33k	RS 148-859	D3 bias	Auxiliary Devices	6	DR1AUX1
166	R46	27k	RS 148-837	Q1 gate pull-down	CPU and ADC	3	DR1CPU1
167	SW1	SWDIP	RS 337-560	Setup switches	Auxiliary Devices	6	DR1AUX1
168	T1	5VA 0-15 0-15	RS 207-807	ADC PSU	Power Supplies	2	DR1PSU1
169	T2	5VA 0-15 0-15	RS 207-807	Drive PSU	Power Supplies	2	DR1PSU1
170	T3	5VA 0-9 0-9	RS 207-970	Digital/Fan PSU	Power Supplies	2	DR1PSU1
171	U1	MC68HC705C8S	Motorola	Microprocessor	CPU and ADC	3	DR1CPU1
172	U2	74HC164	RS 301-347	Address Generator	CPU and ADC	3	DR1CPU1
173	U3	74HC164	RS 301-347	Address Generator	CPU and ADC	3	DR1CPU1
174	U4	X28C256	RS 656-041	Memory	CPU and ADC	3	DR1CPU1
175	U5	MAX252ACHL	Maxim	Isolated RS232	CPU and ADC	3	DR1CPU1
176	U6	AD1145BG	Analog Devices	Primary DAC	Drive 1	4	DR1DRV1
177	U7	AD1145BG	Analog Devices	Secondary DAC	Drive 1	4	DR1DRV1
178	U8	TLC2654CN	FN TLC2654CN	U6 output	Drive 1	4	DR1DRV1
179	U9	TLC2654CN	FN TLC2654CN	U7 output	Drive 1	4	DR1DRV1

Programable Current Drive Revised: 30 November 1993 From DR1M1Q1A.CAL

#	Ref	Part	Supplier	Function	Sheetname	No.	File.sch
180	U10	TLC2654CN	FN TLC2654CN	Summing amplifier	Drive 1	4	DR1DRV1
181	U11A	TLE2062ACP	RS 264-765	VDD+ generator	Drive 1	4	DR1DRV1
182	U11B	TLE2062ACP	(A/B is dual pk.)	VDD- generator	Drive 1	4	DR1DRV1
183	U12	TLC2654CN	FN TLC2654CN	Output stage	Drive 1	4	DR1DRV1
184	U13	OPA654M	Burr Brown	Power Output	Drive 1	4	DR1DRV1
185	U14	AD1145BG	Analog Devices	Heater/Drive 2	Heater/Drive2	5	DR1DRV2
186	U15	TLC2654CN	FN TLC2654CN	U14 output	Heater/Drive2	5	DR1DRV2
187	U16	TLC2654CN	FN TLC2654CN	Output stage	Heater/Drive2	5	DR1DRV2
188	U17	OPA654M	Burr Brown	Power Output	Heater/Drive2	5	DR1DRV2
189	U18	MAX663CPA	Maxim	Voltage regulator	Heater/Drive2	5	DR1DRV2
190	U19	EXO3-20	FN 221-764	Oscillator module	Heater/Drive2	5	
191	U20	CD4040BCN	FN CD4040CN	Divider	Heater/Drive2	5	
192	U21	CD40106BCN	FN CD40106BCN	Inverter	Heater/Drive2	5	
193	U22	MC74HC573	RS 643-512	Buffer	Auxiliary Devices	6	DR1AUX1
194	U23	LM35CZ	FN LM35CZ	Temperature sensor	Auxiliary Devices	6	DR1AUX1
195	U24	LM35CZ	FN LM35CZ	Temperature sensor	Auxiliary Devices	6	DR1AUX1
196	U25	MAX132CNG	Maxim	Auxiliary ADC	Auxiliary Devices	6	DR1AUX1
197	U26	MAX328CPE	Maxim	Analog Multiplexer	Auxiliary Devices	6	DR1AUX1
198	VAR1	V250LA40B	FN V250LA40B	Varistor	Power Supplies	2	DR1PSU1
199	VR1	10k	Vishay 1260W 10k 5%	Trim resistor	Drive 1	4	DR1DRV1
200	VR2	10k	Vishay 1260W 10k 5%	Trim resistor	Heater/Drive2	5	DR1DRV2
201	X1	4.0MHz	RS 658-895	CPU crystal	CPU and ADC	3	DR1CPU1
202	X2	32768Hz	RS 304-447	Aux ADC crystal	Auxiliary Devices	6	DR1AUX1

* Present if REFx is AD586LQ

** Present if REFx is LT1027BCN8


PROGRAMMABLE CURRENT DRIVE
REV. 02/92
DEPARTMENT OF ENGINEERING
UNIVERSITY OF WARWICK
COMPONENT SIDE TRACK
M0560A

```
Assembler for 6805 & 6305 family of microprocessors, ver 2.2b.
Copyright (C) Embedded Systems Ltd., 1984-1988.


CurrentDrive Page 1, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11


00001                           nam   CurrentDrive
00002  0000                     opt   sym,list,nomex,llen=110,page=73
00003  0000                     ttl   Version 1.12 incorporating FP2.11 sful


00006                    ********************************************
00007                    *                                          *
00008                    *   Embedded control software for          *
00009                    *   Ultra High Precision Programable       *
00010                    *   Current Drive                          *
00011                    *                                          *
00012                    *   Written by D.C.Dyer                    *
00013                    *                                          *
00014                    ********************************************

00016                    * History Record
00017                    *  1.1   Incorporating FP2.11A written by D. C. Dyer
00018                    *        Floating Point Interpreter, Command decoder, and interface to
00019                    *        Maxim Max132 ADC. Specifically FP1V22 and FP2V11A
00020                    *  1.4   Numerous small changes to ADC interface
00021                    *        Command decoder available
00022                    *          A,B,H,I,L,M
00023                   *  1.5   Added commands to write to Primary & Secondary DACs commands P,S
00024                    *  1.6   Correction of parameters account for polarity
00025                    *        Reading of precision ADC with R command
00026                    *        scanning precision ADC as part of interrupt routine
00027                    *  1.6A  RESET of 1175 and inclusion of status field in R command
00028                    *  1.6C  interrupt services AD1175 before MAX132
00029                    *  1.7   Revised routines fpbra to fpbmi; Introduced fpbc, fpbs, fpdly
00030                    *        moving average sample for Precision ADC
00031                    *        fpadc #0..#7 = MAX132, #8 latest AD1175K, #9..#F mean AD1175K
00032                    *        (average of last 16 readings - see ADCBL)
00033                    *        Command C outputs PADC and PADCA, command D ouptuts PADCA
00034                   *  1.8   Commands to read the internal DIL switches, read/write to EEPROM
00035                    *        Read precision ADC in hex, MAX132 in hex, update all DACS
00036                    *        CTS made low during power-on, then in rdl high to enable
00037                    *        character so be sent (if hardware hand-shake is used)
00038                    *  1.9   All commands end with Ack/Nack = Y/N character CR LF
00039                    *  1.10  TU=> unprotect TP=> protect EEPROM
00040                    *        If EEPROM is protected Wxxxxyy gives NACK
00041                   *  1.11  Routines which control DACs are now interrupt protected, because
00042                   *        service routine padcs calls getst which clears PB0,PB1, and getr
00043                    *        leaves them high. putcm also corrupts port A.
00044                    *        NWRLB,NWRHB are both controlled prior to any enable of DAC
00045                    *  1.12  support for making LED on front panel blink. Initialised to
00046                    *        blink-blink-on



00051                    *      A collection of useful macros
00052                    *      -----------------------------


00054                    dely   macr
00055                           lda   #\0
00056                           sta   TOUT
00057                    \.0    tst   TOUT      Tout is decremented every 2.5 milli seconds
00058                           bne   \.0       until zero
00059                           endm

00061                    brst   macr            may be replaced by brset
00062                           nop             nop required by EVM-05
00063                           fcb   \0*2
00064                           fcb   \1
00065                    \.0    fcb   \2-\.0-1
```

```
00066                           endm

00068                   brcl    macr            may be replaced by brclr
00069                           nop             nop required by EVM-05
00070                           fcb   \0*2+1
00071                           fcb   \1
00072                   \.0     fcb   \2-\.0-1
00073                           endm

00075                   mul     macr            define multiply instruction
00076                           fcb   $42
00077                           endm

00079                   * partial product
00080                   pmul    macr            (arg2)= (arg2) + arg0 * arg1
00081                           lda   \0
00082                           ldx   \1
00083                           mul
00084                           add   \2
00085                           sta   \2
00086                           txa
00087                           adc   \2-1      high part at lower address
00088                           sta   \2-1
00089                           bcc   \.1       allow for carry to higher byte
00090                           inc   \2-2
00091                   \.1     equ   *
00092                           endm

00094                   fc      macr  c         defines one or more codes
00095                           fcb   \0*16+\1
00096                           ifgt  NARG-2
00097                           fcb   \2
00098                           endc
00099                           ifgt  NARG-3
00100                           fcb   \3
00101                           endc
00102                           endm

00104                   * used when defining 'fop's for interpreter jump table
00105                   fopdef macr
00106                   f\0     equ   (*-foptab)/2
00107                           fdb   fp\0
00108                           endm

00110                   * used when defining function group for interpreter jump table
00111                   fundef macr
00112                   f\0     equ   (*-funtab)/2
00113                           fdb   fp\0
00114                           endm

00116                   * used when defining branch group interpreter jump table
00117                   fbrdef macr
00118                   f\0     equ   (*-fbrtab)/2
00119                           fdb   fp\0
00120                           endm

00122                   *       constants
00123       0008        BS      equ   $08       Back Space
00124       000D        CR      equ   $0D       Carriage return
00125       000A        LF      equ   $0A       Line Feed
00126       0020        SP      equ   $20       Space

00128                   *       Port Addresses
00129       0000        PA      equ   0         Port A Data Register - I/O
00130       0001        PB      equ   1         Port B Data Register - I/O
00131       0002        PC      equ   2         Port C Data Register - I/O
00132       0003        PD      equ   3         Port D input register
00133       0004        DDRA    equ   4         Data Direction Register for port A
```

```
00134   0005    DDRB    equ   5       Data Direction Register for port B
00135   0006    DDRC    equ   6       Data Direction Register for port C
00136   0007    DDRD    equ   7       Data Direction Register for port C on MC68HC05C9
00137   000A    SPCR    equ   $A      Serial Peripheral Control Register
00138   000B    SPSR    equ   $B      Serial Peripheral Status Register
00139   000C    SPDAT   equ   $C      Serial Peripheral DATa register
00140   000D    SCBRR   equ   $D      Serial Communications Baud Rate Generator
00141   000E    SCCR1   equ   $E      Serial Communications Control Register 1
00142   000F    SCCR2   equ   $F      Serial Communications Control Register 2
00143   0010    SCSR    equ   $10     Serial Communications Status Register
00144   0011    SCDAT   equ   $11     Serial Communications DATa register
00145   0012    TCR     equ   $12     Timer Control Register
00146   0013    TSR     equ   $13     Timer Status Register
00147   0014    ICHR    equ   $14     Input Capture High Register
00148   0015    ICLR    equ   $15     Input Capture Low Register
00149   0016    OCHR    equ   $16     Output Compare High Register
00150   0017    OCLR    equ   $17     Output Compare Low Register
00151   0018    CHR     equ   $18     Counter High Register
00152   0019    CLR     equ   $19     Counter Low Register
00153   001A    ACHR    equ   $1A     Alternate Counter High Register
00154   001B    ACLR    equ   $1B     Alternate Counter Low Register
00155   001C    PR      equ   $1C     Program Register
00156   001D    COPR    equ   $1D     COP reset register
00157   001E    COPC    equ   $1E     COP control register
00158   1FDF    OPT1    equ   $1FDF   option register
00159   3FDF    OPT2    equ   $3FDF   option register in 705C9 (Emulator)

00161   00A0    EEPPA   equ   $A0     include physical device address of EEPROM

00163           *       Bit Assignments
00164           *       Timer Control Register - TCR - See P4-9
00165   0007    ICIE    equ   7       Input Capture Interrupt Enable
00166   0006    OCIE    equ   6       Output Compare Interrupt Enable
00167   0005    TOIE    equ   5       Timer Overflow Interrupt Enable
00168   0001    IEDG    equ   1       Input EDGe
00169   0000    OLVL    equ   0       Output LeVeL

00171           *       Timer Status Register - TSR - See p4-10
00172   0007    ICF     equ   7       Input Capture Flag
00173   0006    OCF     equ   6       Output Compare Flag
00174   0005    TOF     equ   5       Timer Overflow Flag

00176           *       Serial Communications Control Register 1 - SCCR1 - See p5-5
00177   0007    R8      equ   7       Receive bit #8
00178   0006    T8      equ   6       Transmit bit #8
00179   0004    M       equ   4       Mode bit (0=>8 bits, 1=>9)
00180   0003    WAKE    equ   3       Wake-up enable - See 5-7

00182           *       Serial Communications Control Register 2 - SCCR2 - See p5-7
00183   0007    TIE     equ   7       Transmit Interrupt Enable
00184   0006    TCIE    equ   6       Transmission Complete Interrupt Enable
00185   0005    RIE     equ   5       Receive Interrupt Enable
00186   0004    ILIE    equ   4       Idle Line Interrupt Enable
00187   0003    TE      equ   3       Transmit Enable
00188   0002    RE      equ   2       Receive Enable
00189   0001    RWU     equ   1       Receiver Wake-Up
00190   0000    SBK     equ   0       Send BreaK

00192           *       Serial Communications Status Register - SCSR - See p5-9
00193   0007    TDRE    equ   7       Transmit ata Register Empty
00194   0006    TC      equ   6       Transmission Complete
00195   0005    RDRF    equ   5       Receive Data Register Full
00196   0004    IDLE    equ   4       Idle detect bit
00197   0003    OR      equ   3       Overrun error
00198   0002    NF      equ   2       Noise Flag
00199   0001    FE      equ   1       Framing Error

00201           *       Serial Communications Baud Rate Register - SCBRR - See p5-10
```

```
00202   0005        SCP1   equ   5       Serial Communications Prescaler
00203   0004        SCP0   equ   4       Divide by 1(00), 3(01),4(10),13(11)
00204   0002        SCR2   equ   2       Divide by 2**(SCR2,1,0)
00205   0001        SCR1   equ   1       i.e. SCR2=1,SCR0=0,SRC0=1 =>
00206   0000        SCR0   equ   0       divide by 2**5 = 32

00208               *      Serial Peripheral Control Register - SPCR - See p6-7
00209   0007        SPIE   equ   7       Serial Peripheral Interrupt Enable
00210   0006        SPE    equ   6       Serial Peripheral output Enable
00211   0004        MSTR   equ   4       MaSTeR bit
00212   0003        CPOL   equ   3       Clock Polarity
00213   0002        CPHA   equ   2       Clock Phase
00214   0001        SPR1   equ   1       Serial Peripheral Rate Select
00215   0000        SPR0   equ   0       Divide by 2(00),4(01),16(10),32(11)

00217               *      Serial Peripheral Status Register - SPSR - See p6-8
00218   0007        SPIF   equ   7       Serial PerIpheral Flag
00219   0006        WCOL   equ   6       Write COLlision
00220   0004        MODF   equ   4       MODe Fault flag

00222               *      Option Register
00223   0001        IRQ    equ   1       0=> -edge,  1=>-edge and level
00224   0003        SEC    equ   3       Security bit 1=> bootloader disabled
00225   0006        RAM1   equ   6       96 bytes of ram at $100
00226   0007        RAM0   equ   7       32 bytes of ram at $20

00228               *      Program Register
00229   0006        CPEN   equ   6       Charge Pump ENable
00230   0003        ERASE  equ   3       Clock Polarity
00231   0002        LATA   equ   2       LATch array A
00232   0001        LATB   equ   1       LATch Array B
00233   0000        EEPGM  equ   0       Electrically Erase/PRograM

00235               *      Port A
00236               *      Bidirectional Data Bus

00238               *      Port B
00239               *      All bits are outputs
00240   0000        NWRLB  equ   0       Not WRite Low Byte / A0
00241   0000        A0     equ   NWRLB
00242   0001        NWRHB  equ   1       Not WRite High Byte / A1
00243   0001        A1     equ   NWRHB
00244   0002        NRD    equ   2       Not Read
00245   0003        NWR    equ   3       Not Write
00246   0004        NLDAC  equ   4       Not LoaD AdC
00247   0005        RESET  equ   5       Reset AD1175 - ADC1 22 bit ADC
00248   0006        FREQ   equ   6       50/60Hz frequency select of ADC1
00249   0007        RTS    equ   7       RS232 output

00251               *      Port C
00252               *      All bits are outputs
00253   0000        PC0    equ   0       Not used
00254   0001        NSIS   equ   1       Not Switch Input Select     - U22
00255   0002        N1175  equ   2       Not select for AD1175       - ADC1
00256   0003        N1145H equ   3       Not AD1145 Heater select    - U14
00257   0004        N1145S equ   4       Not AD1145 Secondary select - U7
00258   0005        N1145P equ   5       Not AD1145 Primary select   - U6
00259   0006        N132   equ   6       Not MAX132 select           - U25
00260   0007        NEEP   equ   7       Not EEPROM select           - U4

00262               *      Port D
00263               *      Directions implicit by function on MC68HC705C8
00264               *      Emulator using 'C9 part needs data direction register set up
00265               *      see cfigtb
00266   0000        RX     equ   0       RS232 input
00267   0001        TX     equ   1       RS232 output
00268   0002        MISO   equ   2       Master In Slave Out from MAX132
00269   0003        MOSI   equ   3       Master Out Slave In to MAX132
```

```
00270        0004          SCLK    equ   4         Serial clock to MAX132
00271        0005          NSS     equ   5         Not Slave Select, tied to +5v
00272        0006          PD6     equ   6         Does not exist
00273        0007          CTS     equ   7         RS232 input
00274        0050          zrams   equ   $50       Start of RAM
00275        00FF          zrame   equ   $FF       End of RAM
00276        0030          zram0   equ   $30       Start of extra RAM block - OPTION reg bit #7
00277        0100          zram1   equ   $100      Start of extra RAM block - OPtion reg bit #6


00280        0200          zroms   equ   $0200     Start of (EEP)rom area
00281        10FF          zrome   equ   $10FF     End of (EEP)rom area
00282        1FF4          zvect   equ   $1FF4     Start of VECTor table


00285                      *       vector table
00286                      *       ------------
00287  1FF4                        org   zvect
00288  1FF4 0203           spi     fdb   spis      Serial Peripheral Interface
00289  1FF6 0205           sci     fdb   scis      Serial Communications Interface
00290  1FF8 0D62           tim     fdb   tims      TIMer Service routine
00291  1FFA 0207           irq     fdb   irqs      external Interrupt ReQuest
00292  1FFC 0209           swi     fdb   swis      Software Interrupt
00293  1FFE 0200           reset   fdb   start     Start-up routine

00295                      *       option register
00296                      *       ---------------
00297  1FDF                        org   OPT1      force no security bit by default in 'C8
00298  1FDF 02                     fcb   $02
00299  3FDF                        org   OPT2      force no security bit by default in 'C9
00300  3FDF 02                     fcb   $02

00302                      *       Ram locations
00303                      *       -------------

00305                      * Floating point numbers use a 3 byte mantissa and a 1 byte exponent.
00306                      * The mantissa is normalised so that the left most bit is 1 and this
00307                      * is then replace by the sign. The exponent is biased by $80, and zero
00308                      * is represented by an exponent of $00. If the exponent is $00 the
00309                      * mantissa bits are undefined.

00311  0030                        org   zram0     Floating point variables FP0..FP7 fit here
00312        0030          fpram   equ   *         start of variables
00313  0030                FP0     rmb   4         temporary result

00315        0034          FP1     equ   *
00316  0034                FP1EX   RMB   1         FP1 Exponent
00317  0035                FP1HB   RMB   1         FP1 Highest Significant Byte
00318  0036                FP1MB   RMB   1         FP1 Middle Significant Byte
00319  0037                FP1LB   RMB   1         FP1 Least Significant Byte

00321        0038          FP2     equ   *
00322  0038                FP2EX   RMB   1         FP2 Exponent
00323  0039                FP2HB   RMB   1         FP2 Highest Significant Byte
00324  003A                FP2MB   RMB   1         FP2 Middle Significant Byte
00325  003B                FP2LB   RMB   1         FP2 Least Significant Byte

00327  003C                FP3     RMB   4         additional floating point variables
00328  0040                FP4     RMB   4
00329  0044                FP5     RMB   4
00330  0048                FP6     RMB   4
00331  004C                FP7     RMB   4

00333  0050                        org   zrams     define use of RAM locations

00335  0050                FCNT0   RMB   1         counters accessed by user not fp package
00336  0051                FCNT1   RMB   1
00337  0052                FCNT2   RMB   1
```

```
00338  0053              FCNT3  RMB   1
00339  0054              FCNT4  RMB   1
00340  0055              FCNT5  RMB   1
00341  0056              FCNT6  RMB   1
00342  0057              FCNT7  RMB   1

00344  0058              DMAL   RMB   1        Decimal conversion including FPT5-FPT0
00345  0059              FPT5   RMB   1        Temp area for product or quotient
00346  005A              FPT4   RMB   1        needs to be twice as long as mantissa
00347  005B              FPT3   RMB   1
00348  005C              FPT2   RMB   1
00349  005D              FPT1   RMB   1
00350  005E              FPT0   RMB   1

00352  005F              FP1GB  RMB   1        FP1 Guard Byte
00353  0060              FPEXE  rmb   1        exponent extension
00354  0061              FPEXOF rmb   1        exponent offset used in square root function
00355  0062              FPCNT1 rmb   1        counter within package
00356  0063              FP1SGN rmb   1        sign of fp1
00357  0064              FP2SGN rmb   1        sign of fp2
00358  0065              FPRSGN rmb   1        sign of result

00360  0066              FPERR  RMB   1        error code
00361       0001         over   equ   1
00362       0002         divzer equ   2
00363       0003         fixerr equ   3

00365  0067              FLAGS  rmb   1        flags to control interpreter operation
00366       0000         flexit equ   0        set to exit interpreter
00367       0001         flnoex equ   1        set for no execution of low nibble


00370  0068              FCODE  RMB   1        function code for/from interpreter
00371  0069              SMODC  RMB   7        self modifiable code see 'cfigtb'
00372                    *      lda   $????
00373                    *      rts
00374                    *      jmp   $????


00377                    * All variables are modified by timer interrupt service routine except
*
00378       0018         ADCLEN equ   8*3
00379  0070              ADCR   rmb   ADCLEN   space for all ADC readings
00380  0088              ADCC   rmb   ADCLEN   space for copy of ADC readings
00381  00A0              ADCS   rmb   4        sum for moving average
00382  00A4              ADCBP  rmb   1        ADC Buffer Pointer
00383  00A5              ADCH   rmb   1
00384  00A6              ADCM   rmb   1
00385  00A7              ADCL   rmb   1
00386  00A8              SPIT   rmb   1        store of last byte sent out via SPI
00387  00A9              TOUT   rmb   1        decremented at time out rate until zero
00388  00AA              TOVF   rmb   1        decremented at overflow rate until zero
00389  00AB              STATE  rmb   1        state counter for ADC control
00390  00AC              CHAN   rmb   1        channel counter for ADC multiplexer
00391  00AD              SAMCNT rmb   1        SAMple CouNT. 8=> complete list of readings
00392                    * Next two declarations must be adjacent
00393  00AE              PADCR  rmb   3        Precision ADC readings (first byte is ms)
00394  00B1              PADCST rmb   1        Precision ADC status

00396  00B2              PADCC  rmb   4        Precision ADC copy of readings incl. STATUS
00397  00B6              PADCA  rmb   3        Precision ADC Average (first byte is ms )

00399  00B9              MODE   rmb   1        MODE flags
00400                    * allocation of bits in MODE byte
00401       0000         QUIET  equ   0        1=>ADC sampling starts with fsyn => quiet RS232
00402       0001         NEXT   equ   1        1=> wait for next conversion of Precision ADC
00403       0007         ACK    equ   7        ACKnowledge byte for all commands.

00405  00BA              TEMPX  rmb   1        temp locations
```

```
00406  00BB                TEMPA  rmb   1
00407  00BC                BP     rmb   1         Buffer Pointer for command buffer

00409  00BD                PARAM  rmb   1         parameter(s) for commands
00410  00BE                PARH   rmb   1         high byte of parameter
00411  00BF                PARL   rmb   1         low byte of parameter

00413  00C0                BLSTAT rmb   1         BLink STATe
00414  00C1                BLCNT  rmb   1         BLink interval counter
00415  00C2                BLIVL  rmb   1         BLink InterVaL n * 0.131s

00417  0100                       org   zram1     auxilliary ram
00418       0028           CMDLEN equ   40
00419  0100                CMDBUF rmb   CMDLEN

00421                      * buffer for moving average window
00422       0030           ADCBL  equ   16*3      number of bytes in buffer
00423       0004           ADCBD  equ   4         number of bits to shift
00424  0128                ADCB   rmb   ADCBL     Precision ADC buffer


00427                      *      start of main code
00428                      *      -----------------
00429  0200                       org   zroms
00430  0200 CC 020B        start  jmp   main

00432                      *      null routines
00433  0203 20 FE     0203 spis   bra   spis      Serial Peripheral Interface
00434  0205 20 FE     0205 scis   bra   scis      Serial Communications Interface
00435  0207 20 FE     0207 irqs   bra   irqs      external Interrupt ReQuest
00436  0209 20 FE     0209 swis   bra   swis      Software Interrupt

00438  020B 9C             main   rsp              reset stack pointer
00439  020C 9B                    sei              disable interrupts
00440  020D CD 0CFE               jsr   cfig       set up ports and all special locations
00441  0210 9A                    cli

00443                      *      make power on LED flash
00444  0211 9D                    nop              changed to sei during development/testing
00445  0212 1A 01                 bset  RESET,PB   reset AD1175 precision ADC
00446  0214 9D                    nop
00447  0215 9D                    nop
00448  0216 9D                    nop
00449  0217 1B 01                 bclr  RESET,PB
00450  0219 CD 0231        main1  jsr   rdl        read a line
00451  021C 1F B9                 bclr  ACK,MODE   anticipate not acknowledged
00452  021E CD 026A               jsr   cmd        CoMmand Decode and execute
00453  0221 A6 4E                 lda   #'N        anticipate not acknowledged
00454  0223                       brcl  ACK,MODE,main2 anticipate not acknowledged
00455  0227 A6 59                 lda   #'Y
00456  0229 CD 0FA7        main2  jsr   zputw      'Y' or 'N' then CR LF
00457  022C CD 0F8B               jsr   crlf       end of command
00458  022F 20 E8     0219       bra   main1

00460                      * read command line into buffer
00461  0231 1F 01          rdl    bclr  RTS,PB     allow characters to arrive
00462  0233 5F                    clrx             point to start of buffer
00463  0234 CD 0F9B        rdl1   jsr   zgetw      get character
00464  0237 A4 7F                 and   #$7F       strip msb
00465  0239 A1 08                 cmp   #BS        is it a back space ?
00466  023B 26 13     0250        bne   rdl2       no => contine as before
00467  023D 5D                    tstx             if there have been no characters then ignore
00468  023E 27 F4     0234        beq   rdl1
00469  0240 CD 0FA7               jsr   zputw      echo a backspace
00470  0243 A6 20                 lda   #SP        over-write character
00471  0245 CD 0FA7               jsr   zputw
00472  0248 A6 08                 lda   #BS
00473  024A CD 0FA7               jsr   zputw
```

```
00474  024D 5A                    decx            remove entry
00475  024E 20 E4     0234        bra   rdl1      and try again
00476  0250 CD 0FA7         rdl2  jsr   zputw     echo
00477  0253 A1 0D                 cmp   #CR       test for carriage return
00478  0255 26 06     025D        bne   rdl3
00479  0257 A6 0A                 lda   #LF
00480  0259 CD 0FA7               jsr   zputw     and send line feed
00481  025C 4F                    clra            put null code at end of buffer
00482  025D D7 0100         rdl3  sta   CMDBUF,x
00483  0260 27 07     0269        beq   rdlz
00484  0262 A3 27                 cpx   #CMDLEN-1
00485  0264 27 CE     0234        beq   rdl1
00486  0266 5C                    incx
00487  0267 20 CB     0234        bra   rdl1
00488  0269 81               rdlz  rts

00490                          * command decode
00491  026A 3F BC              cmd   clr   BP        clear buffer pointer
00492  026C CD 02BD                 jsr   ibnc      get next character from input buffer
00493  026F A1 41                   cmp   #'A
00494  0271 25 14     0287          blo   cmdz      less than 'A'
00495  0273 A1 5A                   cmp   #'Z
00496  0275 22 10     0287          bhi   cmdz      greater than 'Z'
00497  0277 A0 41                   sub   #'A
00498  0279 48                      lsla            form word offset
00499  027A 97                      tax
00500  027B D6 0288                 lda   cmdtb,x   get jump address
00501  027E B7 6E                   sta   SMODC+5   into self modifying code
00502  0280 D6 0289                 lda   cmdtb+1,x
00503  0283 B7 6F                   sta   SMODC+6
00504  0285 BC 6D                   jmp   SMODC+4   execute
00505  0287 81                cmdz  rts


00508                          * a list of command from A..Z
00509  0288 033D               cmdtb fdb   cmdA      A - synchronises ADC then outputs 8 numbers
00510  028A 035F                     fdb   cmdB      B - repeated cmdA
00511  028C 0367                     fdb   cmdC      C - repeatedly outputs PADC, averaged PADC
00512  028E 0388                     fdb   cmdD      D - repeatedly outputs averaged PADC
00513  0290 02BC                     fdb   nul       E
00514  0292 02BC                     fdb   nul       F
00515  0294 02BC                     fdb   nul       G
00516  0296 03DA                     fdb   cmdH      H - Prepare Heater DAC
00517  0298 03A7                     fdb   cmdI      I - Output a ramp to Heater DAC
00518  029A 02BC                     fdb   nul       J
00519  029C 02BC                     fdb   nul       K
00520  029E 0404                     fdb   cmdL      L - Load all DACs - pulses -LDAC line
00521  02A0 0410                     fdb   cmdM      M - set MODE byte
00522  02A2 02BC                     fdb   nul       N
00523  02A4 02BC                     fdb   nul       O
00524  02A6 041A                     fdb   cmdP      P - Prepare Primary DAC
00525  02A8 0461                     fdb   cmdQ      Q - Read precision ADC 20 times
00526  02AA 0471                     fdb   cmdR      R - Read precision ADC as 6+2 hex characters
00527  02AC 0495                     fdb   cmdS      S - Prepare Secondary DAC
00528  02AE 04C2                     fdb   cmdT      T - Enable/Disable software protection of EEPROM
00529  02B0 057B                     fdb   cmdU      U - Read DIL switches as hex pair
00530  02B2 05C3                     fdb   cmdV      V - Read bytes from EEPROM
00531  02B4 05EA                     fdb   cmdW      W - Write a byte to EEPROM
00532  02B6 0615                     fdb   cmdX      X - Read moving average of recision ADC in Hex
00533  02B8 063B                     fdb   cmdY      Y - Read all MAX132 channels in hex
00534  02BA 064C                     fdb   cmdZ      Z - Set some/all DACs

00536                          * nothing to do
00537  02BC 81                 nul   rts

00539                         * get next character from input buffer - x points to character just read
00540                          * characters are folded to upper case
00541                          * z=1 if end of buffer
```

```
00542  02BD BE BC        ibnc   ldx   BP       get buffer pointer
00543  02BF D6 0100             lda   CMDBUF,x
00544  02C2 98                  clc            anticipate end of buffer
00545  02C3 27 09    02CE      beq   ibncz
00546  02C5 3C BC               inc   BP
00547  02C7 A1 60               cmp   #$60     test for lower case
00548  02C9 25 02    02CD      blo   ibnc1
00549  02CB A0 20               sub   #$20     move to upper case
00550  02CD 99           ibnc1  sec            flag valid character by cy=1
00551  02CE 81           ibncz  rts

00553                    * get parameter
00554                    * treat characters as hexadecimal and pack into a byte
00555  02CF 3F BD        getpa  clr   PARAM
00556  02D1 AD EA    02BD       bsr   ibnc     get next character
00557  02D3 24 1B    02F0       bcc   getpaz   exit with cy=0 if end of buffer
00558  02D5 AD 1C    02F3       bsr   chtb     convert ascii hex to binary
00559  02D7 24 17    02F0       bcc   getpaz   not HEX
00560  02D9 B7 BD               sta   PARAM
00561  02DB AD E0    02BD       bsr   ibnc     get next character
00562  02DD 24 11    02F0       bcc   getpaz   exit with cy=0 if end of buffer
00563  02DF AD 12    02F3       bsr   chtb     convert ascii hex to binary
00564  02E1 24 0D    02F0       bcc   getpaz   not HEX
00565  02E3 38 BD               lsl   PARAM    move to ms nibble
00566  02E5 38 BD               lsl   PARAM
00567  02E7 38 BD               lsl   PARAM
00568  02E9 38 BD               lsl   PARAM
00569  02EB BB BD               add   PARAM
00570  02ED B7 BD               sta   PARAM
00571  02EF 99                  sec            flag as valid by cy=1
00572  02F0 B6 BD        getpaz lda   PARAM
00573  02F2 81                  rts

00575                    * convert ascii hex character into binary
00576                    * s=1 if ok otherwise 0
00577  02F3 A1 30        chtb   cmp   #'0
00578  02F5 25 15    030C       blo   chtby    below '0', and cannot be converted
00579  02F7 A1 39               cmp   #'9
00580  02F9 22 04    02FF       bhi   chtb1
00581  02FB A0 30               sub   #'0      now in binary
00582  02FD 20 0A    0309       bra   chtbx
00583  02FF A1 41        chtb1  cmp   #'A
00584  0301 25 09    030C       blo   chtby    below 'A', and cannot be converted
00585  0303 A1 46               cmp   #'F
00586  0305 22 05    030C       bhi   chtby    above 'F', and cannot be converted
00587  0307 A0 37               sub   #'A-10
00588  0309 99           chtbx  sec            indicated conversion ok
00589  030A 20 01    030D       bra   chtbz
00590  030C 98           chtby  clc            indicates no conversion
00591  030D 81           chtbz  rts

00593                    * convert high and low nibbles to ASCII and send
00594  030E B7 BB        casb   sta   TEMPA
00595  0310 BF BA               stx   TEMPX
00596  0312 AD 0D    0321       bsr   cash
00597  0314 B6 BB               lda   TEMPA
00598  0316 48                  asla
00599  0317 48                  asla
00600  0318 48                  asla
00601  0319 48                  asla
00602  031A AD 05    0321       bsr   cash
00603  031C B6 BB               lda   TEMPA
00604  031E BE BA               ldx   TEMPX
00605  0320 81                  rts

00607                    * convert and send high nibble
00608  0321 44           cash   lsra
00609  0322 44                  lsra
```

```
00610  0323 44                  lsra
00611  0324 44                  lsra
00612  0325 97                  tax
00613  0326 D6 032D             lda   cashtb,x
00614  0329 CD 0FA7             jsr   zputw
00615  032C 81                  rts
00616  032D              cashtb fcc   "0123456789ABCDEF"

00618  033D AE 03        cmdA   ldx   #cAtb/256 pointer high
00619  033F A6 47               lda   #cAtb%256 pointer low
00620  0341 CD 06BD             jsr   fintp    interpret
00621  0344 1E B9               bset  ACK,MODE acknowledge
00622  0346 81                  rts

00624  0347              cAtb   fc    fcnt,1,10
00625  0349              cAtb1  fc    fbr,fsyn
00626  034A                     fc    fadc,0
00627  034B                     fc    ffun,fout
00628  034C                     fc    fadc,1
00629  034D                     fc    ffun,fout
00630  034E                     fc    fadc,2
00631  034F                     fc    ffun,fout
00632  0350                     fc    fadc,3
00633  0351                     fc    ffun,fout
00634  0352                     fc    fadc,4
00635  0353                     fc    ffun,fout
00636  0354                     fc    fadc,5
00637  0355                     fc    ffun,fout
00638  0356                     fc    fadc,6
00639  0357                     fc    ffun,fout
00640  0358                     fc    fadc,7
00641  0359                     fc    ffun,fout
00642  035A                     fc    ffun,fcrlf
00643  035B                     fc    fcnt,8+1,cAtb1-*-1
00644  035D                     fc    ffun,fcrlf
00645  035E                     fc    fbr,fexit

00647                           * repeated cmdA
00648  035F AD DC    033D cmdB  bsr   cmdA
00649  0361 CD 0F93             jsr   zget
00650  0364 24 F9    035F       bcc   cmdB
00651  0366 81                  rts


00654                           * outputs current reading from Precision ADC and Averaged reading from
00655                           * precision ADC
00656  0367 AD 08    0371 cmdC  bsr   cmdC1
00657  0369 CD 0F93             jsr   zget
00658  036C 24 F9    0367       bcc   cmdC
00659  036E 1E B9               bset  ACK,MODE acknowledge
00660  0370 81                  rts

00662  0371 AE 03        cmdC1  ldx   #cC1tb/256 pointer high
00663  0373 A6 79               lda   #cC1tb%256 pointer low
00664  0375 CD 06BD             jsr   fintp    interpret
00665  0378 81                  rts

00667  0379              cC1tb  fc    fcnt,1,10
00668  037B              cC1tb1 fc    fcnt,2,4
00669  037D              cC1tb2 fc    fadc,8
00670  037E                     fc    ffun,fout
00671  037F                     fc    fadc,9
00672  0380                     fc    ffun,fout
00673  0381                     fc    fcnt,8+2,cC1tb2-*-1
00674  0383                     fc    ffun,fcrlf
00675  0384                     fc    fcnt,8+1,cC1tb1-*-1
00676  0386                     fc    ffun,fcrlf
00677  0387                     fc    fbr,fexit
```

```
00680  0388 AD 08    0392 cmdD   bsr   cmdD1
00681  038A CD 0F93              jsr   zget
00682  038D 24 F9    0388        bcc   cmdD
00683  038F 1E B9                bset  ACK,MODE acknowledge
00684  0391 81                   rts

00686  0392 AE 03          cmdD1  ldx   #cD1tb/256 pointer high
00687  0394 A6 9A                lda   #cD1tb%256 pointer low
00688  0396 CD 06BD              jsr   fintp    interpret
00689  0399 81                   rts

00691  039A               cD1tb  fc    fcnt,1,10
00692  039C               cD1tb1 fc    fcnt,2,8
00693  039E               cD1tb2 fc    fadc,9
00694  039F                      fc    ffun,fout
00695  03A0                      fc    fcnt,8+2,cD1tb2-*-1
00696  03A2                      fc    ffun,fcrlf
00697  03A3                      fc    fcnt,8+1,cD1tb1-*-1
00698  03A5                      fc    ffun,fcrlf
00699  03A6                      fc    fbr,fexit


00702                     * test Heater DAC by outputting a RAMP
00703  03A7 3F 5E         cmdI   clr   FPT0
00704  03A9 3F 5D                clr   FPT1
00705  03AB CD 02CF              jsr   getpa    get high byte
00706  03AE 24 29    03D9        bcc   cmdIz    ignore if absent
00707  03B0 B7 5B                sta   FPT3
00708  03B2 CD 02CF              jsr   getpa    get high byte
00709  03B5 24 22    03D9        bcc   cmdIz    ignore if absent
00710  03B7 B7 5C                sta   FPT2
00711  03B9 B6 5D         cmdI1  lda   FPT1     get and output high part
00712  03BB CD 03EA              jsr   cmdHh
00713  03BE B6 5E                lda   FPT0     get and output low part
00714  03C0 CD 03F7              jsr   cmdHl
00715  03C3 CD 0409              jsr   cmdLs    pulse -LDAC line
00716  03C6 B6 5E                lda   FPT0
00717  03C8 BB 5C                add   FPT2
00718  03CA B7 5E                sta   FPT0
00719  03CC B6 5D                lda   FPT1
00720  03CE B9 5B                adc   FPT3
00721  03D0 B7 5D                sta   FPT1
00722  03D2 CD 0F93       cmdI2  jsr   zget     continue until a character has arrived
00723  03D5 24 E2    03B9        bcc   cmdI1
00724  03D7 1E B9                bset  ACK,MODE acknowledge
00725  03D9 81            cmdIz  rts

00727                     * write to Heater DAC buffer
00728  03DA CD 042D       cmdH   jsr   getpw    get word parameter
00729  03DD 24 0A    03E9        bcc   cmdHz    part is missing
00730  03DF B6 BE                lda   PARH     output high part
00731  03E1 AD 07    03EA        bsr   cmdHh
00732  03E3 B6 BF                lda   PARL
00733  03E5 AD 10    03F7        bsr   cmdHl
00734  03E7 1E B9                bset  ACK,MODE acknowledge
00735  03E9 81            cmdHz  rts

00737  03EA 9B             cmdHh  sei           protect sequence
00738  03EB B7 00                 sta   PA       output high byte to heater DAC
00739  03ED 13 01                 bclr  NWRHB,PB select high byte
00740  03EF 10 01                 bset  NWRLB,PB but not low byte
00741  03F1 17 02                 bclr  N1145H,PC
00742  03F3 16 02                 bset  N1145H,PC
00743  03F5 9A                    cli           allow interrupts again
00744  03F6 81                    rts

00746  03F7 9B             cmdHl  sei           protect sequence
00747  03F8 B7 00                 sta   PA       output low byte to heater DAC
```

```
00748  03FA 11 01              bclr  NWRLB,PB select low byte
00749  03FC 12 01              bset  NWRHB,PB but not high byte
00750  03FE 17 02              bclr  N1145H,PC
00751  0400 16 02              bset  N1145H,PC
00752  0402 9A                 cli           allow interrupts again
00753  0403 81                 rts

00755                       * pulse LDAC line low and acknowledge
00756  0404 AD 03    0409 cmdL   bsr   cmdLs
00757  0406 1E B9              bset  ACK,MODE acknowledge
00758  0408 81                 rts

00760                       * pulse LDAC line low
00761  0409 9B           cmdLs  sei           protect sequence
00762  040A 19 01              bclr  NLDAC,PB load all DAC's from internal latches
00763  040C 18 01              bset  NLDAC,PB
00764  040E 9A                 cli           allow interrupts again
00765  040F 81                 rts

00767                       * set MODE to first parameter
00768  0410 CD 02CF       cmdM   jsr   getpa    get parameter
00769  0413 24 04    0419        bcc   cmdMz    ignore if absent
00770  0415 B7 B9              sta   MODE
00771  0417 1E B9              bset  ACK,MODE acknowledge
00772  0419 81           cmdMz  rts

00774                       * write to Primary DAC buffer 2's complement of input parameters
00775  041A CD 042D       cmdP   jsr   getpw    get word parameter
00776  041D 24 0D    042C        bcc   cmdPz    part is missing
00777  041F CD 043C              jsr   paneg
00778  0422 B6 BE                lda   PARH     output high part
00779  0424 AD 21    0447        bsr   cmdPh
00780  0426 B6 BF                lda   PARL
00781  0428 AD 2A    0454        bsr   cmdPl
00782  042A 1E B9              bset  ACK,MODE acknowledge
00783  042C 81           cmdPz  rts

00785                       * get two bytes as a word parameter into PARH:PARL
00786                       * return with carry clear if characters are absent or invalid
00787  042D CD 02CF       getpw  jsr   getpa    get high byte
00788  0430 24 09    043B        bcc   getpwz   ignore if absent
00789  0432 B7 BE                sta   PARH
00790  0434 CD 02CF              jsr   getpa    get high byte
00791  0437 24 02    043B        bcc   getpwz   ignore if absent
00792  0439 B7 BF                sta   PARL
00793  043B 81           getpwz rts

00795                       * parameter PARH:PARL negate
00796  043C 4F           paneg  clra
00797  043D B0 BF              sub   PARL
00798  043F B7 BF              sta   PARL
00799  0441 4F                 clra
00800  0442 B2 BE              sbc   PARH
00801  0444 B7 BE              sta   PARH
00802  0446 81                 rts

00804  0447 9B           cmdPh  sei           protect sequence
00805  0448 B7 00              sta   PA       output high byte to Primary DAC
00806  044A 13 01              bclr  NWRHB,PB select high byte
00807  044C 10 01              bset  NWRLB,PB but not low byte
00808  044E 1B 02              bclr  N1145P,PC
00809  0450 1A 02              bset  N1145P,PC
00810  0452 9A                 cli           allow interrupts again
00811  0453 81                 rts

00813  0454 9B           cmdPl  sei           protect sequence
00814  0455 B7 00              sta   PA       output low byte to Primary DAC
00815  0457 11 01              bclr  NWRLB,PB select low byte
```

```
00816  0459 12 01              bset  NWRHB,PB but not high byte
00817  045B 1B 02              bclr  N1145P,PC
00818  045D 1A 02              bset  N1145P,PC
00819  045F 9A                 cli           allow interrupts again
00820  0460 81                 rts

00822                    * Read precision ADC 20 times
00823  0461 3F 5E        cmdQ  clr   FPTO     counter
00824  0463 AD 0C    0471 cmdQ1 bsr   cmdR     do reading
00825  0465 CD 0F8B           jsr   crlf
00826  0468 3C 5E             inc   FPTO
00827  046A B6 5E             lda   FPTO
00828  046C A1 14             cmp   #20
00829  046E 26 F3    0463     bne   cmdQ1
00830  0470 81                rts

00832                    * Read precision ADC as 6+2 hex characters
00833  0471              cmdR  brcl  NEXT,MODE,cmdR2
00834  0475              cmdR1 brst  0,PADCST,cmdR1
00835  0479 9B           cmdR2 sei             protect transfer
00836  047A 5F                 clrx
00837  047B E6 AE        cmdR3 lda   PADCR,x   copy data and status
00838  047D E7 B2              sta   PADCC,x
00839  047F 5C                 incx
00840  0480 A3 04              cpx   #4
00841  0482 26 F7    047B      bne   cmdR3
00842  0484 10 B1              bset  0,PADCST set lsb to indicate data has been read
00843  0486 9A                 cli
00844  0487 5F                 clrx
00845  0488 E6 B2        cmdR4 lda   PADCC,x  output from high byte to low byte
00846  048A CD 030E            jsr   casb
00847  048D 5C                 incx
00848  048E A3 04              cpx   #4
00849  0490 26 F6    0488      bne   cmdR4
00850  0492 1E B9              bset  ACK,MODE acknowledge
00851  0494 81                 rts

00853                    * write to Secondary DAC buffer 2's complement of input parameters
00854  0495 CD 042D      cmdS  jsr   getpw    get word parameter
00855  0498 24 0D    04A7      bcc   cmdSz    part is missing
00856  049A CD 043C            jsr   paneg
00857  049D B6 BE              lda   PARH     output high part
00858  049F AD 07    04A8      bsr   cmdSh
00859  04A1 B6 BF              lda   PARL
00860  04A3 AD 10    04B5      bsr   cmdSl
00861  04A5 1E B9              bset  ACK,MODE acknowledge
00862  04A7 81           cmdSz rts

00864  04A8 9B           cmdSh sei             protect sequence
00865  04A9 B7 00              sta   PA       output high byte to Secondary DAC
00866  04AB 13 01              bclr  NWRHB,PB select high byte
00867  04AD 10 01              bset  NWRLB,PB but not low byte
00868  04AF 19 02              bclr  N1145S,PC
00869  04B1 18 02              bset  N1145S,PC
00870  04B3 9A                 cli           allow interrupts again
00871  04B4 81                 rts

00873  04B5 9B           cmdSl sei             protect sequence
00874  04B6 B7 00              sta   PA       output low byte to Secondary DAC
00875  04B8 11 01              bclr  NWRLB,PB select low byte
00876  04BA 12 01              bset  NWRHB,PB but not high byte
00877  04BC 19 02              bclr  N1145S,PC
00878  04BE 18 02              bset  N1145S,PC
00879  04C0 9A                 cli           allow interrupts again
00880  04C1 81                 rts

00882                    * if parameter='P' then protect
00883                    * if parameter='U' then unprotect
```

```
00884                       * any other parameter is invalid
00885  04C2 CD 02BD      cmdT   jsr   ibnc
00886  04C5 24 29   04F0        bcc   cmdTz     no parameter
00887  04C7 A1 50               cmp   #'P
00888  04C9 27 09   04D4        beq   cmdT1
00889  04CB A1 55               cmp   #'U       unprotect
00890  04CD 26 21   04F0        bne   cmdTz
00891  04CF CD 04F1             jsr   unp
00892  04D2 20 1A   04EE        bra   cmdT2
00893  04D4               cmdT1  dely  1
00894  04DC 9B                   sei             should be OK for approx 1ms but protect anyway
00895  04DD A6 7F                lda   #$7F      address to read then write
00896  04DF B7 BE                sta   PARH
00897  04E1 A6 FF                lda   #$FF
00898  04E3 B7 BF                sta   PARL
00899  04E5 CD 058F              jsr   eepr
00900  04E8 B7 BD                sta   PARAM     PARAM=contents of ($7FFF)
00901  04EA 9A                   cli
00902  04EB CD 0527              jsr   prot
00903  04EE 1E B9         cmdT2  bset  ACK,MODE
00904  04F0 81            cmdTz  rts

00906                       * issue obscure control sequence to remove protection from EEPROM
00907  04F1               unp    dely  1
00908  04F9 9B                   sei             should be OK for approx 1ms but protect anyway
00909  04FA 5F                   clrx            work through table
00910  04FB D6 0569       unp1   lda   cmdTb2,x
00911  04FE CD 0D57              jsr   spioi
00912  0501 5C                   incx
00913  0502 D6 0569              lda   cmdTb2,x
00914  0505 CD 0D57              jsr   spioi
00915  0508 5C                   incx
00916  0509 D6 0569              lda   cmdTb2,x get data
00917  050C B7 00                sta   PA        onto bus
00918  050E 14 01                bset  NRD,PB    set up control lines
00919  0510 17 01                bclr  NWR,PB
00920  0512 1F 02                bclr  NEEP,PC   and enable
00921  0514 1E 02                bset  NEEP,PC
00922  0516 16 01                bset  NWR,PB
00923  0518 5C                   incx
00924  0519 A1 20                cmp   #$20
00925  051B 26 DE   04FB        bne   unp1
00926  051D 9A                   cli
00927  051E                      dely  4
00928  0526 81                   rts

00930                       * issue obscure control sequence to create protection for EEPROM
00931  0527               prot   dely  1
00932  052F 9B                   sei             should be OK for approx 1ms but protect anyway
00933  0530 5F                   clrx            work through table
00934  0531 D6 0560       prot1  lda   cmdTb1,x
00935  0534 CD 0D57              jsr   spioi
00936  0537 5C                   incx
00937  0538 D6 0560              lda   cmdTb1,x
00938  053B CD 0D57              jsr   spioi
00939  053E 5C                   incx
00940  053F D6 0560              lda   cmdTb1,x get data
00941  0542 B7 00                sta   PA        onto bus
00942  0544 14 01                bset  NRD,PB    set up control lines
00943  0546 17 01                bclr  NWR,PB
00944  0548 1F 02                bclr  NEEP,PC   and enable
00945  054A 1E 02                bset  NEEP,PC
00946  054C 16 01                bset  NWR,PB
00947  054E 5C                   incx
00948  054F A1 A0                cmp   #$A0
00949  0551 26 DE   0531        bne   prot1
00950  0553 CD 05AA              jsr   eepw      write byte previously read
00951  0556 9A                   cli
```

```
00952  0557                      dely  4
00953  055F 81                   rts

00955  0560 55           cmdTbl fcb   $55,$55,$AA protect sequence
00956  0563 2A                  fcb   $2A,$AA,$55
00957  0566 55                  fcb   $55,$55,$A0

00959  0569 55           cmdTb2 fcb   $55,$55,$AA unprotect sequence
00960  056C 2A                  fcb   $2A,$AA,$55
00961  056F 55                  fcb   $55,$55,$80
00962  0572 55                  fcb   $55,$55,$AA
00963  0575 2A                  fcb   $2A,$AA,$55
00964  0578 55                  fcb   $55,$55,$20


00967                     * read state of DIL switches and output as hex pair
00968  057B 9B            cmdU  sei             protect transfer
00969  057C 3F 04                clr   DDRA      change bus to inputs
00970  057E 13 02                bclr  NSIS,PC   enable buffer U22
00971  0580 9D                   nop             a few us to settle
00972  0581 B6 00                lda   PA        get data
00973  0583 12 02                bset  NSIS,PC   release bus
00974  0585 33 04                com   DDRA      restore to outputs
00975  0587 9A                   cli
00976  0588 43                   coma            change polarity
00977  0589 CD 030E              jsr   casb
00978  058C 1E B9                bset  ACK,MODE acknowledge
00979  058E 81                   rts

00981                     * read EEPROM at address given in PARH:PARL
00982                     * should be called from interrupt-protected environment
00983  058F B6 BE         eepr  lda   PARH      output address
00984  0591 CD 0D57              jsr   spioi     via SPI
00985  0594 B6 BF                lda   PARL
00986  0596 CD 0D57              jsr   spioi
00987  0599 3F 04                clr   DDRA      change to inputs
00988  059B 15 01                bclr  NRD,PB    set up control lines
00989  059D 16 01                bset  NWR,PB
00990  059F 1F 02                bclr  NEEP,PC   and enable
00991  05A1 B6 00                lda   PA        get data
00992  05A3 1E 02                bset  NEEP,PC
00993  05A5 14 01                bset  NRD,PB
00994  05A7 33 04                com   DDRA      change back to outputs
00995  05A9 81                   rts


00998                     * write to EEPROM at address given in PARH:PARL
00999                     * withe data in PARAM
01000                     * should be called from interrupt-protected environment
01001  05AA B6 BE         eepw  lda   PARH      output address
01002  05AC CD 0D57              jsr   spioi     via SPI
01003  05AF B6 BF                lda   PARL
01004  05B1 CD 0D57              jsr   spioi
01005  05B4 B6 BD                lda   PARAM     get data
01006  05B6 B7 00                sta   PA        onto bus
01007  05B8 14 01                bset  NRD,PB    set up control lines
01008  05BA 17 01                bclr  NWR,PB
01009  05BC 1F 02                bclr  NEEP,PC   and enable
01010  05BE 1E 02                bset  NEEP,PC
01011  05C0 16 01                bset  NWR,PB
01012  05C2 81                   rts


01015                     * read a block of bytes from the EEPROM U4
01016                     * because address is controlled by SPI interface this routine is
01017                     * interlaced with interrupt timing
01018  05C3 CD 042D       cmdV  jsr   getpw     get word into PARH:PARL as start address
01019  05C6 24 21    05E9        bcc   cmdVz     part is missing
```

```
01020  05C8 CD 02CF             jsr   getpa     get byte into PARAM as counter
01021  05CB 24 1C    05E9       bcc   cmdVz     part is missing
01022  05CD              cmdV1  dely  1
01023  05D5 9B                   sei            should be OK for approx 1ms but protect anyway
01024  05D6 CD 058F             jsr   eepr      read EEPROM
01025  05D9 9A                   cli            allow interrupts again
01026  05DA CD 030E             jsr   casb      output in hex
01027  05DD 3C BF               inc   PARL      increment address
01028  05DF 26 02    05E3       bne   cmdV2
01029  05E1 3C BE               inc   PARH
01030  05E3 3A BD         cmdV2 dec   PARAM     reduce loop counter
01031  05E5 26 E6    05CD       bne   cmdV1
01032  05E7 1E B9               bset  ACK,MODE acknowledge
01033  05E9 81            cmdVz rts

01035                           * write a byte to the EEPROM U4
01036                           * because address is controlled by SPI interface this routine is
01037                           * interlaced with interrupt timing
01038  05EA CD 042D       cmdW  jsr   getpw     get word parameter into PARH:PARL
01039  05ED 24 25    0614       bcc   cmdWz     part is missing
01040  05EF CD 02CF             jsr   getpa     get byte parameter into PARAM
01041  05F2 24 20    0614       bcc   cmdWz     part is missing
01042  05F4                     dely  1
01043  05FC 9B                  sei            should be OK for approx 1ms but protect anyway
01044  05FD CD 05AA             jsr   eepw      write to EEPROM
01045  0600 9A                  cli            allow interrupts again
01046  0601                     dely  4
01047  0609 9B                  sei            should be OK for approx 1ms but protect anyway
01048  060A CD 058F             jsr   eepr      read from EEPROM
01049  060D 9A                  cli            allow interrupts again
01050  060E B1 BD               cmp   PARAM     see if byte was written OK
01051  0610 26 02    0614       bne   cmdWz     leave as negative acknowledge
01052  0612 1E B9               bset  ACK,MODE acknowledge
01053  0614 81            cmdWz rts

01055                           * read moving average of precision adc in hex
01056                           * this is in PADCA, PADCA+1 and PADCA+2 and may be modified
01057                           * during timer interrupt. Care is take to interlace readings.
01058  0615                cmdX  dely  1
01059  061D 9B                  sei            should be OK for approx 1ms but protect anyway
01060  061E B6 B8               lda   PADCA+2
01061  0620 B7 BD               sta   PARAM
01062  0622 B6 B7               lda   PADCA+1
01063  0624 B7 BF               sta   PARL
01064  0626 B6 B6               lda   PADCA
01065  0628 B7 BE               sta   PARH      copy to temporary area
01066  062A 9A                  cli
01067  062B CD 030E             jsr   casb      now output as 6 hex characters
01068  062E B6 BF               lda   PARL
01069  0630 CD 030E             jsr   casb
01070  0633 B6 BD               lda   PARAM
01071  0635 CD 030E             jsr   casb
01072  0638 1E B9               bset  ACK,MODE acknowledge
01073  063A 81                  rts


01076                           * copy current MAX132 U25 readings and output in hex
01077  063B CD 0ADF       cmdY  jsr   adccp     copy block with interrupt protect
01078  063E 5F                  clrx
01079  063F E6 88         cmdY1 lda   ADCC,x    get byte(s)
01080  0641 CD 030E             jsr   casb      and output in hex pairs
01081  0644 5C                  incx
01082  0645 A3 18               cpx   #ADCLEN
01083  0647 26 F6    063F       bne   cmdY1
01084  0649 1E B9               bset  ACK,MODE acknowledge
01085  064B 81                  rts

01087                           * update some/all DACS if buffer empties DAC value does not change
```

```
01088  064C CD 0495       cmdZ   jsr   cmdS     update Secondary DAC
01089  064F                      brcl  ACK,MODE,cmdZz
01090  0653 CD 041A              jsr   cmdP      update Primary DAC
01091  0656 CD 03DA              jsr   cmdH      update Heater DAC
01092  0659 CD 0404              jsr   cmdL      load all DACs simultaneously
01093  065C 81            cmdZz  rts

01095                     * interpreter code structure
01096                     * a single code-byte consists of two 4-bit codes each called 'fop'.
01097                     * 'fop' indicates one of 16 operations and may require an additional
01098                     * 4 bits of 'sub' code, as well as additional bytes for operand(s).
01099                     * Within a byte if 'fop' is in the high nibble then the ssub-code is
01100                     * taken from the right nibble; if 'fop' is in the right nibble and a
01101                     * sub-code is needed then zero is assumed.

01103                     * define 'fop's ; fopdef creates table entries and offsets
01104                     * by concatanating 'f'+'arg' and 'fp'+'arg. See macro definition
01105                     * basic floating point operations jump table
01106       065D          foptab equ   *
01107  065D                       fopdef nop
01108  065F                       fopdef sld1
01109  0661                       fopdef sst1
01110  0663                       fopdef sld2
01111  0665                       fopdef ld1
01112  0667                       fopdef st1
01113  0669                       fopdef ld2
01114  066B                       fopdef add
01115  066D                       fopdef sub
01116  066F                       fopdef mul
01117  0671                       fopdef div
01118  0673                       fopdef br
01119  0675                       fopdef fun
01120  0677                       fopdef cnt
01121  0679                       fopdef adc
01122  067B 0726                  fdb   fpnop    F

01124                     * branch group jump table
01125       067D          fbrtab equ   *
01126  067D                       fbrdef bra
01127  067F                       fbrdef beq
01128  0681                       fbrdef bne
01129  0683                       fbrdef bpl
01130  0685                       fbrdef bmi
01131  0687                       fbrdef call
01132  0689                       fbrdef ret
01133  068B                       fbrdef exit
01134  068D                       fbrdef wt
01135  068F                       fbrdef syn
01136  0691                       fbrdef bc
01137  0693                       fbrdef bs
01138  0695                       fbrdef dly
01139  0697 0726                  fdb   fpnop    D
01140  0699 0726                  fdb   fpnop    E
01141  069B 0726                  fdb   fpnop    F



01146                     * function group jump table
01147       069D          funtab equ   *
01148  069D                       fundef clr
01149  069F                       fundef abs
01150  06A1                       fundef neg
01151  06A3                       fundef sgn
01152  06A5                       fundef sqt
01153  06A7                       fundef sqr
01154  06A9                       fundef swp
01155  06AB                       fundef fix
```

```
01156  06AD                      fundef flt
01157  06AF                      fundef int
01158  06B1                      fundef frac
01159  06B3                      fundef out
01160  06B5                      fundef crlf
01161  06B7                      fundef spac
01162  06B9 0726                 fdb   fpnop    E
01163  06BB 0726                 fdb   fpnop    F

01165                      * start of interpreter - may also be entered at fintpl to continue
01166                      ********************************************************************

01168  06BD BF 6A         fintp  stx   SMODC+1  high part of source address
01169  06BF B7 6B                sta   SMODC+2  low part of source address
01170  06C1 CD 070F       fintpl jsr   gnb      get next byte to interpret
01171  06C4 B7 68                sta   FCODE    save code for later
01172  06C6 3F 67                clr   FLAGS
01173  06C8 44                   lsra
01174  06C9 44                   lsra
01175  06CA 44                   lsra
01176  06CB 44                   lsra
01177  06CC CD 06ED              jsr   fdec     execute using foptab
01178  06CF                      brst  flnoex,FLAGS,fintp2
01179  06D3 B6 68                lda   FCODE
01180  06D5 A4 0F                and   #$0F     get low nibble
01181  06D7 3F 68                clr   FCODE    default sub-code to zero
01182  06D9 CD 06ED              jsr   fdec     execute
01183  06DC               fintp2 brcl  flexit,FLAGS,fintpl
01184  06E0 81                   rts           return

01186                      * set up table offset to access branch group
01187  06E1 12 67         fpbr   bset  flnoex,FLAGS
01188  06E3 AE 20                ldx   #fbrtab-foptab form offset
01189  06E5 20 08    06EF        bra   fdecl

01191                      * set up table offset to access function group
01192  06E7 12 67         fpfun  bset  flnoex,FLAGS
01193  06E9 AE 40                ldx   #funtab-foptab form offset
01194  06EB 20 02    06EF        bra   fdecl

01196                      * main decoder using look-up tables
01197  06ED AE 00         fdec   ldx   #foptab-foptab
01198  06EF BF 5E         fdecl  stx   FPTO     store offset
01199  06F1 48                   lsla          form word offset
01200  06F2 BB 5E                add   FPTO     point to right table
01201  06F4 97                   tax
01202  06F5 D6 065D              lda   foptab,x get jump address
01203  06F8 B7 6E                sta   SMODC+5  into self modifying code
01204  06FA D6 065E              lda   foptab+1,x
01205  06FD B7 6F                sta   SMODC+6
01206  06FF B6 68                lda   FCODE    copy of low nibble code if required
01207  0701 A4 0F                and   #$0F     mask out high bits
01208  0703 BC 6D                jmp   SMODC+4  execute

01210                      * convert number in x into bit position in A
01211  0705 A4 07         convb  and   #$07
01212  0707 97                   tax
01213  0708 4F                   clra
01214  0709 99                   sec
01215  070A 49            convbl rola
01216  070B 5A                   decx
01217  070C 2A FC    070A        bpl   convbl
01218  070E 81                   rts

01220                      * get next byte from stream being interpreted and increment address
01221  070F BD 69         gnb    jsr   SMODC    a=code byte
01222  0711 3C 6B         sminc  inc   SMODC+2  increment address
01223  0713 26 02    0717        bne   smincz
```

```
01224  0715 3C 6A              inc   SMODC+1
01225  0717 81           smincz rts

01227                    * add accumulator to address with sign extension
01228  0718 4D           smadd  tsta          sign extend first
01229  0719 2A 02    071D      bpl   smadd1
01230  071B 3A 6A              dec   SMODC+1  reduce high byte of address
01231  071D BB 6B       smadd1 add   SMODC+2  add to low byte
01232  071F B7 6B              sta   SMODC+2
01233  0721 24 02    0725      bcc   smaddz
01234  0723 3C 6A              inc   SMODC+1  correct high byte
01235  0725 81           smaddz rts



01239                    * no operation
01240                    **************

01242  0726 81           fpnop  rts           return with no other operation


01245                    * load and store operations
01246                    ***************************

01248                    * load FP1 using code in low nibble of A
01249  0727 12 67        fpsld1 bset  flnoex,FLAGS indicate no execution of low-nibble code
01250  0729 CD 07E7             jsr   cstol    convert short to long code
01251  072C 20 03    0731       bra   fpld1a

01253                    * load FP1 from source determined by x
01254  072E CD 070F      fpld1  jsr   gnb      get next byte
01255  0731 CD 07E0      fpld1a jsr   compx    compute value of x
01256  0734 25 2A    0760       bcs   fp1lx    load from external device = EEPROM
01257  0736 27 02    073A       beq   fp1lra   load from ram
01258  0738 26 11    074B       bne   fp1lro   load from rom

01260                    * load FP1 from page zero using x as offset i.e. FP1=(RAM)
01261  073A E6 30        fp1lra lda   fpram,x
01262  073C B7 34               sta   FP1EX
01263  073E E6 31               lda   fpram+1,x
01264  0740 B7 35               sta   FP1HB
01265  0742 E6 32               lda   fpram+2,x
01266  0744 B7 36               sta   FP1MB
01267  0746 E6 33               lda   fpram+3,x
01268  0748 B7 37               sta   FP1LB
01269  074A 81                  rts

01271                    * load FP1 from constants in ROM using x as offset i.e. FP1=(ROM)
01272  074B D6 0CDE      fp1lro lda   fprom,x
01273  074E B7 34               sta   FP1EX
01274  0750 D6 0CDF             lda   fprom+1,x
01275  0753 B7 35               sta   FP1HB
01276  0755 D6 0CE0             lda   fprom+2,x
01277  0758 B7 36               sta   FP1MB
01278  075A D6 0CE1             lda   fprom+3,x
01279  075D B7 37               sta   FP1LB
01280  075F 81                  rts

01282                  * load FP1 from constants in EEPROM using a as offset/2 i.e. FP1=(EEPROM)
01283  0760 AD 1C    077E fp1lx  bsr   fplx     load external to CMDBUF
01284  0762 AE 34               ldx   #FP1
01285  0764 20 04    076A       bra   fp2lx1
01286                  * load FP1 from constants in EEPROM using a as offset/2 i.e. FP1=(EEPROM)
01287  0766 AD 16    077E fp2lx  bsr   fplx     load external to CMDBUF
01288  0768 AE 38               ldx   #FP2
01289  076A C6 0100      fp2lx1 lda   CMDBUF   now copy to FPx
01290  076D F7                  sta   ,x
01291  076E C6 0101             lda   CMDBUF+1
```

```
01292  0771 E7 01              sta   1,x
01293  0773 C6 0102            lda   CMDBUF+2
01294  0776 E7 02              sta   2,x
01295  0778 C6 0103            lda   CMDBUF+3
01296  077B E7 03              sta   3,x
01297  077D 81                  rts

01299                    * not yet implemented
01300                    * action is to copy 4 bytes of EEPROM to CMDBUF
01301  077E 48           fplx   asla          move msb of offset to carry
01302  077F 5F                  clrx
01303  0780 59                  rolx          and then into x
01304                    * now copy from assigned location plus offset
01305  0781 81                  rts

01307                    * store FP1 using code in low nibble of A
01308  0782 12 67        fpsst1 bset  flnoex,FLAGS indicate no execution of low-nibble code
01309  0784 CD 07E7             jsr   cstol    convert short to long code
01310  0787 20 03    078C      bra   fpst1a

01312                    * store FP1 to destination determined by x
01313  0789 CD 070F      fpst1  jsr   gnb      get next byte
01314  078C CD 07E0      fpst1a jsr   compx    compute value of x
01315  078F 25 04    0795      bcs   fpst1z   external not yet defined
01316  0791 27 03    0796      beq   fp1sra   store to ram
01317  0793 26 00    0795      bne   fpst1z   store to rom not defined
01318  0795 81           fpst1z rts           load FP1

01320                    * store FP1 to page zero using x as offset i.e. (RAM)=FP1
01321  0796 B6 34        fp1sra lda   FP1EX
01322  0798 E7 30               sta   fpram,x
01323  079A B6 35               lda   FP1HB
01324  079C E7 31               sta   fpram+1,x
01325  079E B6 36               lda   FP1MB
01326  07A0 E7 32               sta   fpram+2,x
01327  07A2 B6 37               lda   FP1LB
01328  07A4 E7 33               sta   fpram+3,x
01329  07A6 81                  rts

01331                    * load FP2 using code in low nibble of A
01332  07A7 12 67        fpsld2 bset  flnoex,FLAGS indicate no execution of low-nibble code
01333  07A9 CD 07E7             jsr   cstol    convert short to long code
01334  07AC 20 03    07B1      bra   fpld2a

01336                    * load FP2 from source determined by x
01337  07AE CD 070F      fpld2  jsr   gnb      get next byte
01338  07B1 CD 07E0      fpld2a jsr   compx    compute value of x
01339  07B4 25 B0    0766      bcs   fp2lx    load from external device = EEPROM
01340  07B6 27 02    07BA      beq   fp2lra   load from ram
01341  07B8 26 11    07CB      bne   fp2lro   load from rom

01343                    * load FP2 from page zero using x as offset i.e. FP1=(RAM)
01344  07BA E6 30        fp2lra lda   fpram,x
01345  07BC B7 38               sta   FP2EX
01346  07BE E6 31               lda   fpram+1,x
01347  07C0 B7 39               sta   FP2HB
01348  07C2 E6 32               lda   fpram+2,x
01349  07C4 B7 3A               sta   FP2MB
01350  07C6 E6 33               lda   fpram+3,x
01351  07C8 B7 3B               sta   FP2LB
01352  07CA 81                  rts

01354                    * load FP2 from constants in ROM using x as offset i.e. FP2=(ROM)
01355  07CB D6 0CDE      fp2lro lda   fprom,x
01356  07CE B7 38               sta   FP2EX
01357  07D0 D6 0CDF             lda   fprom+1,x
01358  07D3 B7 39               sta   FP2HB
01359  07D5 D6 0CE0             lda   fprom+2,x
```

```
01360  07D8 B7 3A              sta   FP2MB
01361  07DA D6 0CE1            lda   fprom+3,x
01362  07DD B7 3B              sta   FP2LB
01363  07DF 81                 rts

01365                  * support routines used by above load/store operations
01366                  * input 00xxxxxx = RAM        output x=xxxxxx00
01367                  *       01xxxxxx = ROM               a=yyyyyyy0
01368                  *       1yyyyyyy = External
01369  07E0 97         compx  tax           copy of byte code number
01370  07E1 58                aslx
01371  07E2 58                aslx          x=??????00
01372  07E3 48                asla
01373  07E4 A5 80             bit   #$80    CY=1 if external, CY=0 if internal
01374  07E6 81                rts           if CY=0 then Z=1 if RAM, Z=0 if rom

01376                  * converts 0000?xxx to 0?000xxx; ?=0=>RAM, ?=1=>ROM, xxx=FPx or FKx
01377                  * 0 to  7 => RAM FP0 to FP7
01378                  * 8 to 15 => ROM FK0 to FK7
01379  07E7 AB 08      cstol  add   #$08     find out if bit #3 is set
01380  07E9 29 04  07EF       bhcs  cstol1
01381  07EB A4 07             and   #$07     remove bit #3
01382  07ED 20 02  07F1       bra   cstolz
01383  07EF AB 30      cstol1 add   #$30     move to bit #6
01384  07F1 81         cstolz rts

01386                  * basic floating point operation fpsub, fpadd, fpmul and fpdiv
01387                  ****************************************************************

01389                  * floating point subtract
01390  07F2 CD 0964    fpsub  jsr   fp2tst   nothing to do if FP2=0
01391  07F5 27 08  07FF       beq   fpsubz
01392  07F7 B6 39             lda   FP2HB    toggle sign bit
01393  07F9 A8 80             eor   #$80
01394  07FB B7 39             sta   FP2HB
01395  07FD 20 06  0805       bra   fpadd0
01396  07FF 81         fpsubz rts

01398                  * floating point add
01399  0800 CD 0964    fpadd  jsr   fp2tst   nothing to do if FP2=0
01400  0803 27 6C  0871       beq   fpaddz
01401  0805 CD 0960    fpadd0 jsr   fp1tst   test FP1
01402  0808 27 13  081D       beq   fpadd2   result is FP2
01403  080A B6 34             lda   FP1EX    find difference between exponents
01404  080C B0 38             sub   FP2EX
01405  080E 97                tax           keep count
01406  080F 26 05  0816       bne   fpadd1
01407  0811 CD 08EC           jsr   fppre    prepare mantissas
01408  0814 20 30  0846       bra   fpadd7   continue immediately - no shifts necessary
01409  0816 24 1E  0836 fpadd1 bcc   fpadd5   branch if FP1EX > FP2EX
01410  0818 40                nega          find positive difference
01411  0819 A1 18             cmp   #24      test for worthwhile shift
01412  081B 25 07  0824       bcs   fpadd3   do shift
01413  081D AE 08      fpadd2 ldx   #FP2-fpram
01414  081F CD 073A           jsr   fp1lra   FP1=FP2
01415  0822 20 4D  0871       bra   fpaddz
01416  0824 B6 38      fpadd3 lda   FP2EX    set up exponent of result
01417  0826 B7 34             sta   FP1EX
01418  0828 CD 08EC           jsr   fppre    prepare
01419  082B 34 35      fpadd4 lsr   FP1HB
01420  082D 36 36             ror   FP1MB
01421  082F 36 37             ror   FP1LB
01422  0831 5C                incx
01423  0832 26 F7  082B       bne   fpadd4
01424  0834 20 10  0846       bra   fpadd7
01425  0836 A1 18      fpadd5 cmp   #24      test for worthwhile shift
01426  0838 24 37  0871       bcc   fpaddz   FP2 is too small therefore result is FP1
01427  083A CD 08EC           jsr   fppre    prepare
```

```
01428  083D 34 39        fpadd6 lsr   FP2HB    do shift
01429  083F 36 3A               ror   FP2MB
01430  0841 36 3B               ror   FP2LB
01431  0843 5A                  decx
01432  0844 26 F7    083D        bne   fpadd6
01433  0846              fpadd7 brst  7,FPRSGN,fpadd8
01434  084A CD 09A5             jsr   uadd     addition may cause carry
01435  084D CD 0925             jsr   norm
01436  0850 B6 64               lda   FP2SGN
01437  0852 B7 65               sta   FPRSGN   make sign of result = sign of original FP2
01438  0854 20 18    086E        bra   fpaddb   now restore expected sign
01439  0856 CD 0987      fpadd8 jsr   usub
01440                    * FP1  FP2   abs(FP1)-abs(FP2)  required sign    use
01441                    * +4   -2     4-2 => + => CY=0       +           sign (FP1)
01442                    * +4   -5     4-5 => - => CY=1       -           sign (FP2)
01443                    * -4   +2     4-2 => + => CY=0       -           sign (FP1)
01444                    * -4   +5     4-5 => - => CY=1       +           sign (FP2)
01445                    * if CY=0 restore sign(FP1) else restore sign(FP2)
01446  0859 24 09    0864        bcc   fpadd9   CY=0 if abs(FP1)>abs(FP2)
01447  085B CD 0872             jsr   ineg
01448  085E B6 64               lda   FP2SGN
01449  0860 B7 65               sta   FPRSGN   make sign of result = sign of original FP2
01450  0862 20 04    0868        bra   fpadda
01451  0864 B6 63      fpadd9 lda   FP1SGN
01452  0866 B7 65               sta   FPRSGN   make sign of result = sign of original FP1
01453  0868 3F 5F      fpadda clr   FP1GB    guard byte = 0
01454  086A 98                  clc
01455  086B CD 0925             jsr   norm
01456  086E CD 0905      fpaddb jsr   fpend    restore correct sign
01457  0871 81           fpaddz rts

01459                    * integer negate of FP1 mantissa
01460  0872 4F           ineg   clra           negate result
01461  0873 B0 37               sub   FP1LB
01462  0875 B7 37               sta   FP1LB
01463  0877 4F                  clra
01464  0878 B2 36               sbc   FP1MB
01465  087A B7 36               sta   FP1MB
01466  087C 4F                  clra
01467  087D B2 35               sbc   FP1HB
01468  087F B7 35               sta   FP1HB
01469  0881 81                  rts

01471                    * floating point multiply
01472  0882 CD 0960      fpmul  jsr   fp1tst   if FP1=0 then
01473  0885 27 26    08AD        beq   fpmulz   exit
01474  0887 CD 0964             jsr   fp2tst   if FP2=0
01475  088A 26 05    0891        bne   fpmul0   then
01476  088C CD 0B2F             jsr   fp1clr   FP1=0
01477  088F 20 1C    08AD        bra   fpmulz   exit
01478  0891 CD 08EC      fpmul0 jsr   fppre    prepare
01479  0894 CD 09B9             jsr   umul     multiply mantissas and return with cy=0
01480  0897 CD 0925             jsr   norm     normalise
01481  089A B6 34               lda   FP1EX    add exponents with byte extension
01482  089C BB 38               add   FP2EX
01483  089E 24 02    08A2        bcc   fpmul1
01484  08A0 3C 60               inc   FPEXE    correct extension byte
01485  08A2 A0 80      fpmul1 sub   #$80     remove one bias
01486  08A4 B7 34               sta   FP1EX
01487  08A6 24 02    08AA        bcc   fpmul2
01488  08A8 3A 60               dec   FPEXE    correct extension byte
01489  08AA CD 0905      fpmul2 jsr   fpend    see if exponent is too big/small
01490  08AD 81           fpmulz rts

01492                    * floating point divide
01493  08AE CD 0960      fpdiv  jsr   fp1tst   if FP1=0 then
01494  08B1 27 FA    08AD        beq   fpmulz   exit
01495  08B3 CD 0964             jsr   fp2tst   if FP2=0
```

```
01496  08B6 26 0B    08C3        bne   fpdiv0   then
01497  08B8 B6 35                lda   FP1HB    set up Result Sign = sign of FP1
01498  08BA A4 80                and   #$80
01499  08BC B7 65                sta   FPRSGN
01500  08BE CD 0976              jsr   fp1max   FP1=max
01501  08C1 20 EA    08AD        bra   fpmulz   exit
01502  08C3 CD 08EC       fpdiv0 jsr   fppre    prepare
01503  08C6 3F 5E                clr   FPT0
01504  08C8 3F 5D                clr   FPT1
01505  08CA 3F 5C                clr   FPT2
01506  08CC CD 0A6B              jsr   udiv     divide mantissas msb in CY
01507  08CF 39 5F                rol   FP1GB    save in Guard Byte
01508  08D1 B6 34                lda   FP1EX    subtract exponents with byte extension
01509  08D3 B0 38                sub   FP2EX
01510  08D5 24 02    08D9        bcc   fpdiv1
01511  08D7 3A 60                dec   FPEXE    correct extension byte
01512  08D9 AB 80         fpdiv1 add   #$80     restore bias
01513  08DB B7 34                sta   FP1EX
01514  08DD 24 02    08E1        bcc   fpdiv2
01515  08DF 3C 60                inc   FPEXE    correct extension byte
01516  08E1 36 5F         fpdiv2 ror   FP1GB    get msb
01517  08E3 3F 5F                clr   FP1GB
01518  08E5 CD 0925              jsr   norm
01519  08E8 CD 0905              jsr   fpend    see if exponent is too big/small
01520  08EB 81            fpdivz rts


01523                     * support routines used by fpsub, fpadd, fpmul and fpdiv
01524                     * prepare to do fp operation - compute sign of result and set msb's
01525  08EC 3F 60         fppre  clr   FPEXE    clear exponent extension
01526  08EE 3F 66                clr   FPERR    remove error
01527  08F0 B6 35                lda   FP1HB
01528  08F2 A4 80                and   #$80
01529  08F4 B7 63                sta   FP1SGN   sign of FP1  .
01530  08F6 B6 39                lda   FP2HB
01531  08F8 A4 80                and   #$80
01532  08FA B7 64                sta   FP2SGN   sign of FP2
01533  08FC B8 63                eor   FP1SGN
01534  08FE B7 65                sta   FPRSGN   sign FP1 exor sign FP2
01535  0900 1E 35                bset  7,FP1HB  set msb's
01536  0902 1E 39                bset  7,FP2HB
01537  0904 81                   rts

01539                     * end fp operation - test for errors, restore sign etc.
01540  0905 3D 60         fpend  tst   FPEXE    FPEXE  Condition
01541  0907 27 0E    0917        beq   fpend2   =0     no error
01542  0909 2B 07    0912        bmi   fpend1   <0     underflow
01543  090B 12 66                bset  over,FPERR >0     overflow
01544  090D CD 0976              jsr   fp1max   set to max value using FPRSGN
01545  0910 20 12    0924        bra   fpendz
01546  0912 CD 0B2F       fpend1 jsr   fp1clr
01547  0915 20 0D    0924        bra   fpendz
01548  0917 CD 0960       fpend2 jsr   fp1tst   alter sign if non-zero
01549  091A 27 08    0924        beq   fpendz
01550  091C B6 35                lda   FP1HB
01551  091E A4 7F                and   #$7F     remove normalised bit
01552  0920 BA 65                ora   FPRSGN   combine with required sign for result
01553  0922 B7 35                sta   FP1HB
01554  0924 81            fpendz rts

01556                     * normalise mantissa so that msb=1
01557  0925 24 0B    0932 norm   bcc   norm0
01558  0927 36 35                ror   FP1HB
01559  0929 36 36                ror   FP1MB
01560  092B 36 37                ror   FP1LB
01561  092D CD 0952              jsr   incex    increment exponent
01562  0930 20 1F    0951        bra   normz    exit
01563  0932 B6 35         norm0  lda   FP1HB    nothing to do if msb=1
```

```
01564  0934 2B 1B    0951        bmi   normz    so exit
01565  0936 BA 36                ora   FP1MB    test for zero
01566  0938 BA 37                ora   FP1LB
01567  093A 26 08    0944        bne   norm2
01568  093C B7 34                sta   FP1EX    force FP1=0
01569  093E 20 11    0951        bra   normz    and exit
01570  0940               norm1  brst  7,FP1HB,normz and exit
01571  0944 38 5F         norm2  asl   FP1GB
01572  0946 39 37                rol   FP1LB
01573  0948 39 36                rol   FP1MB
01574  094A 39 35                rol   FP1HB
01575  094C CD 0959              jsr   decex
01576  094F 20 EF    0940        bra   norm1
01577  0951 81            normz  rts

01579  0952 3C 34         incex  inc   FP1EX    increment exponent with byte extension
01580  0954 26 02    0958        bne   incexz
01581  0956 3C 60                inc   FPEXE    correct extension byte
01582  0958 81            incexz rts

01584  0959 3A 34         decex  dec   FP1EX    increment exponent with byte extension
01585  095B 26 02    095F        bne   decexz
01586  095D 3A 60                dec   FPEXE    correct extension byte
01587  095F 81            decexz rts

01589                     * test fp1 for zero
01590  0960 AE 34         fp1tst ldx   #FP1
01591  0962 20 02    0966        bra   fptst

01593                     * test fp2 for zero
01594  0964 AE 38         fp2tst ldx   #FP2

01596                     * test floating pint number pointed to by X for zero
01597  0966 F6            fptst  lda   ,x
01598  0967 EA 01                ora   1,x
01599  0969 EA 02                ora   2,x
01600  096B EA 03                ora   3,x
01601  096D 27 06    0975        beq   fptstz
01602  096F E6 01                lda   1,x      get sign bit
01603  0971 A4 80                and   #$80
01604  0973 AA 01                ora   #$01     force to be non-zero
01605  0975 81            fptstz rts

01607                     * maximise floating point accumulator 1, ie all but sign bit to ones
01608                     * sign bit from FPRSGN
01609  0976 A6 FF         fp1max lda   #$FF
01610  0978 B7 34                sta   FP1EX
01611  097A B7 35                sta   FP1HB
01612  097C B7 36                sta   FP1MB
01613  097E B7 37                sta   FP1LB
01614  0980                      brst  7,FPRSGN,fp1maz
01615  0984 1F 35                bclr  7,FP1HB
01616  0986 81            fp1maz rts

01618                     * unsigned operations
01619                     * Unsigned subtraction : Mantissa parts of FP1=FP1-FP2
01620  0987 98            usub   clc
01621  0988 3F 5F         usub0  clr   FP1GB
01622  098A 39 5F                rol   FP1GB    get lsb of Guard Byte=Carry
01623  098C B6 37                lda   FP1LB    byte by byte
01624  098E B0 3B                sub   FP2LB
01625  0990 B7 37                sta   FP1LB
01626  0992 B6 36                lda   FP1MB
01627  0994 B2 3A                sbc   FP2MB
01628  0996 B7 36                sta   FP1MB
01629  0998 B6 35                lda   FP1HB
01630  099A B2 39                sbc   FP2HB
01631  099C B7 35                sta   FP1HB
```

```
01632  099E B6 5F              lda   FP1GB
01633  09A0 A2 00              sbc   #0       if underflow then CY=1, and FP1GB=$FF
01634  09A2 B7 5F              sta   FP1GB    otherwise CY=0 and FP1GB=$00
01635  09A4 81                 rts

01637                    * Unsigned addition    : Mantissa parts of FP1=FP1+FP2
01638  09A5 98           uadd  clc
01639  09A6 B6 37        uadd0 lda   FP1LB    byte by byte
01640  09A8 B9 3B              adc   FP2LB
01641  09AA B7 37              sta   FP1LB
01642  09AC B6 36              lda   FP1MB
01643  09AE B9 3A              adc   FP2MB
01644  09B0 B7 36              sta   FP1MB
01645  09B2 B6 35              lda   FP1HB
01646  09B4 B9 39              adc   FP2HB
01647  09B6 B7 35              sta   FP1HB    if overflow then CY=1 otherwise CY=0
01648  09B8 81                 rts

01650                    * Unsigned multiply
01651                    * do first partial product
01652  09B9 B6 37        umul  lda   FP1LB
01653  09BB BE 3B              ldx   FP2LB
01654  09BD                    mul
01655  09BE B7 5E              sta   FPT0     store lowest part
01656  09C0 BF 5D              stx   FPT1
01657  09C2 3F 5C              clr   FPT2     clear higher parts
01658  09C4 3F 5B              clr   FPT3
01659  09C6 3F 5A              clr   FPT4
01660  09C8 3F 59              clr   FPT5
01661                    * now do other partial products
01662  09CA                    pmul  FP1LB,FP2MB,FPT1
01663  09DC                    pmul  FP1MB,FP2LB,FPT1
01664  09EE                    pmul  FP1LB,FP2HB,FPT2
01665  0A00                    pmul  FP1MB,FP2MB,FPT2
01666  0A12                    pmul  FP1HB,FP2LB,FPT2
01667  0A24                    pmul  FP1MB,FP2HB,FPT3
01668  0A36                    pmul  FP1HB,FP2MB,FPT3
01669  0A48                    pmul  FP1HB,FP2HB,FPT4
01670  0A5A B6 59              lda   FPT5     copy result to FP1
01671  0A5C B7 35              sta   FP1HB
01672  0A5E B6 5A              lda   FPT4
01673  0A60 B7 36              sta   FP1MB
01674  0A62 B6 5B              lda   FPT3
01675  0A64 B7 37              sta   FP1LB
01676  0A66 B6 5C              lda   FPT2     keep bottom bits in Guard Byte
01677  0A68 B7 5F              sta   FP1GB
01678  0A6A 81                 rts

01680                    * Unsigned divide
01681  0A6B AE 19        udiv  ldx   #25      set up counter
01682  0A6D 98                 clc
01683  0A6E CD 0988      udiv1 jsr   usub0    try subtract FP1=FP1-FP2; cy=1 => won't go
01684  0A71 24 06  0A79        bcc   udiv2
01685  0A73 CD 09A5            jsr   uadd     restore FP1=(FP1-FP2)+FP2=FP1
01686  0A76 98                 clc            indicate subtraction failed
01687  0A77 20 01  0A7A        bra   udiv3
01688  0A79 99           udiv2 sec            indicate subtraction was ok
01689  0A7A 39 5E        udiv3 rol   FPT0     rotate result bits
01690  0A7C 39 5D              rol   FPT1
01691  0A7E 39 5C              rol   FPT2     high bit may produce carry
01692  0A80 39 37              rol   FP1LB
01693  0A82 39 36              rol   FP1MB
01694  0A84 39 35              rol   FP1HB
01695  0A86 5A                 decx
01696  0A87 26 E5  0A6E        bne   udiv1
01697  0A89 36 37              ror   FP1LB    recover msb to CY
01698  0A8B B6 5E              lda   FPT0
01699  0A8D B7 37              sta   FP1LB
```

```
01700  0A8F B6 5D              lda   FPT1
01701  0A91 B7 36              sta   FP1MB
01702  0A93 B6 5C              lda   FPT2
01703  0A95 B7 35              sta   FP1HB
01704  0A97 81          udivz  rts           result in CY:FP1HB:FP1MB:FP1LB


01707                         * branch group decoded by low nibble of FCODE
01708                         ********************************************

01710                         * branch always
01711  0A98 CD 070F       fpbra  jsr   gnb     get byte
01712  0A9B CD 0718              jsr   smadd   and add it in
01713  0A9E 81                   rts

01715  0A9F CD 070F       fpbran jsr   gnb     no branch => throw away offset byte
01716  0AA2 81                   rts

01718                         * branch if FP1=0
01719  0AA3 CD 0960       fpbeq  jsr   fp1tst
01720  0AA6 27 F0    0A98        beq   fpbra
01721  0AA8 20 F5    0A9F        bra   fpbran

01723                         * branch if FP1<>0
01724  0AAA CD 0960       fpbne  jsr   fp1tst
01725  0AAD 26 E9    0A98        bne   fpbra
01726  0AAF 20 EE    0A9F        bra   fpbran

01728                         * branch if FP1 sign bit = 0
01729  0AB1 CD 0960       fpbpl  jsr   fp1tst
01730  0AB4 2A E2    0A98        bpl   fpbra
01731  0AB6 20 E7    0A9F        bra   fpbran

01733                         * branch if FP1 sign bit = 1
01734  0AB8 CD 0960       fpbmi  jsr   fp1tst
01735  0ABB 2B DB    0A98        bmi   fpbra
01736  0ABD 20 E0    0A9F        bra   fpbran

01738  0ABF 81            fpcall rts           call subroutine

01740  0AC0 81            fpret  rts           return from subroutine

01742                         * exit interpreter
01743  0AC1 10 67         fpexit bset  flexit,FLAGS
01744  0AC3 81                   rts

01746                         * if a character has been received then wait for another character
01747  0AC4 CD 0F93       fpwt   jsr   zget    try to get a character
01748  0AC7 24 03    0ACC        bcc   fpwtz
01749  0AC9 CD 0F9B              jsr   zgetw   now wait for a character
01750  0ACC 81            fpwtz  rts

01752                         * synchronise with CHANnel counter; wait for channel 3 to become 0
01753  0ACD               fpsyn  brcl  QUIET,MODE,fpsyn1
01754  0AD1 A6 FF                lda   #$FF
01755  0AD3 B7 AD                sta   SAMCNT
01756  0AD5 B6 AD         fpsyn1 lda   SAMCNT  wait for 8 entries
01757  0AD7 A1 08                cmp   #8
01758  0AD9 26 FA    0AD5        bne   fpsyn1
01759  0ADB CD 0ADF              jsr   adccp   copy ADC block
01760  0ADE 81                   rts

01762  0ADF               adccp  dely  1
01763  0AE7 9B                   sei          protect environment for copy
01764  0AE8 5F                   clrx
01765  0AE9 E6 70         adccp1 lda   ADCR,x  copy block from ADCReadings to
01766  0AEB E7 88                sta   ADCC,x  ADCCopy of readings
01767  0AED 5C                   incx
```

```
01768  0AEE A3 18              cpx   #ADCLEN
01769  0AF0 26 F7    0AE9      bne   adccp1
01770  0AF2 3F AD              clr   SAMCNT    flag that all readings have been copied
01771  0AF4 9A                 cli           enable again
01772  0AF5 81                 rts

01774                      * usage  'fc fbr,fpbc,mask,port,offset'
01775                      * branch if bits selected by mask are all zero
01776  0AF6 CD 070F       fpbc   jsr   gnb      get mask
01777  0AF9 B7 BB             sta   TEMPA
01778  0AFB CD 070F           jsr   gnb
01779  0AFE 97                tax           form an index
01780  0AFF F6                lda   ,x      get data
01781  0B00 B4 BB         fpbc1  and   TEMPA   apply mask
01782  0B02 26 08    0B0C     bne   fpbc2
01783  0B04 CD 070F           jsr   gnb      add offset
01784  0B07 CD 0718           jsr   smadd    to address
01785  0B0A 20 03    0B0F     bra   fpbcz
01786  0B0C CD 070F       fpbc2  jsr   gnb      throw offset away
01787  0B0F 81            fpbcz  rts


01790                      * usage   'fc fbr,fpbs,mask,port,offset'
01791                      * branch if bits selected by mask are all one
01792  0B10 CD 070F       fpbs   jsr   gnb      get mask
01793  0B13 B7 BB             sta   TEMPA
01794  0B15 CD 070F           jsr   gnb
01795  0B18 97                tax           form an index
01796  0B19 F6                lda   ,x      get data
01797  0B1A 43                coma
01798  0B1B 20 E3    0B00     bra   fpbc1

01800                      * delay for multiple of 0.1 seconds, 25 seconds max
01801  0B1D CD 070F       fpdly  jsr   gnb      delay for n * 0.1 sec
01802  0B20 B7 BB             sta   TEMPA
01803  0B22               fpdly1 dely  40
01804  0B2A 3A BB             dec   TEMPA
01805  0B2C 26 F4    0B22     bne   fpdly1
01806  0B2E 81                rts


01809                      * function group decoded by low nibble of FCODE
01810                      ************************************************

01812                      * clear floating point accumulator 1, ie set to zero
01813       0B2F           fpclr  equ   *
01814  0B2F 3F 34         fp1clr clr   FP1EX    FP1=0
01815  0B31 3F 35             clr   FP1HB
01816  0B33 3F 36             clr   FP1MB
01817  0B35 3F 37             clr   FP1LB
01818  0B37 81                rts

01820                      * absolute value
01821  0B38 1F 35         fpabs  bclr  7,FP1HB
01822  0B3A 81                rts

01824                      * negate value
01825  0B3B CD 0960       fpneg  jsr   fp1tst   if FP1<>0 then toggle sign bit
01826  0B3E 27 06    0B46     beq   fpnegz
01827  0B40 B6 35             lda   FP1HB
01828  0B42 A8 80             eor   #$80
01829  0B44 B7 35             sta   FP1HB
01830  0B46 81            fpnegz rts

01832                      * FP1=-1,0,+1 matching sign of FP1
01833  0B47 CD 0960       fpsgn  jsr   fp1tst
01834  0B4A 27 10    0B5C     beq   fpsgnz   nothing to do if zero
01835  0B4C 97                tax           save sign
```

```
01836  0B4D A6 81              lda   #$81      set up +1
01837  0B4F B7 34              sta   FP1EX
01838  0B51 3F 35              clr   FP1HB
01839  0B53 3F 36              clr   FP1MB
01840  0B55 3F 37              clr   FP1LB
01841  0B57 5D                 tstx
01842  0B58 2A 02    0B5C      bpl   fpsgnz    leave as plus 1
01843  0B5A 1E 35              bset  7,FP1HB   make negative
01844  0B5C 81          fpsgnz rts

01846                          * makes use of FP0,FP1,FP2 and FP3
01847  0B5D CD 0960      fpsqt  jsr   fpltst
01848  0B60 27 6D    0BCF      beq   fpsqtz    nothing to do if zero
01849  0B62 2A 05    0B69      bpl   fpsqt0    ok if positive
01850  0B64 CD 0B2F            jsr   fplclr    clear if negative
01851  0B67 20 66    0BCF      bra   fpsqtz    exit
01852  0B69 B6 34      fpsqt0 lda   FP1EX     reduce range
01853  0B6B A0 80              sub   #$80      remove bias
01854  0B6D 47                 asra            divide by 2 saving sign
01855  0B6E B7 61              sta   FPEXOF    exponent offset - 2's complement
01856  0B70 A6 40              lda   #$40
01857  0B72 49                 rola            $80 or $81
01858  0B73 B7 34              sta   FP1EX     new exponent
01859  0B75 AE 0C              ldx   #FP3-fpram store FP1 to FP3 - call it y
01860  0B77 CD 0796            jsr   fp1sra
01861  0B7A                    brst  0,FP1EX,fpsqt1
01862  0B7E AE 14              ldx   #fksqtb-fprom load fp2 with 0.5901621 - constant b
01863  0B80 CD 07CB            jsr   fp2lro
01864  0B83 CD 0882            jsr   fpmul     multiply FP1=FP1*FP2
01865  0B86 AE 10              ldx   #fksqta-fprom load fp2 with 0.4173076 - constant a
01866  0B88 CD 07CB            jsr   fp2lro
01867  0B8B CD 0800            jsr   fpadd     add FP1=FP1+FP2
01868  0B8E 20 10    0BA0      bra   fpsqt2
01869  0B90 AE 10      fpsqt1 ldx   #fksqta-fprom load fp2 with 0.4173076 - constant a
01870  0B92 CD 07CB            jsr   fp2lro
01871  0B95 CD 0882            jsr   fpmul     multiply FP1=FP1*FP2
01872  0B98 AE 14              ldx   #fksqtb-fprom load fp2 with 0.5901621 - constant b
01873  0B9A CD 07CB            jsr   fp2lro
01874  0B9D CD 0800            jsr   fpadd     add FP1=FP1+FP2
01875  0BA0 AE 00      fpsqt2 ldx   #FP0-fpram
01876  0BA2 CD 0796            jsr   fp1sra    store FP1 to FP0 - store estimate xi
01877  0BA5 A6 02              lda   #2
01878  0BA7 B7 62              sta   FPCNT1    set up counter
01879  0BA9 AE 0C      fpsqt3 ldx   #FP3-fpram load FP1 from FP3 - recover y
01880  0BAB CD 073A            jsr   fp1lra
01881  0BAE AE 00              ldx   #FP0-fpram load FP2 from FP0 - recover xi
01882  0BB0 CD 07BA            jsr   fp2lra
01883  0BB3 CD 08AE            jsr   fpdiv     divide FP1=FP1/FP2
01884  0BB6 AE 00              ldx   #FP0-fpram load FP2 from FP0 - recover xi
01885  0BB8 CD 07BA            jsr   fp2lra
01886  0BBB CD 0800            jsr   fpadd     add FP1=FP1+FP2
01887  0BBE 3A 34              dec   FP1EX     divide result by 2
01888  0BC0 AE 00              ldx   #FP0-fpram store FP1 to FP0  - new estimate
01889  0BC2 CD 0796            jsr   fp1sra
01890  0BC5 3A 62              dec   FPCNT1    loop counter
01891  0BC7 26 E0    0BA9      bne   fpsqt3
01892  0BC9 B6 34              lda   FP1EX     correct exponent
01893  0BCB BB 61              add   FPEXOF
01894  0BCD B7 34              sta   FP1EX
01895  0BCF 81          fpsqtz rts

01897                          * square FP1
01898  0BD0 AE 04      fpsqr  ldx   #FP1-fpram
01899  0BD2 CD 07BA            jsr   fp2lra    FP2=FP1
01900  0BD5 CC 0882            jmp   fpmul

01902                          * swap FP1 with FP2
01903  0BD8 AE 03      fpswp  ldx   #3         set up counter
```

```
01904  0BDA E6 34          fpswp1 lda   FP1EX,x  move
01905  0BDC B7 5E                 sta   FPT0     temporary place
01906  0BDE E6 38                 lda   FP2EX,x
01907  0BE0 E7 34                 sta   FP1EX,x  FP1=FP2
01908  0BE2 B6 5E                 lda   FPT0     recover
01909  0BE4 E7 34                 sta   FP1EX,x  FP2=original FP1
01910  0BE6 5A                    decx
01911  0BE7 2A F1    0BDA         bpl   fpswp1
01912  0BE9 81                    rts

01914                       * convert floating point number in FP1 into fixed format
01915                       * Exponent=0; sign+7+8+8 data bits in FP1HB:FP1MB:FP1LB
01916  0BEA CD 0960         fpfix  jsr   fp1tst
01917  0BED 27 24    0C13         beq   fpfixz   nothing to do
01918  0BEF 3F 39                 clr   FP2HB    clear sign of FP2 so that FPRSGN=FP1SGN
01919  0BF1 CD 08EC               jsr   fppre
01920  0BF4 B6 34                 lda   FP1EX    calculate number of shifts
01921  0BF6 A0 80                 sub   #$80
01922  0BF8 22 03    0BFD         bhi   fpfix1
01923  0BFA CC 0B2F               jmp   fp1clr   clear FP1 and return
01924  0BFD A0 18           fpfix1 sub   #24
01925  0BFF 25 06    0C07         blo   fpfix2
01926  0C01 CD 0C14               jsr   imax     set to integer max value
01927  0C04 CC 0905               jmp   fpend    restore sign and return
01928  0C07 34 35           fpfix2 lsr   FP1HB    move right
01929  0C09 36 36                 ror   FP1MB
01930  0C0B 36 37                 ror   FP1LB
01931  0C0D 4C                    inca
01932  0C0E 26 F7    0C07         bne   fpfix2
01933  0C10 CC 0905               jmp   fpend    restore sign and return
01934  0C13 81              fpfixz rts

01936                       * set FP1 to maximum signed integer - 007FFFFF
01937  0C14 A6 7F           imax   lda   #$7F
01938  0C16 B7 35                 sta   FP1HB
01939  0C18 A6 FF                 lda   #$FF
01940  0C1A B7 36                 sta   FP1MB
01941  0C1C B7 37                 sta   FP1LB
01942  0C1E 3F 34                 clr   FP1EX
01943  0C20 81                    rts

01945                       * float integer in FP1 in 3 byte signed format
01946  0C21 3F 39           fpflt  clr   FP2HB    clear sign of FP2 so that FPRSGN=FP1SGN
01947  0C23 CD 08EC               jsr   fppre
01948  0C26 1F 35                 bclr  7,FP1HB  restore integer format
01949  0C28 A6 98                 lda   #$80+24  set up exponent
01950  0C2A CC 0868               jmp   fpadda   normalise, correct sign and return

01952                       * find integer part of FP1. (FP2 corrupted)
01953  0C2D CD 08EC         fpint  jsr   fppre    prepare FP1, FP2 about to contain mask
01954  0C30 CD 0C4B               jsr   fgmsk    generate mask
01955  0C33 CD 0C65         fpint1 jsr   famsk    apply mask
01956  0C36 B6 63                 lda   FP1SGN   use original sign
01957  0C38 B7 65                 sta   FPRSGN
01958  0C3A CC 0868               jmp   fpadda   normalise and return

01960                       * find fractional part of FP1. (FP2 corrupted)
01961  0C3D CD 08EC         fpfrac jsr   fppre    prepare FP1, FP2 about to contain mask
01962  0C40 CD 0C4B               jsr   fgmsk    generate mask
01963  0C43 33 3B                 com   FP2LB    complement mask
01964  0C45 33 3A                 com   FP2MB
01965  0C47 33 39                 com   FP2HB
01966  0C49 20 E8    0C33         bra   fpint1   apply mask, normalise and return

01968                       * generate mask in FP2 mantissa corresponding to the position
01969                       * of the binary point; 1's before and 0's after
01970  0C4B 4F              fgmsk  clra
01971  0C4C B7 39                 sta   FP2HB    clear mask bits
```

```
01972  0C4E B7 3A               sta   FP2MB
01973  0C50 B7 3B               sta   FP2LB
01974  0C52 B6 34               lda   FP1EX
01975  0C54 A0 80               sub   #$80      test for too small
01976  0C56 23 0C    0C64       bls   fgmskz
01977  0C58 99          fgmsk1 sec             introduce 1's
01978  0C59 36 39               ror   FP2HB
01979  0C5B 36 3A               ror   FP2MB
01980  0C5D 36 3B               ror   FP2LB
01981  0C5F 25 03    0C64       bcs   fgmskz   full up
01982  0C61 4A                  deca
01983  0C62 26 F4    0C58       bne   fgmsk1
01984  0C64 81          fgmskz rts

01986                           * apply mask in FP2 to FP1
01987  0C65 B6 37       famsk  lda   FP1LB     apply mask byte by byte
01988  0C67 B4 3B              and   FP2LB
01989  0C69 B7 37              sta   FP1LB
01990  0C6B B6 36              lda   FP1MB
01991  0C6D B4 3A              and   FP2MB
01992  0C6F B7 36              sta   FP1MB
01993  0C71 B6 35              lda   FP1HB
01994  0C73 B4 39              and   FP2HB
01995  0C75 B7 35              sta   FP1HB
01996  0C77 81                  rts

01998                           * counter group with low nibble = xnnn; x=0=>load, x=1=>decrement and
01999                           * branch if non-zero, nnn=counter number
02000                           ***************************************************************************

02002                           * load/decrement 8-bit counter indexed by low nibble
02003  0C78 12 67       fpcnt  bset  flnoex,FLAGS indicate no execution of low-nibble code
02004  0C7A A1 08              cmp   #$08      test for >7
02005  0C7C A4 07              and   #$07      allow only counters 0 to 7
02006  0C7E 97                  tax            save index
02007  0C7F 25 0C    0C8D       bcs   fpcnt1   just load counter
02008  0C81 CD 070F             jsr   gnb      get displacement if needed
02009  0C84 6A 50               dec   FCNT0,x  decrement and branch if non zero
02010  0C86 27 0A    0C92       beq   fpcntz   nothing to do if zero
02011  0C88 CD 0718             jsr   smadd    add a to self modifying code with sign extension
02012  0C8B 20 05    0C92       bra   fpcntz
02013  0C8D CD 070F     fpcnt1 jsr   gnb
02014  0C90 E7 50               sta   FCNT0,x  load counter
02015  0C92 81          fpcntz rts

02017                          * convert ADC channel given in low nibble - pass to FP1 as 0 to 2.0(ish)
02018                           * Number format from CS5505 is 0000 to FFFF for -2.5v to +2.5v
02019                           * Subtract $80 from high byte to change to 2's complement
02020                          ***************************************************************************
02021  0C93 12 67       fpadc  bset  flnoex,FLAGS indicate no execution of low-nibble code
02022  0C95 A1 08              cmp   #8        test for MAX132 range 0 to 7
02023  0C97 25 19    0CB2       blo   fpadc1
02024  0C99 A4 07              and   #7
02025  0C9B 97                  tax            pick up start address of 3-byte number
02026  0C9C DE 0CD6             ldx   fpadct,x #8 -> PADCR, >= #9 ->PADCA
02027  0C9F 9B                  sei            protect transfer
02028  0CA0 E6 02               lda   2,x
02029  0CA2 B7 37               sta   FP1LB
02030  0CA4 E6 01               lda   1,x
02031  0CA6 B7 36               sta   FP1MB
02032  0CA8 F6                  lda   ,x
02033  0CA9 9A                  cli
02034  0CAA A0 40               sub   #$40      convert to 2's complement
02035  0CAC B7 35               sta   FP1HB
02036  0CAE A6 83               lda   #$83      prepare exponent
02037  0CB0 20 12    0CC4       bra   fpadc2
02038  0CB2 AE 03       fpadc1 ldx   #3        form offset
02039  0CB4                     mul
```

```
02040  0CB5 97                  tax           into ADCC table
02041  0CB6 E6 8A               lda   ADCC+2,x get low part
02042  0CB8 B7 37               sta   FP1LB
02043  0CBA E6 89               lda   ADCC+1,x get middle part
02044  0CBC B7 36               sta   FP1MB
02045  0CBE E6 88               lda   ADCC,x   get high part
02046  0CC0 B7 35               sta   FP1HB
02047  0CC2 A6 85               lda   #$85
02048  0CC4 B7 34        fpadc2 sta   FP1EX    anticipate bias
02049  0CC6 3F 60               clr   FPEXE    clear extension
02050  0CC8 3F 65               clr   FPRSGN   anticipate positive
02051  0CCA                     brcl  7,FP1HB,fpadc3 anticipate positive
02052  0CCE 1E 65               bset  7,FPRSGN indicate negative
02053  0CD0 CD 0872             jsr   ineg     integer negate
02054  0CD3 CC 0868      fpadc3 jmp   fpadda   normalise and return

02056                    * table to convert channel numbers above 7 to start address
02057  0CD6 AE           fpadct fcb   PADCR,PADCA,PADCA,PADCA,PADCA,PADCA,PADCA,PADCA

02059                    * start of a block of useful constants - indexed by 0..63.
02060                    * at most 64 constants. First 8 accessible by short load operations.
02061       0CDE         fprom  equ   *
02062  0CDE 00           fk0    fcb   0,0,0,0  zero
02063  0CE2 81           fk1e0  fcb   $81,$00,$00,$00 + 1
02064  0CE6 84           fk1e1  fcb   $84,$20,$00,$00 + 10
02065  0CEA 94           fk1e6  fcb   $94,$74,$24,$00 + 1,000,000
02066  0CEE 7F           fksqta fcb   $7F,$55,$A9,$57 + 0.4173076
02067  0CF2 80           fksqtb fcb   $80,$17,$14,$DD + 0.5901621
02068  0CF6 00                  fcb   0,0,0,0
02069  0CFA 00                  fcb   0,0,0,0

02071                    * General configuration and support routines
02072                    * Note references to SMODC must be present for interpreter to work
02073                    ********************************************************************

02075                    *     Configure registers and ports and memory locations
02076  0CFE 9B           cfig   sei           must be protected
02077  0CFF 5F                  clrx          offset into table
02078  0D00 BF A8        cfig1  stx   SPIT     keep x
02079  0D02 D6 0D25             lda   cfigtb+1,x get data
02080  0D05 DE 0D24             ldx   cfigtb,x get address
02081  0D08 A3 FF               cpx   #$FF
02082  0D0A 27 07     0D13      beq   cfig2    end of table
02083  0D0C F7                  sta   ,x       write to port
02084  0D0D BE A8               ldx   SPIT     recover x
02085  0D0F 5C                  incx           move to next entry
02086  0D10 5C                  incx
02087  0D11 20 ED     0D00      bra   cfig1    and continue
02088  0D13 A6 C2        cfig2  lda   #%11000010 extra ram at $20 and $100
02089  0D15 C7 1FDF             sta   OPT1     68HC705C8
02090  0D18 C7 3FDF             sta   OPT2     68HC705C9 (used in EVM05 emulator)
02091  0D1B A6 42               lda   #%01000010 no start,50Hz, awake,vin,BA=01 => status
02092  0D1D CD 0EF5             jsr   adccom   configure MAX132 ADC
02093  0D20 CD 0E62             jsr   adcbc    clear buffer for precision ADC readings
02094  0D23 81                  rts

02096                    *     configuration table
02097  0D24 00           cfigtb fcb   PA,$00   data bus as outputs
02098  0D26 01                  fcb   PB,%00011111 controls inactive
02099  0D28 02                  fcb   PC,%11111111 all selects on stand-by
02100  0D2A 04                  fcb   DDRA,$FF default state of 'bus' is output
02101  0D2C 05                  fcb   DDRB,$FF
02102  0D2E 06                  fcb   DDRC,$FF CONV,CAL,CS's, SCLas outputs
02103  0D30 07                  fcb   DDRD,%00011000 needed for SPI with EVM05 using MC68HC705C9
02104  0D32 0A                  fcb   SPCR,%01010001
02105  0D34 0D                  fcb   SCBRR,%00110000 9600 baud
02106  0D36 0E                  fcb   SCCR1,%00001000
02107  0D38 0F                  fcb   SCCR2,%00001100
```

```
02108  0D3A 12                    fcb   TCR,%01100000 OCIE,TOIE on, OLVL=1 => LED off
02109  0D3C 1C                    fcb   PR,%00000000 Program EEPROM
02110  0D3E A9                    fcb   TOUT,0   counters to zero
02111  0D40 AA                    fcb   TOVF,0
02112  0D42 AB                    fcb   STATE,40 force time-out
02113  0D44 AC                    fcb   CHAN,0
02114  0D46 AD                    fcb   SAMCNT,$F8 allow ($100-$F8+8) test/warm-up cycles
02115  0D48 B9                    fcb   MODE,0   allow sampling during RS232 reception
02116                       * following are needed for floating point routines
02117  0D4A 69                    fcb   SMODC,$C6 lda extended
02118  0D4C 6C                    fcb   SMODC+3,$81 rts
02119  0D4E 6D                    fcb   SMODC+4,$CC jmp extended
02120  0D50 C0                    fcb   BLSTAT,5 BLink STATe => 5 changes
02121  0D52 C1                    fcb   BLCNT,0  BLink interval counter is cleared
02122  0D54 C2                    fcb   BLIVL,4  BLink InterVaL 4 * 0.131s = 0.5
02123  0D56 FF                    fcb   $FF      Stopper code


02126                       *      spi (interface) output and input a byte

02128  0D57 B7 A8          spioi sta   SPIT     keet last byte sent out
02129  0D59 B7 0C                sta   SPDAT    output a byte to get a byte
02130  0D5B                spioi1 brcl  SPIF,SPSR,spioi1
02131  0D5F B6 0C                lda   SPDAT    get data
02132  0D61 81                   rts

02134                       * Timer interrupt service routine
02135       04E2           TINVAL equ   2500*4/8 (interval in us)*  (clock in MHz.) / 8
02136  0D62                tims   brcl  TOF,TSR,tims1 (interval in us)*  (clock in MHz.) / 8
02137  0D66 B6 19                lda   CLR      remove overflow flag
02138  0D68 3D C0                tst   BLSTAT   if BLSTAT<>0 then
02139  0D6A 27 20     0D8C       beq   tims1
02140  0D6C 3D C1                tst   BLCNT    if BLCNT<> then
02141  0D6E 27 04     0D74       beq   tims0
02142  0D70 3A C1                dec   BLCNT    dec(BLCNT)
02143  0D72 20 12     0D86       bra   tims0a
02144  0D74 B6 C2          tims0 lda   BLIVL    else begin
02145  0D76 B7 C1                sta   BLCNT    BLCNT:=BLIVL
02146  0D78 B6 12                lda   TCR      TCR:=TCR xor $01 for OLVL
02147  0D7A A8 01                eor   #(1!<OLVL)
02148  0D7C B7 12                sta   TCR
02149  0D7E B6 C0                lda   BLSTAT
02150  0D80 A1 FF                cmp   #$FF     if BLSTAT<>$FF then dec(BLSTAT)
02151  0D82 27 02     0D86       beq   tims0a
02152  0D84 3A C0                dec   BLSTAT   end
02153  0D86 3D AA          tims0a tst   TOVF
02154  0D88 27 02     0D8C       beq   tims1
02155  0D8A 3A AA                dec   TOVF
02156  0D8C                tims1  brcl  OCF,TSR,tims2 end
02157  0D90 B6 17                lda   OCLR     add offset for timing interval
02158  0D92 AB E2                add   #TINVAL%256 low byte
02159  0D94 97                   tax           OCLR must be written last !
02160  0D95 B6 16                lda   OCHR
02161  0D97 A9 04                adc   #TINVAL/256 high byte
02162  0D99 B7 16                sta   OCHR
02163  0D9B BF 17                stx   OCLR     re-enable by write to low byte
02164  0D9D CD 0DAA              jsr   padcs    service precision adc
02165  0DA0 CD 0E77              jsr   adcs     service adc
02166  0DA3 3D A9                tst   TOUT
02167  0DA5 27 02     0DA9       beq   tims2
02168  0DA7 3A A9                dec   TOUT
02169  0DA9 80             tims2 rti           may be extended to allow for ICF

02171                       * service precision adc
02172  0DAA CD 0DBF        padcs  jsr   getst    get status byte
02173  0DAD A5 81                 bit   #$81     leave if warming up or busy
02174  0DAF 26 0D     0DBE        bne   padcsz
02175  0DB1 B7 B1                 sta   PADCST   save in status byte
```

```
02176  0DB3 CD 0DD2             jsr   getr      get readings
02177  0DB6 A6 00               lda   #$00      DEFCON command
02178  0DB8 CD 0DEA             jsr   putcm     put command
02179  0DBB CD 0DFA             jsr   sampb     sample to buffer and moving sum
02180  0DBE 81           padcsz rts

02182                           * get status of AD1175K by reading register 0
02183  0DBF 11 01        getst  bclr  A0,PB     set to register 00
02184  0DC1 13 01               bclr  A1,PB
02185  0DC3 3F 04               clr   DDRA      change port A to inputs
02186  0DC5 15 02               bclr  N1175,PC select AD1175K
02187  0DC7 15 01               bclr  NRD,PB   enable reading
02188  0DC9 B6 00               lda   PA        get data
02189  0DCB 14 01               bset  NRD,PB   disable reading
02190  0DCD 14 02               bset  N1175,PC disable AD1175K
02191  0DCF 33 04               com   DDRA      restore to outputs
02192  0DD1 81                  rts

02194                           * get readings from precision adc
02195  0DD2 3F 04        getr   clr   DDRA      change port A to inputs
02196  0DD4 AE 02               ldx   #2        form index to PADCR table
02197  0DD6 3C 01        getr1  inc   PB        move from status byte to next byte(s)
02198  0DD8 15 02               bclr  N1175,PC select AD1175K
02199  0DDA 15 01               bclr  NRD,PB   enable reading
02200  0DDC B6 00               lda   PA        get data
02201  0DDE E7 AE               sta   PADCR,x   store in table
02202  0DE0 14 01               bset  NRD,PB   disable reading
02203  0DE2 14 02               bset  N1175,PC disable AD1175K
02204  0DE4 5A                  decx           move to higher byte
02205  0DE5 2A EF     0DD6      bpl   getr1
02206  0DE7 33 04               com   DDRA      restore to outputs
02207  0DE9 81                  rts

02209                           * put command to precision ADC
02210  0DEA B7 00        putcm  sta   PA
02211  0DEC 11 01               bclr  A0,PB     set to register 00
02212  0DEE 13 01               bclr  A1,PB
02213  0DF0 15 02               bclr  N1175,PC select AD1175K
02214  0DF2 17 01               bclr  NWR,PB   enable writing
02215  0DF4 9D                  nop
02216  0DF5 16 01               bset  NWR,PB   disable writing
02217  0DF7 14 02               bset  N1175,PC disable AD1175K
02218  0DF9 81                  rts

02220                           * sum:=sum-buf(i); buf(i):=new entry:=ADCH,M,L;sum:=sum+buf(i);
02221                           * wrapup/round i;
02222                           * Routine timed at 120us
02223  0DFA BE A4        sampb  ldx   ADCBP
02224  0DFC B6 A3               lda   ADCS+3    sum:=sum-buf(index)
02225  0DFE D0 012A             sub   ADCB+2,x
02226  0E01 B7 A3               sta   ADCS+3
02227  0E03 B6 A2               lda   ADCS+2
02228  0E05 D2 0129             sbc   ADCB+1,x
02229  0E08 B7 A2               sta   ADCS+2
02230  0E0A B6 A1               lda   ADCS+1
02231  0E0C D2 0128             sbc   ADCB,x
02232  0E0F B7 A1               sta   ADCS+1
02233  0E11 24 02     0E15      bcc   sampb1
02234  0E13 3A A0               dec   ADCS      account for borrow from highest byte
02235  0E15 B6 AE        sampb1 lda   PADCR     buf(index):=PADCR High to Low
02236  0E17 D7 0128             sta   ADCB,x
02237  0E1A B6 AF               lda   PADCR+1
02238  0E1C D7 0129             sta   ADCB+1,x
02239  0E1F B6 B0               lda   PADCR+2
02240  0E21 D7 012A             sta   ADCB+2,x
02241  0E24 B6 A3               lda   ADCS+3    sum:=sum+buf(index)
02242  0E26 DB 012A             add   ADCB+2,x
02243  0E29 B7 A3               sta   ADCS+3
```

```
02244  0E2B B6 A2              lda   ADCS+2
02245  0E2D D9 0129            adc   ADCB+1,x
02246  0E30 B7 A2              sta   ADCS+2
02247  0E32 B6 A1              lda   ADCS+1
02248  0E34 D9 0128            adc   ADCB,x
02249  0E37 B7 A1              sta   ADCS+1
02250  0E39 24 02    0E3D      bcc   sampb2
02251  0E3B 3C A0              inc   ADCS      account for carry to highest byte
02252  0E3D 9F          sampb2 txa           get start point
02253  0E3E AB 03              add   #3
02254  0E40 A1 30              cmp   #ADCBL
02255  0E42 26 01    0E45      bne   sampb3
02256  0E44 4F                 clra          wrap round index
02257  0E45 B7 A4       sampb3 sta   ADCBP
02258  0E47 B6 A3              lda   ADCS+3   copy to PADCA
02259  0E49 B7 B8              sta   PADCA+2
02260  0E4B B6 A2              lda   ADCS+2
02261  0E4D B7 B7              sta   PADCA+1
02262  0E4F B6 A1              lda   ADCS+1
02263  0E51 B7 B6              sta   PADCA
02264  0E53 B6 A0              lda   ADCS
02265  0E55 AE 04              ldx   #ADCBD   now divide by 4,8,16 etc
02266  0E57 44          sampb4 lsra
02267  0E58 36 B6              ror   PADCA
02268  0E5A 36 B7              ror   PADCA+1
02269  0E5C 36 B8              ror   PADCA+2
02270  0E5E 5A                 decx
02271  0E5F 26 F6    0E57      bne   sampb4
02272  0E61 81                 rts           PADCA now is sum/2^n (n=ADCBD)

02274                   * ADC Clear Buffer and pointers
02275  0E62 5F          adcbc  clrx          clear ADC buffer, pointer and sum
02276  0E63 4F                 clra
02277  0E64 B7 A4              sta   ADCBP
02278  0E66 B7 A0              sta   ADCS
02279  0E68 B7 A1              sta   ADCS+1
02280  0E6A B7 A2              sta   ADCS+2
02281  0E6C B7 A3              sta   ADCS+3
02282  0E6E D7 0128     adcbc1 sta   ADCB,x
02283  0E71 5C                 incx
02284  0E72 A3 30              cpx   #ADCBL
02285  0E74 26 F8    0E6E      bne   adcbc1
02286  0E76 81                 rts

02288                   * service routine for ADC called from within timer service routine
02289                   * MAX132 manual is confusing in its numbering of registers.
02290                  * The letters B and A have been substituted for RS0 and RS1 respectively
02291                   * Hence BA = 00 => Data bits B10..B3
02292                   *       BA = 01 => Status bits and B2..B0
02293                   *       BA = 10 => Data Bits B18..B11
02294                   *       BA = 11 => Invalid
02295                  * See MAX132 manual for detailed(ish !) explanation of registers and bits

02297  0E77 3C AB       adcs   inc   STATE
02298  0E79 B6 AB              lda   STATE
02299  0E7B A1 0A              cmp   #10
02300  0E7D 25 75    0EF4      blo   adcsz     nothing to do
02301  0E7F 26 11    0E92      bne   adcs1     greater than 10
02302  0E81 B6 AC              lda   CHAN      move to next channel
02303  0E83 4C                 inca
02304  0E84 A4 07              and   #7        wrap to 3 bits
02305  0E86 B7 AC              sta   CHAN
02306  0E88 48                 asla          move up 4 bits as expected by register 1
02307  0E89 48                 asla          of MAX132
02308  0E8A 48                 asla
02309  0E8B 48                 asla
02310  0E8C 4C                 inca          select register 1
02311  0E8D CD 0EF5            jsr   adccom    send new channel address
```

```
02312  0E90 20 62    0EF4        bra   adcsz    now end
02313  0E92 A1 28          adcs1  cmp   #40
02314  0E94 24 57    0EED        bhs   adcs3    greater than 100ms
02315  0E96 A6 42                lda   #%01000010 no start,50Hz,awake,vin,BA=01 = status
02316  0E98 CD 0EF5              jsr   adccom
02317  0E9B A5 40                bit   #%01000000 wait for EOC to become high
02318  0E9D 27 55    0EF4        beq   adcsz    nothing to do
02319  0E9F B7 A7                sta   ADCL     store status and lsb's
02320  0EA1 A6 44                lda   #%01000100 no start,50Hz,awake,vin,BA=10 => B18..B11
02321  0EA3 CD 0EF5              jsr   adccom
02322  0EA6 A6 40                lda   #%01000000 no start,50Hz,awake,vin,BA=00 => B10..B3
02323  0EA8 CD 0EF5              jsr   adccom
02324  0EAB B7 A5                sta   ADCH
02325  0EAD A6 C2                lda   #%11000010 start,50Hz,awake,vin,BA=01 => status
02326  0EAF CD 0EF5              jsr   adccom
02327  0EB2 B7 A6                sta   ADCM
02328  0EB4 B6 A7                lda   ADCL     extract polarity and data bits
02329  0EB6 48                   asla
02330  0EB7 48                   asla
02331  0EB8 48                   asla
02332  0EB9 48                   asla
02333  0EBA 48                   asla
02334  0EBB 36 A5                ror   ADCH     move polarity to msb
02335  0EBD 36 A6                ror   ADCM
02336  0EBF 46                   rora
02337  0EC0 AE 04                ldx   #4
02338  0EC2 37 A5          adcs2  asr   ADCH     now move to right justify
02339  0EC4 36 A6                ror   ADCM
02340  0EC6 46                   rora
02341  0EC7 5A                   decx
02342  0EC8 26 F8    0EC2        bne   adcs2
02343  0ECA B7 A7                sta   ADCL
02344  0ECC B6 AC                lda   CHAN
02345  0ECE 4A                   deca           restore previous channel
02346  0ECF A4 07                and   #7
02347  0ED1 AE 03                ldx   #3
02348  0ED3                      mul
02349  0ED4 97                   tax            form an index
02350  0ED5 B6 A5                lda   ADCH
02351  0ED7 E7 70                sta   ADCR,x   copy to destination
02352  0ED9 B6 A6                lda   ADCM
02353  0EDB E7 71                sta   ADCR+1,x
02354  0EDD B6 A7                lda   ADCL
02355  0EDF E7 72                sta   ADCR+2,x
02356  0EE1 3F AB                clr   STATE
02357  0EE3 B6 AD                lda   SAMCNT   increase sample count
02358  0EE5 A1 08                cmp   #8
02359  0EE7 27 0B    0EF4        beq   adcsz    limit at 8
02360  0EE9 3C AD                inc   SAMCNT
02361  0EEB 20 07    0EF4        bra   adcsz
02362  0EED A6 C2          adcs3  lda   #%11000010 start,50Hz,awake,vin,BA=01 => status
02363  0EEF CD 0EF5              jsr   adccom
02364  0EF2 3F AB                clr   STATE
02365  0EF4 81             adcsz  rts

02367                     * MAXIM MAX132 ADC communication via SPI
02368  0EF5 1D 02         adccom bclr  N132,PC
02369  0EF7 CD 0D57              jsr   spioi
02370  0EFA 1C 02                bset  N132,PC
02371  0EFC 81                   rts

02373                     * output integer part of FP1
02374  0EFD AE 0C         fpout  ldx   #fk1e6-fprom get constant of 1e6
02375  0EFF CD 07CB              jsr   fp2lro   load FP2 from rom
02376  0F02 CD 0882              jsr   fpmul    find product
02377  0F05 CD 0BEA              jsr   fpfix
02378  0F08 A6 20                lda   #SP      output a space
02379  0F0A CD 0FA7              jsr   zputw
```

```
02380  0F0D A6 2B              lda   #'+      anticipate plus sign
02381  0F0F                    brcl  7,FP1HB,fpout1
02382  0F13 1F 35              bclr  7,FP1HB  clear sign bit
02383  0F15 A6 2D              lda   #'-      minus sign
02384  0F17 CD 0FA7      fpout1 jsr   zputw    output sign
02385  0F1A CD 0F5F            jsr   btod     binary to decimal conversion
02386                     *     jsr   drnd           round up
02387  0F1D CD 0F48            jsr   dprnt    in fixed point format x.xxxxx
02388  0F20 81                 rts

02390                     * output a carriage return and line feed
02391  0F21 CC 0F8B      fpcrlf jmp   crlf

02393                     * output a space
02394  0F24 A6 20        fpspac lda   #$20     code for a space
02395  0F26 CC 0FA7            jmp   zputw    output and return

02397                     * decimal round up if least significant digit is >=5
02398  0F29 AE 05        drnd   ldx   #5       get least significant digit
02399  0F2B E6 59               lda   DMAL+1,x digit #6, ie 7th. digit
02400  0F2D A1 05               cmp   #$05     test for range
02401  0F2F 25 16   0F47        blo   drndz    nothing to do as digit is 0..4
02402  0F31 99                  sec           force increment
02403  0F32 E6 58        drnd1  lda   DMAL,x   BCD add
02404  0F34 A9 00               adc   #0
02405  0F36 A1 0A               cmp   #10      decimal adjust
02406  0F38 25 07   0F41        bcs   drnd2
02407  0F3A AB 06               add   #6
02408  0F3C A4 0F               and   #$F
02409  0F3E 99                  sec
02410  0F3F 20 01   0F42        bra   drnd3
02411  0F41 98           drnd2  clc
02412  0F42 E7 58        drnd3  sta   DMAL,x
02413  0F44 5A                  decx
02414  0F45 2A EB   0F32        bpl   drnd1
02415  0F47 81           drndz  rts

02417                     * print decimal string without leading zero suppression
02418                     * decimal point after first digit
02419  0F48 5F           dprnt  clrx
02420  0F49 E6 58        dprnt1 lda   DMAL,x   get number
02421  0F4B AB 30               add   #$30     convert to ascii
02422  0F4D CD 0FA7             jsr   zputw
02423  0F50 5C                  incx
02424  0F51 A3 01               cpx   #1       decimal point after first digit
02425  0F53 26 05   0F5A        bne   dprnt2
02426  0F55 A6 2E               lda   #'.      output a decimal point
02427  0F57 CD 0FA7             jsr   zputw
02428  0F5A A3 07        dprnt2 cpx   #7       loop ?
02429  0F5C 26 EB   0F49        bne   dprnt1
02430  0F5E 81                  rts

02432                     * binary to decimal conversion
02433  0F5F AE 06        btod   ldx   #6       clear result registers
02434  0F61 6F 58        btod1  clr   DMAL,x
02435  0F63 5A                  decx
02436  0F64 2A FB   0F61        bpl   btod1
02437  0F66 A6 18               lda   #24      24 loops
02438  0F68 B7 62               sta   FPCNT1
02439  0F6A 39 37        btod2  rol   FP1LB    shift binary pattern
02440  0F6C 39 36               rol   FP1MB
02441  0F6E 39 35               rol   FP1HB
02442  0F70 AE 06               ldx   #6       set up loop counter
02443  0F72 E6 58        btod3  lda   DMAL,x   decimal double
02444  0F74 49                  rola
02445  0F75 A1 0A               cmp   #10      decimal adjust
02446  0F77 25 07   0F80        bcs   btod4
02447  0F79 AB 06               add   #6
```

```
02448  0F7B A4 0F              and   #$F
02449  0F7D 99                 sec
02450  0F7E 20 01    0F81      bra   btod5
02451  0F80 98           btod4  clc
02452  0F81 E7 58        btod5  sta   DMAL,x
02453  0F83 5A                  decx
02454  0F84 2A EC    0F72      bpl   btod3
02455  0F86 3A 62               dec   FPCNT1
02456  0F88 26 E0    0F6A      bne   btod2
02457  0F8A 81           btod6  rts

02459                    * output CR and LF
02460  0F8B A6 0D        crlf   lda   #CR
02461  0F8D AD 18    0FA7       bsr   zputw
02462  0F8F A6 0A               lda   #LF
02463  0F91 20 14    0FA7       bra   zputw

02465                    * see if there is a character in the receive register and get it
02466  0F93              zget   brcl  RDRF,SCSR,zgetz
02467  0F97 B6 11               lda   SCDAT    get character
02468  0F99 99                  sec            flag a character
02469  0F9A 81           zgetz  rts

02471                    * wait until there is a received character
02472  0F9B AD F6    0F93 zgetw  bsr   zget     try to get a character
02473  0F9D 24 FC    0F9B       bcc   zgetw    wait until successful
02474  0F9F 81                  rts

02476                    * try to output a character
02477  0FA0              zput   brcl  TDRE,SCSR,zputz
02478  0FA4 B7 11               sta   SCDAT    send data with carry set
02479  0FA6 81           zputz  rts            carry set => data sent

02481                    * wait until character has been output to buffer
02482  0FA7 AD F7    0FA0 zputw  bsr   zput     try to send
02483  0FA9 24 FC    0FA7       bcc   zputw    wait until successful
02484  0FAB 81                  rts
02485  0FAC                     end
```

S Y M B O L T A B L E

.00010	0226	.00440	0474	.00450	0478	.00460	04D8	.00470	04F5	.00480	0522	.00490	052B	.00500	055B
.00510	05D1	.00520	05F8	.00530	0605	.00540	0619	.00550	0652	.00980	06D2	.00990	06DF	.01000	0849
.01010	0943	.01020	0983	.01041	09DC	.01061	09EE	.01081	0A00	.01101	0A12	.01121	0A24	.01141	0A36
.01161	0A48	.01181	0A5A	.01200	0AD0	.01210	0AE3	.01220	0B26	.01230	0B7D	.01250	0CCD	.01260	0D5E
.01270	0D65	.01280	0D8F	.01300	0F12	.01310	0F96	.01320	0FA3	A0	0000	A1	0001	ACHR	001A
ACK	0007	ACLR	001B	ADCB	0128	ADCBD	0004	ADCBL	0030	ADCBP	00A4	ADCC	0088	ADCH	00A5
ADCL	00A7	ADCLEN	0018	ADCM	00A6	ADCR	0070	ADCS	00A0	BLCNT	00C1	BLIVL	00C2	BLSTAT	00C0
BP	00BC	BS	0008	CHAN	00AC	CHR	0018	CLR	0019	CMDBUF	0100	CMDLEN	0028	COPC	001E
COPR	001D	CPEN	0006	CPHA	0002	CPOL	0003	CR	000D	CTS	0007	DDRA	0004	DDRB	0005
DDRC	0006	DDRD	0007	DMAL	0058	EEPGM	0000	EEPPA	00A0	ERASE	0003	FCNT0	0050	FCNT1	0051
FCNT2	0052	FCNT3	0053	FCNT4	0054	FCNT5	0055	FCNT6	0056	FCNT7	0057	FCODE	0068	FE	0001
FLAGS	0067	FP0	0030	FP1	0034	FP1EX	0034	FP1GB	005F	FP1HB	0035	FP1LB	0037	FP1MB	0036
FP1SGN	0063	FP2	0038	FP2EX	0038	FP2HB	0039	FP2LB	003B	FP2MB	003A	FP2SGN	0064	FP3	003C
FP4	0040	FP5	0044	FP6	0048	FP7	004C	FPCNT1	0062	FPERR	0066	FPEXE	0060	FPEXOF	0061
FPRSGN	0065	FPT0	005E	FPT1	005D	FPT2	005C	FPT3	005B	FPT4	005A	FPT5	0059	FREQ	0006
ICF	0007	ICHR	0014	ICIE	0007	ICLR	0015	IDLE	0004	IEDG	0001	ILIE	0004	IRQ	0001
LATA	0002	LATB	0001	LF	000A	M	0004	MISO	0002	MODE	00B9	MODF	0004	MOSI	0003
MSTR	0004	N1145H	0003	N1145P	0005	N1145S	0004	N1175	0002	N132	0006	NEEP	0007	NEXT	0001
NF	0002	NLDAC	0004	NRD	0002	NSIS	0001	NSS	0005	NWR	0003	NWRHB	0001	NWRLB	0000
OCF	0006	OCHR	0016	OCIE	0006	OCLR	0017	OLVL	0000	OPT1	1FDF	OPT2	3FDF	OR	0003
PA	0000	PADCA	00B6	PADCC	00B2	PADCR	00AE	PADCST	00B1	PARAM	00BD	PARH	00BE	PARL	00BF
PB	0001	PC	0002	PC0	0000	PD	0003	PD6	0006	PR	001C	QUIET	0000	R8	0007
RAM0	0007	RAM1	0006	RDRF	0005	RE	0002	RESET	0005	RIE	0005	RTS	0007	RWU	0001
RX	0000	SAMCNT	00AD	SBK	0000	SCBRR	000D	SCCR1	000E	SCCR2	000F	SCDAT	0011	SCLK	0004
SCP0	0004	SCP1	0005	SCR0	0000	SCR1	0001	SCR2	0002	SCSR	0010	SEC	0003	SMODC	0069
SP	0020	SPCR	000A	SPDAT	000C	SPE	0006	SPIE	0007	SPIF	0007	SPIT	00A8	SPR0	0000
SPR1	0001	SPSR	000B	STATE	00AB	T8	0006	TC	0006	TCIE	0006	TCR	0012	TDRE	0007
TE	0003	TEMPA	00BB	TEMPX	00BA	TIE	0007	TINVAL	04E2	TOF	0005	TOIE	0005	TOUT	00A9
TOVF	00AA	TSR	0013	TX	0001	WAKE	0003	WCOL	0006	adcbc	0E62	adcbc1	0E6E	adccom	0EF5
adccp	0ADF	adccp1	0AE9	adcs	0E77	adcs1	0E92	adcs2	0EC2	adcs3	0EED	adcsz	0EF4	brcl	macr
brst	macr	btod	0F5F	btod1	0F61	btod2	0F6A	btod3	0F72	btod4	0F80	btod5	0F81	btod6	0F8A
cAtb	0347	cAtb1	0349	cC1tb	0379	cC1tb1	037B	cC1tb2	037D	cD1tb	039A	cD1tb1	039C	cD1tb2	039E
casb	030E	cash	0321	cashtb	032D	cfig	0CFE	cfig1	0D00	cfig2	0D13	cfigtb	0D24	chtb	02F3
chtb1	02FF	chtbx	0309	chtby	030C	chtbz	030D	cmd	026A	cmdA	033D	cmdB	035F	cmdC	0367
cmdC1	0371	cmdD	0388	cmdD1	0392	cmdH	03DA	cmdHh	03EA	cmdHl	03F7	cmdHz	03E9	cmdI	03A7
cmdI1	03B9	cmdI2	03D2	cmdIz	03D9	cmdL	0404	cmdLs	0409	cmdM	0410	cmdMz	0419	cmdP	041A
cmdPh	0447	cmdPl	0454	cmdPz	042C	cmdQ	0461	cmdQ1	0463	cmdR	0471	cmdR1	0475	cmdR2	0479
cmdR3	047B	cmdR4	0488	cmdS	0495	cmdSh	04A8	cmdSl	04B5	cmdSz	04A7	cmdT	04C2	cmdT1	04D4
cmdT2	04EE	cmdTb1	0560	cmdTb2	0569	cmdTz	04F0	cmdU	057B	cmdV	05C3	cmdV1	05CD	cmdV2	05E3
cmdVz	05E9	cmdW	05EA	cmdWz	0614	cmdX	0615	cmdY	063B	cmdY1	063F	cmdZ	064C	cmdZz	065C
cmdtb	0288	cmdz	0287	compx	07E0	convb	0705	convb1	070A	crlf	0F8B	cstol	07E7	cstol1	07EF
cstolz	07F1	decex	0959	decexz	095F	dely	macr	divzer	0002	dprnt	0F48	dprnt1	0F49	dprnt2	0F5A
drnd	0F29	drnd1	0F32	drnd2	0F41	drnd3	0F42	drndz	0F47	eepr	058F	eepw	05AA	fabs	0001
fadc	000E	fadd	0007	famsk	0C65	fbc	000A	fbeq	0001	fbmi	0004	fbne	0002	fbpl	0003
fbr	000B	fbra	0000	fbrdef	macr	fbrtab	067D	fbs	000B	fc	macr	fcall	0005	fclr	0000
fcnt	000D	fcrlf	000C	fdec	06ED	fdec1	06EF	fdiv	000A	fdly	000C	fexit	0007	ffix	0007
fflt	0008	ffrac	000A	ffun	000C	fgmsk	0C4B	fgmsk1	0C58	fgmskz	0C64	fint	0009	fintp	06BD
fintp1	06C1	fintp2	06DC	fixerr	0003	fk0	0CDE	fk1e0	0CE2	fk1e1	0CE6	fk1e6	0CEA	fksqta	0CEE
fksqtb	0CF2	fld1	0004	fld2	0006	flexit	0000	flnoex	0001	fmul	0009	fneg	0002	fnop	0000
fopdef	macr	foptab	065D	fout	000B	fp1clr	0B2F	fp1lra	073A	fp1lro	074B	fp1lx	0760	fp1max	0976
fp1maz	0986	fp1sra	0796	fp1tst	0960	fp2lra	07BA	fp2lro	07CB	fp2lx	0766	fp2lx1	076A	fp2tst	0964
fp\0	undf	fpabs	0B38	fpadc	0C93	fpadc1	0CB2	fpadc2	0CC4	fpadc3	0CD3	fpadct	0CD6	fpadd	0800
fpadd0	0805	fpadd1	0816	fpadd2	081D	fpadd3	0824	fpadd4	082B	fpadd5	0836	fpadd6	083D	fpadd7	0846
fpadd8	0856	fpadd9	0864	fpadda	0868	fpaddb	086E	fpaddz	0871	fpbc	0AF6	fpbc1	0B00	fpbc2	0B0C
fpbcz	0B0F	fpbeq	0AA3	fpbmi	0AB8	fpbne	0AAA	fpbpl	0AB1	fpbr	06E1	fpbra	0A98	fpbran	0A9F
fpbs	0B10	fpcall	0ABF	fpclr	0B2F	fpcnt	0C78	fpcnt1	0C8D	fpcntz	0C92	fpcrlf	0F21	fpdiv	08AE
fpdiv0	08C3	fpdiv1	08D9	fpdiv2	08E1	fpdivz	08EB	fpdly	0B1D	fpdly1	0B22	fpend	0905	fpend1	0912
fpend2	0917	fpendz	0924	fpexit	0AC1	fpfix	0BEA	fpfix1	0BFD	fpfix2	0C07	fpfixz	0C13	fpflt	0C21
fpfrac	0C3D	fpfun	06E7	fpint	0C2D	fpint1	0C33	fpld1	072E	fpld1a	0731	fpld2	07AE	fpld2a	07B1
fplx	077E	fpmul	0882	fpmul0	0891	fpmul1	08A2	fpmul2	08AA	fpmulz	08AD	fpneg	0B3B	fpnegz	0B46
fpnop	0726	fpout	0EFD	fpout1	0F17	fppre	08EC	fpram	0030	fpret	0AC0	fprom	0CDE	fpsgn	0B47
fpsgnz	0B5C	fpsld1	0727	fpsld2	07A7	fpspac	0F24	fpsqr	0BD0	fpsqt	0B5D	fpsqt0	0B69	fpsqt1	0B90
fpsqt2	0BA0	fpsqt3	0BA9	fpsqtz	0BCF	fpsst1	0782	fpst1	0789	fpst1a	078C	fpst1z	0795	fpsub	07F2
fpsubz	07FF	fpswp	0BD8	fpswp1	0BDA	fpsyn	0ACD	fpsyn1	0AD5	fptst	0966	fptstz	0975	fpwt	0AC4

fpwtz	0ACC	fret	0006	fsgn	0003	fsld1	0001	fsld2	0003	fspac	000D	fsqr	0005	fsqt	0004
fsst1	0002	fst1	0005	fsub	0008	fswp	0006	fsyn	0009	fundef	macr	funtab	069D	fwt	0008
getpa	02CF	getpaz	02F0	getpw	042D	getpwz	043B	getr	0DD2	getr1	0DD6	getst	0DBF	gnb	070F
ibnc	02BD	ibnc1	02CD	ibncz	02CE	imax	0C14	incex	0952	incexz	0958	ineg	0872	irq	1FFA
irqs	0207	main	020B	main1	0219	main2	0229	mul	macr	norm	0925	norm0	0932	norm1	0940
norm2	0944	normz	0951	nul	02BC	over	0001	padcs	0DAA	padcsz	0DBE	paneg	043C	pmul	macr
prot	0527	prot1	0531	putcm	0DEA	rdl	0231	rdl1	0234	rdl2	0250	rdl3	025D	rdlz	0269
reset	1FFE	sampb	0DFA	sampb1	0E15	sampb2	0E3D	sampb3	0E45	sampb4	0E57	sci	1FF6	scis	0205
smadd	0718	smadd1	071D	smaddz	0725	sminc	0711	smincz	0717	spi	1FF4	spioi	0D57	spioi1	0D5B
spis	0203	start	0200	swi	1FFC	swis	0209	tim	1FF8	tims	0D62	tims0	0D74	tims0a	0D86
tims1	0D8C	tims2	0DA9	uadd	09A5	uadd0	09A6	udiv	0A6B	udiv1	0A6E	udiv2	0A79	udiv3	0A7A
udivz	0A97	umul	09B9	unp	04F1	unp1	04FB	usub	0987	usub0	0988	zget	0F93	zgetw	0F9B
zgetz	0F9A	zput	0FA0	zputw	0FA7	zputz	0FA6	zram0	0030	zram1	0100	zrame	00FF	zrams	0050
zrome	10FF	zroms	0200	zvect	1FF4										

```
1       program cc1v15fp;
2       { Program to control current drives with messages sent from host or keyboard }
3       { same as cc1v15f.pas but with additional comments and revised layout        }

4       uses
5        dos,crt,ccsup,unit_a0,unit_a1,unit_a2,unit_dd;

6       const
7         file_text   = 'cc1v15f.dt'; {file for ASCII  copy of calibration parameters}
8         file_binary = 'cc1v15f.db'; {file for binary copy of calibration parameters}

9       { Development History
10        1v1   Data types and variables for one drive on COM1
11        1v2   Data TYpes for 1..3 drives but using just 1
12        1v2a  minor but essential corrections to 1v2
13        1v3   includes routines to convert longint to hex
14        1v4   major changes to allow 2 drives with scanning
15        1v4a  loop_state now initialised
16        1v4b  wt initialised
17        1v5   auxiliary ADC channels read once every 2 seconds
18              timeout during get_reply from drive
19        1v6   Windows and screen layout established
20        1v6a  some variables displayed
21        1v7   Refined presentation
22        1v7a  Amplifier 1 shown to 4dp for testing
23        1v8   Warning Box with Blink
24              Time per loop controlled by ticks_loop_max
25        1v8a  Difference between set point and precision adc is also averaged
26        1v9   Host communications established
27        1v10  Keyboard Entry primitives
28        1v10b Main program revised into smaller procedures
29        1v10d Keyboard entry almost complete
30        1v10e Mean loop time and loop monitor displayed, Ctrl Left Arrow available
31        1v11  AIM  Automatic determination of gains and offset for precision drive.
32        1v11a      Moving average filters rationalised
33        1v11b AIM  calibrate via internal source
34        1v11c      revised position of initial set_source(host);
35        1v11d      1v11c has strange compile-time error
36        1v11e      may calibrate p_dac
37        1v11f      corrected incorrect scope of if in cal_p
38        1v11g      uses fp_p_dac in calibration
39        1v11h      tests levels with p_dac=%10000010...0 to %01111110..0
40        1v12  AIM  to create a file of calibration points.
41                   regression fits now volts per lsb
42        1v12a      longint declaration in fun(bits..) to give expected results
43        1v12b      regression data stored in stats.x[] and y[]
44                   test and binary files created to store stats
45        1v12c      Revised Keywords. Reads calibration file.
46                   Does open loop control of dacs using calibration information
47        1v12d      corrections to calculation of p_code in main control loop
48        1v13  AIM  trimming both manual and automatic. revised filenemes
49                   length of filter for p_adc is altered depending on activity
50                   for calibration and after keyboard entry
51                     length=precision_adc_filter_length otherwise it is shorter
52        1v13a      heater is controlled and power estimated. files name.1pt
53                   contains resistance of coil then heater in ohms.
54                   find_nearest_index applies to stats_type generally
55        1v13c      keeps a copy of the fp_last_setpoint to decide if look_up
56                   is needed;
57        1v14  AIM  revision of structure for parameters and data files
58        1v14a      Reports on time of last calibration
59        1v14b      reports current time
60        1v14c      white_text now included in set_source
61        1v15  AIM  new procedure for displaying options
62                   stats_type now includes temperature_in
63                   temperature during calibration stored in parameter block
64                   estimate of finish time for calibration
65        1v15b      drive 1 showed out of range value for fp_s_dac after calibration
66        1v15c      cal_finished placed in set_dacs, revised keyboard routines
67        1v15d AIM  major revision of keyboard entry
68        1v15e      beginnings of parameter form
69        1v15f      fully operational
70        1v15fp     revised format of source code }
```

```
71      const
72        max_filter_length       = 50;
73        loop_time_filter_length = 25;
74        stats_data_max          = 200;
75        max_drive               = 2;
76        number_of_parameters    = 2;
77        space_for_parameters    = 10;
78        r_ref = 70.0;  { Refererence resistance assumed same for coil and heater }
79        v_max = 5;     { Maximum voltage from dac is 5 volts }
80        differential_range = 2 ; {During keyboard entry limit difference in drives}
81        max_error          = 0.000005;
82        ticks_Y_max        = 18 * 5 ; { 18.2 ticks per second * number of seconds }
83        ticks_timeout_max  = 18 * 3 ; { 18.2 ticks per second * number of seconds }
84        ticks_loop_max     = 18;
85        ticks_clock_max    = 18;      { up_date visible clock once per second }
86        wait = 0;
87        data_block_null : data_block_type = (v:0.0; vs:'0.0         '; vv:false);

88      type {Main}
89        float          = real;
90        com_state_type = (idle, command_sent, ack_received, nack_received,
91                          valid_reply,time_out,command_received,reply_sent,
92                          valid_command);
93        com_type       = record
94                           number,address,IRQ          : word;
95                           last_command, last_reply    : string;
96                           state                       : com_state_type;
97                           error                       : integer;
98                           ticks_timeout               : longint;
99                           response                    : boolean;
100                        end;

101       file_type      = record
102                          inp,out : text;
103                        end;

104       loop_state_type = (loop_idle,
105                          sent_X, received_X, sent_Y, received_Y, sent_Z, received_Z);

106       filter_type    = record
107                          history  : array[1..max_filter_length] of float;
108                          count,index,length,guard : integer;
109                          mean     : float;
110                          valid    : boolean;
111                        end;

112       loop_time_type = record
113                          current_tick,last_tick : longint;
114                          filter : filter_type;
115                        end;


116       cal_state_type = (cal_idle,cal_start,cal_p,cal_s,cal_finished);

117       stats_type     = record
118                          n : integer;
119                          x,y,error : array[1..stats_data_max] of float;
120                          xb,yb,a,b,r,sx,sx2,sy,sy2,sxy,var_x,var_y,cov : float;
121                          temperature_in : float
122                        end;

123       calibrate_type = record
124                          state           : cal_state_type;
125                          p_stats,s_stats  : stats_type;
126                        end;

127       param_array_type = array[1..space_for_parameters] of data_block_type;

128       param_type       = record
129                            valid : boolean;
130                            p_stats,s_stats  : stats_type;
131                            case  integer of
132                              0 : (r_coil, r_heater : data_block_type);
133                              1 : (data : param_array_type);
```

```
134                         end;

135       drive_type     = record
136                          number                : integer;
137                          com                   : com_type;
138                          f                     : file_type;
139                          longint_precision_adc : longint;
140                          p_dac, s_dac, h_dac   : string[4];
141                          p_code,s_code,h_code  : longint;
142                          s_trim                : integer;
143                          vref,
144                          fp_setpoint, fp_precision_adc,
145                          fp_p_dac, fp_s_dac, fp_h_dac,
146                          fp_agnd, fp_d1, fp_d2,
147                          fp_temperature_in, fp_temperature_out,
148                          fp_last_setpoint      : float;
149                          nearest_index         : integer;
150                          precision_adc,agnd,d1,d2,sensor_in,sensor_out : string;
151                          loop_state    : loop_state_type;
152                          ticks_Y       : longint;
153                          ticks_loop    : longint;
154                          padc_filter   : filter_type;
155                          fp_mean_error : float;
156                          loop_time     : loop_time_type;
157                          monitor_count : integer;
158                          cal           : calibrate_type;
159                          p             : param_type;
160                          fp_power      : float;
161                        end;

162       host_type      = record
163                          com           : com_type;
164                          f             : file_type;
165                          fp_setpoint   : float;
166                          drive_number  : integer;
167                          drive_id      : char;
168                          fault         : boolean;
169                        end;

170       source_type    = (host_pc,keyboard,calibrate);
171       warning_type   = (exit,abort,calib);

172       keyboard_type = record
173                         data       : data_block_type;
174                         x,y,f1,f2  : integer;
175                         index      : integer;
176                         first      : boolean;
177                         drive_data : array[1..max_drive] of data_block_type;
178                         kb_drive_number : integer;
179                       end;

180     const {Main}
181       com_ref : array[1..4] of com_type =
182                 ((number:1; address:$3F8; IRQ : 4),
183                  (number:2; address:$2F8; IRQ : 3),
184                  (number:3; address:$3E8; IRQ : 5),
185                  (number:4; address:$2E8; IRQ : 7));
186       screen_offset_hex : array[1..max_drive] of integer = (39,67);
187       screen_offset_fp  : array[1..max_drive] of integer = (25,53);
188       monitor_char      : array[1..4] of char = ('|','-','|','-');

189     var {Main}
190       drive     : array[1..max_drive] of drive_type;
191       host      : host_type;
192       kb        : keyboard_type;
193       done,flag : boolean;
194       source    : source_type;
195       ticks_clock,tries,loop_count, reply_count : longint;
196       ExitSave  : pointer;
197       j,precision_adc_filter_length : integer;
198       ft        : text;
199       fp        : file of param_type;

200     function sign(x : float) : integer;
```

```
201     begin
202       if x>0 then
203         sign := 1
204       else
205         if x<0 then
206           sign := -1
207         else
208           sign := 0;
209     end;

210     function right(s : string; n:integer) : string;
211     var
212       start : integer;
213     begin
214       start := length(s)-n+1;
215       if start<1 then
216         start := 1;
217       right := copy(s,start,n);
218     end;

219     function file_exists(name: string) : boolean;
220     var
221       f : file;
222     begin
223       {$I-}
224       assign(f,name);
225       reset(f);
226       close(f);
227       {I+}
228       file_exists := (IOResult=0) and (name<>'');
229     end;

230     procedure white_text;
231     begin
232       TextAttr := white;
233     end;

234     procedure white_text_with_underline;
235     begin
236       TextAttr := 9;
237     end;

238     procedure grey_text;
239     begin
240       TextAttr := lightgray;
241     end;

242     procedure blinking_white_text;
243     begin
244       TextAttr := $80+white;
245     end;

246     function ticks : longint;
247     begin
248       ticks := longint(Ptr($40,$6C)^); {Specific location holds real-time tick}
249     end;

250     procedure message(s : string; delay_in_milli_seconds : integer);
251     var
252       xmin,xmax,width,depth : byte;
253     begin {message}
254       width := length(s);
255       if delay_in_milli_seconds<>0 then
256         depth := 4
257       else
258         depth := 5;
259       xmin := 40-width div 2 - 2;
260       xmax := 40+width div 2 + 2;
261       openwindow(xmin,10,xmax,10+depth,' message ',white,$7A);
262       put_string(2,2,s);
263       if delay_in_milli_seconds=0 then
264         begin
265           put_string(2,3,'Press any key to continue');
266           inkey
```

```
267           end
268         else
269           delay(delay_in_milli_seconds*1000);
270         closewindow;
271       end; {message}

272       function longint_to_hex(number : longint; count : integer) : string;
273       var
274         r : string;
275         i : integer;
276       const
277         hex_table : array[0..15] of char = ('0','1','2','3','4','5','6','7',
278                                             '8','9','A','B','C','D','E','F');
279       begin {longint_to_hex}
280         r := '';
281         for i := count downto 1 do
282           begin
283             r := hex_table[number and 15] + r;
284             number := number shr 4;
285           end;
286         longint_to_hex := r;
287       end; {longint_to_hex}

288       function fp_to_hex(number,scale : float; count : integer) : string;
289       begin
290         fp_to_hex := longint_to_hex(round(number*scale), count);
291       end;

292       function hex_char_to_value(ch : char) : byte;
293       begin
294         case ch of
295           '0'..'9' : hex_char_to_value := ord(ch)-ord('0');
296           'A'..'F' : hex_char_to_value := ord(ch)-ord('A')+10;
297         else
298           hex_char_to_value := 0;
299         end;
300       end;

301       function hex_to_longint(hex_string : string) : longint;
302       var
303         t : longint;
304         i : integer;
305       begin
306         if length(hex_string)=0 then
307           hex_to_longint := 0
308         else
309           begin
310             t := 0;
311             for i := 1 to length(hex_string) do
312               t :=  (t shl 4) + hex_char_to_value(hex_string[i]);
313             hex_to_longint := t
314           end;
315       end;

316       function p_adc_hex_to_fp(hex_string:string) : float;
317       begin
318         p_adc_hex_to_fp := 5.0*(hex_to_longint(hex_string)-$400000)/$200000;
319       end;

320       function max132_hex_to_fp(hex_string:string; v : float) : float;
321       var
322         t:string;
323       begin
324         if hex_string[1]='0' then
325           t := '00'
326         else
327           t := 'FF';
328         max132_hex_to_fp :=
329          v*(hex_to_longint(t+hex_string))/(655*512);
330       end;

331       procedure put_highlighted_string(x,y:integer; s:string; i:integer);
332       begin
333         grey_text;
```

```
334         put_string(x,y,s);
335         change_attribute(x+i-1,y,$9);
336         white_text;
337       end;

338       procedure edit_keyboard_entry;
339         procedure delete_char( var d : data_block_type; i : integer);
340         begin
341           with d do vs := copy(vs,1,i-1)+right(vs,length(vs)-i)+' ';
342         end;

343         procedure insert_char( var d : data_block_type; var i:integer; code:integer);
344         begin
345           with d do
346             begin
347               vs := copy(vs,1,i-1)+chr(code)+copy(vs,i,length(vs)-i);
348               if i<length(vs) then inc(i);
349             end;
350         end;

351         procedure validate_data( var d:data_block_type; f1,f2 : integer);
352         var
353           error_code : integer;
354         begin
355           with d do
356             begin
357               val(strip_spaces(vs),v,error_code);
358               if error_code=0 then
359                  update(d,d.v,f1,f2);
360             end;
361         end;

362       begin {edit_keyboard_entry}
363         with kb do
364           begin
365             case last_key_code of
366               CHome              : begin
367                                      data.vs := spaces(9);
368                                      index   := 1;
369                                    end;
370               HOME               : index := 1;
371               Endkey             : index := length(data.vs);
372               Delkey             : delete_char(data,index);
373               RightArrow         : wrap_up(index,length(data.vs));
374               LeftArrow          : wrap_down(index,length(data.vs));
375               BS                 : begin
376                                      if index>1 then
377                                        dec(index);
378                                      delete_char(data,index);
379                                    end;
380               ord('-'),ord('.'),ord(' '),ord('0')..ord('9')
381                    : begin
382                        if first then
383                          begin
384                            data.vs := spaces(length(data.vs));
385                            case last_key_code of
386                              ord('-') : index := 1;
387                              ord('.') :
388                                begin
389                                  data.vs := spaces(f1-f2-2)+'0'+spaces(f2+1);
390                                  index := f1-f2;
391                                end;
392                              else
393                                index := 2;
394                            end;
395                          end;
396                        insert_char(data,index,last_key_code);
397                      end;
398               CR,TAB,STAB,DownArrow,UpArrow  : validate_data(data,f1,f2);
399             end;
400             first := false;
401             if data.vv then
402               begin
403                 grey_text;
```

```
404               put_string(x,y,data.vs);
405               white_text;
406             end
407           else
408            put_highlighted_string(x,y,data.vs,index);
409         end;
410       end; {edit_keyboard_entry}

411       procedure start_keyboard_entry(value:float; xd,yd,f1d,f2d : integer);
412       begin
413         with kb do
414           begin
415             f1 := f1d; f2 := f2d; x := xd; y := yd;
416             update(data,value,f1,f2);
417             index := length(data.vs);
418             first := true;
419             put_highlighted_string(x,y,data.vs,index);
420             data.vv := false;
421           end;
422       end;

423       procedure start_drive_entry(n : integer);
424       begin
425         with kb do
426           begin
427             kb_drive_number := n;
428             start_keyboard_entry(drive[n].fp_setpoint,screen_offset_fp[n],8,9,6);
429           end;
430       end;

431       procedure set_filter (var f : filter_type;
432                             new_value:float;
433                             new_length,new_guard : integer);
434       var
435         i : integer;
436       begin {set_filter}
437         with f do
438           begin
439             length := new_length;
440             guard  := new_guard;
441             for i  := 1 to length do history[i] := new_value;
442             index := 1;
443             count := 0;
444             mean  := new_value;
445             valid := false;
446           end;
447       end; {set_filter}

448       procedure apply_filter (var f : filter_type; new_value : float);
449       var
450         i   : integer;
451         sum : float;
452       begin {apply_filter}
453         with f do
454           begin
455             history[index] := new_value;
456             inc(count);
457             valid := valid or (count>=length+guard);
458             wrap_up(index,length);
459             sum := 0;
460             for i := 1 to length do sum := sum+history[i];
461             mean := sum/length;
462           end;
463       end; {apply_filter}

464       procedure set_stats( var s : stats_type);
465       var
466         i : integer;
467       begin
468         with s do
469           begin
470             n := 0; sx := 0; sx2 := 0; sy := 0; sy2 := 0; sxy := 0;
471             for i := 1 to stats_data_max do
472               begin
```

```
473               x[i] := 0; y[i] := 0; error[i] := 0;
474             end;
475         end;
476       end;

477       procedure enter_stats( var s: stats_type; xd,yd:float);
478       begin
479         with s do
480           begin
481             inc(n);
482             sx   := sx+xd;
483             sx2  := sx2+sqr(xd);
484             sy   := sy+yd;
485             sy2  := sy2+sqr(yd);
486             sxy  := sxy+xd*yd;
487             x[n] := xd;
488             y[n] := yd;
489           end;
490       end;

491       procedure apply_stats(var s: stats_type);
492       var
493         i : integer;
494       begin
495         with s do
496           if n>1 then
497             begin
498               xb := sx/n; yb := sy/n;
499               var_x := (sx2-sqr(sx)/n)/n;
500               var_y := (sy2-sqr(sy)/n)/n;
501               b := (sxy-(sx*sy)/n)/(sx2-sqr(sx)/n);
502               a := 1/n*(sy-b*sx);
503               cov := 0;
504               for i := 1 to n do
505                 begin
506                   error[i] := y[i]-(a+b*x[i]);
507                   cov := cov+(x[i]-xb)*(y[i]-yb);
508                 end;
509               cov := cov/n;
510               r := cov/sqrt(var_x*var_y);
511             end
512           else
513             begin
514               a := 0; b := y[1]; error[1] := 0; r := 1;
515             end;
516       end;

517       procedure report_time_of_last_calibration;
518       var
519         time : longint;
520         DT   : DateTime; { from dos unit }
521       begin
522         GetFTime(fp,time);
523         UnpackTime(time,DT);
524         grey_text;
525         with DT do put_string(42,21,' Last calibration '+
526                               copy(time_string(Hour,Min,Sec),1,5)+
527                               ' on '+date_string(Year,Month,Day));
528         put_string(42,23,spaces(37));
529         white_text;
530       end;

531       procedure report_current_time;
532       var
533         Year,Month,Day,DayofWeek,Hour,Min,Sec,Sec100 : word;
534       begin
535         if abs(ticks-ticks_clock)>ticks_clock_max then
536           begin
537             ticks_clock := ticks;
538             GetDate(Year,Month,Day,DayofWeek);
539             GetTime(Hour,Min,Sec,Sec100);
540             grey_text;
541             put_string(50,22,'Currently '+ copy(time_string(Hour,Min,Sec),1,5)+
542                        ' on '+date_string(Year,Month,Day));
```

```
543           white_text;
544         end;
545       end;

546       procedure write_all_parameters;
547         procedure write_stats(var s : stats_type);
548         var
549           i : integer;
550         begin
551           with s do
552             begin
553               writeln(ft,'Temperature (in) = ',temperature_in:7:4);
554               writeln(ft,'Number= ',n);
555               for i := 1 to n do
556                 writeln(ft,i:4,' ',x[i]:11:1,' ',y[i]:12:8,' ',error[i]:12:8);
557               writeln(ft,'mean x= ',xb:11:1,' mean_y :=  ',yb:12:8);
558               writeln(ft,'a= ',a,' b= ',b,' r= ',r);
559             end;
560         end;
561       var
562         i : integer;
563       begin {write_all_parameters}
564         message('Saving parameters and calibration data',3);
565         assign(ft,file_text);
566         rewrite(ft);
567         assign(fp,file_binary);
568         rewrite(fp);
569         for i := 1 to max_drive do
570           with drive[i] do
571             begin
572               writeln(ft,'Drive= ',i);
573               p.p_stats := cal.p_stats;
574               p.s_stats := cal.s_stats;
575               writeln(ft,'Coil Resistance  = ',p.r_coil.vs);
576               writeln(ft,'Heater Resistance= ',p.r_heater.vs);
577               writeln(ft,'Primary DAC statistics');
578               write_stats(p.p_stats);
579               writeln(ft,'Secondary DAC statistics');
580               write_stats(p.s_stats);
581               write(fp,p);
582               report_time_of_last_calibration;
583             end;
584         close(ft);
585         close(fp);
586       end; {write_all_parameters}

587       procedure restore_all_parameters;
588       var
589         i : integer;
590       begin {restore_all_parameters}
591         for i := 1 to max_drive do
592           begin
593             drive[i].p.valid := false;
594             for j := 1 to space_for_parameters do
595               drive[i].p.data[j] := data_block_null;
596             white_text;
597           end;
598         if file_exists(file_binary) then
599           begin
600             assign(fp,file_binary);
601             reset(fp);
602             message('Restoring parameters and calibration data',3);
603             for i := 1 to max_drive do
604               with drive[i] do
605                 begin
606                   read(fp,p);
607                   cal.p_stats := p.p_stats;
608                   cal.s_stats := p.s_stats;
609                 end;
610             report_time_of_last_calibration;
611             close(fp);
612           end
613         else
614           begin
```

```
615           for i := 1 to max_drive do drive[i].p.valid := false;
616          put_string(42,21,'No Previous Calibration');
617         end;
618       end; {restore_all_parameters}

619       procedure prepare_com_files(var ft : file_type; com : com_type);
620       type
621         direction_type = (input,output);

622         procedure prepare_com_file( var f : text; com : com_type;
623                                     direction : direction_type);
624         var
625           error : word;
626           options : _OptionRecord;
627         begin {prepare_com_file}
628           with _StdOpen do
629            begin
630              _In_Queue_Size  := 1024;
631              _Out_Queue_Size := 1024;
632              _Port_Address   := com.address;
633              _Int_Level      := com.IRQ;
634            end;
635           __assignDD(f,com.number);
636           if direction = input then
637             reset(f)
638           else
639             rewrite(f);
640           error := __RetOpDD(f,options);
641           if (error<>0) then
642             writeln('Unable to read port settings - error=',error);
643           with options do
644             begin
645               _BaudRate   := 9600;
646               _Parity     := 0;
647               _DataBits   := 8;
648               _StopBits   := 1;
649               _RequireCTS := false;
650             end;
651           error := __SetOpDD(f,options);
652           if (error<>0) then
653             writeln('Unable to set port settings - error=',error);
654         end; {prepare_com_file}
655       begin {prepare_com_files}
656         prepare_com_file(ft.inp,com,input);
657         prepare_com_file(ft.out,com,output);
658       end; {prepare_com_files}

659       procedure initialise_com( var com : com_type; channel : integer);
660       begin
661       with com do
662         begin
663           number  := com_ref[channel].number;
664           address := com_ref[channel].address;
665           IRQ := com_ref[channel].IRQ;
666           last_command := '';
667           last_reply   := '';
668           state    := idle;
669           response := true;
670         end;
671       end;

672       procedure initialise_loop_time(var d : drive_type);
673       begin
674        with d.loop_time do
675         begin
676           current_tick := ticks;
677           set_filter(filter,0,loop_time_filter_length,2);
678         end;
679       end;

680       procedure initialise_drive(var dr : drive_type; channel : integer);
681       var
682        i : integer;
683       begin
```

```
684         with dr do
685           begin
686             number := channel;
687             initialise_com(com,channel);
688             prepare_com_files(f,com);
689             case channel of
690               1 :  drive[1].vref := 0.61772; {voltage across R44 of DRive 1}
691               2 :  drive[2].vref := 0.61700; {voltage across R44 of DRive 2}
692             end;
693             loop_state := loop_idle;
694             longint_precision_adc := 0;
695             s_trim      := 0;
696             fp_setpoint := 0;
697             fp_last_setpoint := 1.0e10 ; {impossibly large to force lookup}
698             fp_precision_adc := 0;
699             fp_p_dac := 0.0;
700             fp_s_dac := 0.0;
701             fp_h_dac := 0.0;
702             fp_agnd  := 0.0;
703             fp_d1 := 0.0;
704             fp_d2 := 0.0;
705             fp_temperature_in  := 0.0;
706             fp_temperature_out := 0.0;
707             precision_adc := '';
708             agnd := '';
709             d1   := '';
710             d2   := '';
711             sensor_in  := '';
712             sensor_out := '';
713             loop_state := loop_idle;
714             ticks_Y    := ticks-ticks_Y_max;
715             ticks_loop := ticks-ticks_loop_max;
716             set_filter(padc_filter,fp_setpoint,precision_adc_filter_length,2);
717             initialise_loop_time(dr);
718             cal.state  := cal_idle;
719             monitor_count := 1;
720           end;
721       end; {initialise_drive}

722       procedure initialise_host(var hs : host_type; channel : integer);
723       begin
724         with hs do
725           begin
726             initialise_com(com,channel);
727             prepare_com_files(f,com);
728             fp_setpoint := 0;
729             fault := false;
730           end;
731       end;

732       procedure send_drive_command(var dr : drive_type; message : string);
733       begin
734         with dr do
735           begin
736           {$I-}
737             write(f.out,message+chr(13));
738           {$I+}
739             with com do
740               begin
741                 last_command := message;
742                 last_reply := '';
743                 error := IOResult;
744                 state := command_sent;
745                 ticks_timeout := ticks;
746               end;
747           end;
748       end; {send_drive_command}

749       procedure send_host_reply(var hs : host_type; message : string);
750       begin
751         with hs do
752           begin
753             {$I-}
754             write(f.out,'K'+message+chr(04));
```

```
755             {$I+}
756             with com do
757               begin
758                 last_command := '';
759                 last_reply   := message;
760                 error := IOResult;
761                 state := reply_sent;
762               end;
763           end;
764       end; {send_host_reply}

765       procedure get_drive_reply(var dr : drive_type);
766       const
767         ack_string  = 'Y'+chr(13)+chr(10);
768         nack_string = 'N'+chr(13)+chr(10);
769       var
770         ch  : char;
771         len : integer;
772         ts  : string;
773       begin {get_drive_reply}
774         with dr.com do
775           begin
776             repeat
777               read(dr.f.inp,ch);
778               if ord(ch)<>26 then
779                 begin
780                   last_reply := last_reply+ch;
781                   if right(last_reply,length(ack_string))=ack_string then
782                     state := ack_received;
783                   if right(last_reply,length(nack_string))=nack_string then
784                     state := nack_received;
785                 end;
786             until ord(ch)=26;
787             if state=ack_received then
788               begin
789                 if pos(last_command,last_reply)=1 then
790                   begin
791                     last_reply := copy(last_reply,length(last_command)+3,
792                                        length(last_reply)-length(last_command)-5);
793                     state := valid_reply;
794                     inc(reply_count);
795                     if not response then
796                       begin
797                         response := true;
798                         put_string(screen_offset_hex[dr.number],2,spaces(12));
799                       end;
800                   end;
801               end;
802             if abs(ticks-ticks_timeout)>ticks_timeout_max then
803               begin
804                 send_drive_command(dr,dr.com.last_command);
805                 response := false;
806                 gotoxy(screen_offset_hex[dr.number],2);
807                 blinking_white_text;
808                 Write('No Response');
809                 white_text;
810               end;
811           end;
812       end; {get_drive_reply}

813       procedure get_host_command(var hs : host_type);
814       const
815         end_string = chr(4);
816       var
817         ch  : char;
818         len : integer;
819         ts  : string;
820       begin {get_host_command}
821         with hs.com do
822           begin
823             repeat
824               read(hs.f.inp,ch);
825               if ord(ch)<>26 then
826                 begin
```

```
827                 last_command := last_command+ch;
828                 if right(last_command,length(end_string))=end_string then
829                   state := command_received;
830               end;
831           until ord(ch)=26;
832           if (state=command_received) and
833              (length(last_command)=13) and
834              (copy(last_command,1,2)='K0') and
835              (copy(last_command,4,1)='V') then
836              with hs do
837                begin
838                drive_id := hs.com.last_command[3];
839                if ord(drive_id)>128 then
840                  drive_number := ord(drive_id)-$80
841                else
842                  drive_number := ord(drive_id)-ord('0');
843                fp_setpoint := 5.0/8388608*hex_to_longint(copy(hs.com.last_command,5,8));
844                if (drive_number in[1..2]) and
845                        (fp_setpoint<5.0) and
846                        (fp_setpoint>-5.0) then
847                  state := valid_command;
848              end;
849        end;
850     end; {get_host_command}

851     procedure put_shaded_number(x,y:integer; number:float;
852                                 f1,f2:integer; flag:boolean);
853     begin
854       if flag then
855         white_text
856       else
857         grey_text;
858       put_number(x,y,number,f1,f2);
859       white_text;
860     end; {put_shaded_number}

861     function find_nearest_index(s:stats_type; yd:float) : integer;
862     var
863       i,j : integer;
864       min_difference,difference : float;
865     begin {find_nearest_index}
866       min_difference := 1e5;
867       with s do
868         begin
869           for i := 1 to n do
870             begin
871               difference := abs(s.y[i]-yd);
872               if difference<min_difference then
873                 begin
874                   j := i;
875                   min_difference := difference;
876                 end;
877             end;
878         end;
879       find_nearest_index := j;
880     end; {find_nearest_index}


881     procedure control_loop;
882     var
883       drive_number : integer;
884       procedure calc_and_disp_aux_adc;
885       begin
886         with drive[drive_number] do
887           begin {calc_and_disp_aux_adc}
888             grey_text;
889             put_string(screen_offset_hex[drive_number],15,d1);
890             put_string(screen_offset_hex[drive_number],16,d2);
891             put_string(screen_offset_hex[drive_number],17,sensor_in);
892             put_string(screen_offset_hex[drive_number],18,sensor_out);
893             white_text;
894             fp_agnd := max132_hex_to_fp(agnd,vref);
895             fp_d1   := (max132_hex_to_fp(d1,vref)-fp_agnd)*24.2;
896             fp_d2   := (max132_hex_to_fp(d2,vref)-fp_agnd)*24.2;
```

```
897             fp_temperature_in  := (max132_hex_to_fp(sensor_in,vref )-fp_agnd)*100.0;
898             fp_temperature_out := (max132_hex_to_fp(sensor_out,vref)-fp_agnd)*100.0;
899             put_number(screen_offset_fp[drive_number]-1,15,fp_d1,8,4);
900             put_number(screen_offset_fp[drive_number]-1,16,fp_d2,8,4);
901             put_number(screen_offset_fp[drive_number],17,fp_temperature_in,5,2);
902             put_number(screen_offset_fp[drive_number],18,fp_temperature_out,5,2);
903           end;
904       end; {calc_and_disp_aux_adc}

905       procedure calc_and_disp_precision_adc;
906       var
907         i : integer;
908         sum : float;
909       begin {calc_and_disp_precision_adc}
910         with drive[drive_number] do
911           begin
912             grey_text;
913             put_string(screen_offset_hex[drive_number],6,precision_adc);
914             white_text;
915             put_number(screen_offset_fp[drive_number],5,fp_setpoint,9,6);
916             put_number(screen_offset_fp[drive_number],6,fp_precision_adc,9,6);
917             apply_filter(padc_filter,fp_precision_adc);
918             fp_mean_error := padc_filter.mean-fp_setpoint;
919             put_shaded_number(screen_offset_fp[drive_number],7,
920                               fp_mean_error,9,6,padc_filter.valid);
921             wrap_up(monitor_count,4);
922             put_string(screen_offset_fp[drive_number]+10,6,
923                        monitor_char[monitor_count]);
924           end;
925       end; {calc_and_disp_precision_adc}

926       procedure display_settings;
927       var
928         sum : longint;
929         i   : integer;
930       begin {display_settings}
931         with drive[drive_number] do
932           begin
933             grey_text;
934             put_string(screen_offset_hex[drive_number],10,p_dac);
935             put_string(screen_offset_hex[drive_number],11,s_dac);
936             put_string(screen_offset_hex[drive_number]+5,11,
937                        longint_to_hex(s_trim,2));
938             put_string(screen_offset_hex[drive_number],12,h_dac);
939             white_text;
940             put_number(screen_offset_fp[drive_number],10,fp_p_dac,8,5);
941             put_number(screen_offset_fp[drive_number],11,fp_s_dac,8,5);
942             put_number(screen_offset_fp[drive_number],12,fp_h_dac,8,5);
943             put_number(screen_offset_fp[drive_number],13,fp_power,5,2);
944             with loop_time do
945               begin
946                 last_tick    := current_tick;
947                 current_tick := ticks;
948                 apply_filter(filter,abs(current_tick-last_tick));
949                 put_shaded_number(screen_offset_fp[drive_number],19,
950                                   filter.mean/18.2,5,2,filter.valid);
951               end;
952           end;
953       end; {display_settings}

954       procedure auto_trim( var d : drive_type);
955       var
956         difference : float;
957       begin
958         with d do
959           begin
960             if padc_filter.valid then
961               begin
962                 difference := (fp_setpoint-padc_filter.mean);
963                 if abs(difference)>max_error then
964                   begin
965                     s_trim := s_trim+sign(difference);
966                     set_filter(padc_filter,fp_setpoint,4,0);
967                   end;
```

```
968               end;
969             end;
970         end; {auto_trim}

971         procedure maintain_constant_power( var d : drive_type);
972         var
973           max_power : float;
974         begin {maintain_constant_power}
975           with d do
976             begin
977               max_power := sqr(5/r_ref)*p.r_coil.v;
978               fp_h_dac  := r_ref*sqrt((max_power-sqr(fp_setpoint/r_ref)*p.r_coil.v)/
979                            p.r_heater.v);
980               fp_power  := sqr(fp_h_dac/r_ref)*p.r_heater.v+
981                            sqr(fp_setpoint/r_ref)*p.r_coil.v;
982             end;
983         end; {maintain_constant_power}

984         procedure set_dacs(var d : drive_type);
985         begin
986           with d do
987             begin
988               case cal.state of
989                 cal_idle,cal_finished :
990                   begin
991                     if fp_setpoint<>fp_last_setpoint then
992                       nearest_index  := find_nearest_index(cal.p_stats,fp_setpoint);
993                     fp_last_setpoint := fp_setpoint;
994                     p_code := round(cal.p_stats.x[nearest_index]);
995                     s_code := round((fp_setpoint-cal.p_stats.y[nearest_index])/
996                                     cal.s_stats.b)+s_trim;
997                     fp_p_dac := p_code*5.0/32768;
998                     fp_s_dac := s_code*5.0/32768;
999                     maintain_constant_power(d);
1000                    h_code   := round(fp_h_dac/5.0*32768);
1001                    auto_trim(d);
1002                  end;
1003                cal_p    :
1004                  begin
1005                    fp_p_dac := fp_setpoint;
1006                    fp_s_dac := 0;
1007                    maintain_constant_power(d);
1008                    p_code := round(fp_p_dac/5.0*32768);
1009                    s_code := 0;
1010                    h_code := round(fp_h_dac/5.0*32768);
1011                  end;
1012                cal_s    :
1013                  begin
1014                    fp_p_dac := 0;
1015                    fp_s_dac := fp_setpoint*64;
1016                    maintain_constant_power(d);
1017                    p_code := 0;
1018                    s_code := round(fp_s_dac/5.0*32768);
1019                    h_code := round(fp_h_dac/5.0*32768);
1020                  end;
1021            end;
1022            p_dac := longint_to_hex(p_code,4);
1023            s_dac := longint_to_hex(s_code,4);
1024            h_dac := longint_to_hex(h_code,4);
1025          end;
1026      end; {set_dacs}

1027    begin {control_loop}
1028      for drive_number := 1 to 2 do
1029      with drive[drive_number] do
1030        begin
1031          case drive[drive_number].loop_state of
1032            loop_idle  :
1033              begin
1034                if abs(ticks-ticks_loop) > ticks_loop_max then
1035                  begin
1036                    send_drive_command(drive[drive_number],'x');
1037                    loop_state := sent_X;
1038                    ticks_loop := ticks;
```

```
1039                    end;
1040                end;
1041              sent_X     :
1042                begin
1043                  get_drive_reply(drive[drive_number]);
1044                  if com.state=valid_reply then
1045                    begin
1046                      precision_adc    := com.last_reply;
1047                      fp_precision_adc :=  p_adc_hex_to_fp(precision_adc);
1048                      calc_and_disp_precision_adc;
1049                      loop_state := received_X;
1050                    end;
1051                end;
1052              received_X :
1053                begin
1054                   if abs(ticks-ticks_Y) > ticks_Y_max then
1055                     begin
1056                       send_drive_command(drive[drive_number],'y');
1057                       loop_state := sent_Y;
1058                       ticks_Y    := ticks;
1059                     end
1060                   else
1061                     loop_state := received_Y;
1062                end;
1063              sent_Y     :
1064                begin
1065                  get_drive_reply(drive[drive_number]);
1066                  if com.state=valid_reply then
1067                    with drive[drive_number] do
1068                      begin
1069                        agnd := copy(com.last_reply,1,6);
1070                        d1   := copy(com.last_reply,13,6);
1071                        d2   := copy(com.last_reply,19,6);
1072                        sensor_in  := copy(com.last_reply,25,6);
1073                        sensor_out := copy(com.last_reply,31,6);
1074                        calc_and_disp_aux_adc;
1075                        loop_state := received_Y;
1076                      end;
1077                end;
1078              received_Y :
1079                begin
1080                   set_dacs(drive[drive_number]);
1081                   send_drive_command(drive[drive_number],'z'+s_dac+p_dac+h_dac);
1082                   loop_state := sent_Z;
1083                end;
1084              sent_Z     :
1085                begin
1086                  get_drive_reply(drive[drive_number]);
1087                  if com.state=valid_reply then
1088                    begin
1089                      loop_state := received_Z;
1090                      display_settings;
1091                    end;
1092                end;
1093              received_Z :
1094                begin
1095                  loop_state := loop_idle;
1096                  inc(loop_count);
1097                end;
1098            end;
1099        end;
1100    end; {control_loop}

1101    procedure display_options(x,y : integer; s:string);
1102    var
1103      i : integer;
1104    begin {display_options}
1105      grey_text;
1106      put_string(x,y,'Options');
1107      gotoxy(x,y+2);
1108      for i := 1 to length(s) do
1109        begin
1110          if s[i] in ['A'..'Z'] then
1111            white_text_with_underline
```

```
1112            else
1113              grey_text;
1114            write(s[i]);
1115          end;
1116      end; {display_options}

1117      procedure draw_main_form;
1118      begin
1119        clrscr;
1120        horizontal_line(1,78,20,1,TextAttr);
1121        vertical_line(40,21,23,1,TextAttr);
1122        put_string2(31,2,'DRIVE 1',59,2,'DRIVE 2');
1123        put_string2(26,3,'Floating',54,3,'Floating');
1124        put_string2(3,5,'Set Point',20,5,'V');
1125        put_string2(3,6,'Precision ADC',20,6,'V');
1126        put_string2(3,7,'Mean Difference',20,7,'V');
1127        put_string2(3,8,'Keyboard Entry',20,8,'V');
1128        put_string2(3,10,'Primary DAC',20,10,'V');
1129        put_string2(3,11,'Secondary DAC',20,11,'V');
1130        put_string2(3,12,'Heater_DAC',20,12,'V');
1131        put_string2(3,13,'Power',20,13,'W');
1132        put_string2(3,15,'Amplifier 1',20,15,'V');
1133        put_string2(3,16,'Amplifier 2',20,16,'V');
1134        put_string2(3,17,'Temperature in',19,17,chr(248)+'C');
1135        put_string2(3,18,'Temperature out',19,18,chr(248)+'C');
1136        put_string2(3,19,'Mean Loop Time',20,19,'s');
1137        display_options(3,21,'Abort Calibrate Exit Host Keyboard');
1138        grey_text;
1139        put_string(3,2,'Source=');
1140        put_string2(39,3,'HEX',67,3,'HEX');
1141        white_text;
1142      end; {draw_main_form}

1143      procedure cursor_control(a,b : byte);
1144      var
1145        Regs : Registers;
1146      begin
1147        Regs.AH := 1;
1148        Regs.CH := a;
1149        Regs.CL := b;
1150        Intr(16,Regs);
1151      end;

1152      procedure cursor_on;
1153      begin
1154        cursor_control(11,12);
1155      end;

1156      procedure cursor_off;
1157      begin
1158        cursor_control($20,12);
1159      end;

1160      {$F+}
1161      procedure MyExit;
1162      begin
1163        ExitProc := ExitSave;
1164        close(drive[1].f.inp);
1165        close(drive[1].f.out);
1166        close(drive[2].f.inp);
1167        close(drive[2].f.out);
1168        cursor_on;
1169        closewindow;
1170        clrscr;
1171      end; {MyExit}
1172      {$F-}

1173      function warning(typ : warning_type) : boolean;
1174      const
1175        message : array[0..2]  of string[25] =
1176                  ('OK to Exit?     Y/N',
1177                   'OK to Abort?    Y/N',
1178                   'OK to Calibrate Y/N');
1179      var
```

```
1180        xmin,xmax,width,depth : byte;
1181        reply,done : boolean;
1182      begin {warning}
1183        width := length(message[ord(typ)]);
1184        depth := 4;
1185        xmin  := 40-width div 2 - 2;
1186        xmax  := 40+width div 2 + 2;
1187        openwindow(xmin,10,xmax,10+depth,' warning ',white,$FA);
1188        put_string(2,2,message[ord(typ)]);
1189        repeat
1190          inkey;
1191          case last_key_code of
1192          ord('y'),ord('Y') : begin
1193                                reply := true;
1194                                done  := true;
1195                              end;
1196          ord('n'),ord('N') : begin
1197                                reply := false;
1198                                done  := true;
1199                              end;
1200          end;
1201        until done;
1202        warning := reply;
1203        closewindow;
1204      end; {warning}

1205      procedure flush_host(var hs : host_type);
1206      var
1207        ch : char;
1208      begin {flush_host}
1209        repeat
1210          read(hs.f.inp,ch);
1211        until ord(ch)=26;
1212      end; {flush_host}

1213      function test_all_parameters : boolean;
1214      var
1215        i,j  : integer;
1216        flag : boolean;
1217      begin {test_all_parameters}
1218        flag  := true;
1219        for i := 1 to max_drive do
1220          with drive[i].p do
1221            begin
1222              valid := true;
1223              for j := 1 to number_of_parameters do
1224                valid := valid and data[j].vv;
1225              flag  := flag and valid;
1226            end;
1227        test_all_parameters := flag;
1228      end; {test_all_parameters}

1229      procedure edit_parameters;
1230      const
1231          screen_offset_param  : array[1..max_drive] of integer = (29,45);
1232      var
1233        i,j        : integer;
1234        local_done : boolean;
1235        procedure draw_form;
1236        var
1237          i,j : integer;
1238        begin {draw_form}
1239          clrscr;
1240          horizontal_line(1,57,7,1,TextAttr);
1241          put_string2(30,2,'DRIVE 1',46,2,'DRIVE 2');
1242          put_string2(3,4,'Coil Resistance',23,4,chr(234));
1243          put_string2(3,5,'Heater Resistance',23,5,chr(234));
1244          display_options(3,8,'Continue');
1245          for i := 1 to max_drive do
1246            for j := 1 to number_of_parameters do
1247              with drive[i].p.data[j] do
1248                if vv then
1249                  put_string(screen_offset_param[i],3+j,vs);
1250        end; {draw_form}
```

```
1251        procedure next_entry;
1252        begin
1253          with drive[i].p do
1254           begin
1255             if not data[j].vv then
1256               update(data[j],0,7,3);
1257             start_keyboard_entry(data[j].v,screen_offset_param[i],j+3,7,3);
1258           end;
1259        end; {next_entry}
1260        procedure cycle_pointers;
1261        begin
1262          case last_key_code of
1263            CR        : begin
1264                          wrap_up(j,number_of_parameters);
1265                          if j=1 then wrap_up(i,max_drive);
1266                        end;
1267            TAB,STAB  : wrap_up(i,max_drive);
1268            UpArrow   : if j>1 then dec(j);
1269            DownArrow : if j<number_of_parameters then inc(j);
1270          end;
1271          next_entry;
1272        end; {cycle_pointers}

1273      begin {edit_parameters}
1274        openwindow(11,5,69,16,' Parameters ',white,white);
1275        draw_form;
1276        local_done := false;
1277        i := 1;
1278        j := 1;
1279        next_entry;
1280        repeat
1281          get_key;
1282          case last_key_code of
1283            ord('c'),ord('C') : begin
1284                                  local_done := test_all_parameters;
1285                                  if not local_done then
1286                                    message('All parameters must be defined',2);
1287                                end;
1288            0 : ;
1289          else
1290            edit_keyboard_entry;
1291          end;
1292          case last_key_code of
1293            CR,TAB,STAB,DownArrow,UpArrow  : if kb.data.vv then
1294                                               begin
1295                                                 drive[i].p.data[j] := kb.data;
1296                                                 cycle_pointers
1297                                               end
1298                                             else
1299                                               message('Invalid Entry',1);
1300          end;
1301        until local_done;
1302        closewindow;
1303      end; {edit_parameters}

1304      procedure execute_host_command(var hs : host_type);
1305      begin
1306        case hs.com.state of
1307          valid_command :
1308            begin
1309              drive[hs.drive_number].fp_setpoint := hs.fp_setpoint;
1310              { possibly change length of filter depending on the difference
1311                between hs.fp_setpoint and drive[?].fp_setpoint }
1312              set_filter(drive[hs.drive_number].padc_filter,
1313                         drive[hs.drive_number].fp_setpoint,
1314                         precision_adc_filter_length,2);
1315              send_host_reply(hs,hs.drive_id);
1316            end;
1317          command_received :
1318            begin
1319              send_host_reply(hs,'E');
1320            end;
1321        end;
1322      end; {execute_host_command}
```

```
1323      procedure set_source(src : source_type);
1324      var
1325        i : integer;
1326      begin {set_source}
1327        source := src;
1328        for i := 1 to max_drive do put_string(screen_offset_fp[i],8,spaces(12));
1329        white_text;
1330        case source of
1331          host_pc   : begin
1332                        put_string(10,2,'HOST PC    ');
1333                        flush_host(host);
1334                        for i := 1 to max_drive do
1335                          with drive[i] do
1336                            begin
1337                              cal.state := cal_idle;
1338                              fp_setpoint := 0;
1339                              set_filter(padc_filter,fp_setpoint,
1340                                         precision_adc_filter_length,3);
1341                              initialise_loop_time(drive[i]);
1342                            end;
1343                      end;
1344          keyboard  : put_string(10,2,'KEYBOARD   ');
1345          calibrate : begin
1346                        put_string(10,2,'CALIBRATION');
1347                        edit_parameters;
1348                        for i := 1 to max_drive do
1349                          drive[i].cal.state := cal_start;
1350                      end;
1351        end;
1352      end; {set_source}

1353      procedure keyboard_activity;
1354      var
1355        i : integer;
1356        s : string;
1357      begin {keyboard_activity}
1358        get_key;
1359        case source of
1360        keyboard :
1361          case last_key_code of
1362            ESC               : set_source(host_pc);
1363            ord('c'),ord('C') : if warning(calib) then
1364                                  set_source(calibrate);
1365            ord('e'),ord('E') : done := warning(exit);
1366            ord('h'),ord('H') : set_source(host_pc);
1367            0                 : { no key => no action };
1368          else
1369            with kb do
1370              begin
1371                edit_keyboard_entry;
1372                if data.vv then
1373                  begin
1374                    case kb_drive_number of
1375                      1 : begin
1376                            drive_data[1] := data;
1377                            start_drive_entry(2);
1378                          end;
1379                      2 : begin
1380                            if last_key_code=TAB then
1381                              start_drive_entry(1)
1382                            else
1383                              if abs(drive_data[1].v-data.v)<=
1384                                     differential_range then
1385                                begin
1386                                  drive_data[2] := data;
1387                                  drive[1].fp_setpoint := drive_data[1].v;
1388                                  set_filter(drive[1].padc_filter,
1389                                             drive[1].fp_setpoint,
1390                                             precision_adc_filter_length,3);
1391                                  drive[2].fp_setpoint := drive_data[2].v;
1392                                  set_filter(drive[2].padc_filter,
1393                                             drive[2].fp_setpoint,
1394                                             precision_adc_filter_length,3);
```

```
1395                                start_drive_entry(1);
1396                              end
1397                            else
1398                              begin
1399                                str(differential_range:3,s);
1400                                message('Differential is greater than '+s,0);
1401                                start_drive_entry(1);
1402                              end
1403                          end;
1404                    end;
1405                  end;
1406              end
1407          end;
1408        host_pc :
1409          case last_key_code of
1410            ord('c'),ord('C') : if warning(calib) then
1411                                  set_source(calibrate);
1412            ord('e'),ord('E') : done := warning(exit);
1413            ord('k'),ord('K') :
1414              begin
1415                set_source(keyboard);
1416                start_drive_entry(1);
1417              end;
1418          end;
1419        calibrate :
1420          case last_key_code of
1421            ord('a'),ord('A') :  if warning(abort) then
1422                                   begin
1423                                     if file_exists(file_binary) then
1424                                       begin
1425                                         restore_all_parameters;
1426                                         set_source(host_pc);
1427                                       end
1428                                     else
1429                                       message('Cannot abort first calibration',wait);
1430                                   end;
1431            ord('e'),ord('E') : done := warning(exit);
1432          end;
1433        end;
1434      end; {keyboard_activity}

1435      procedure calibrate_cycle;
1436      const
1437        p_bits  = 7; {7}
1438        s_bits  = 6; {6}
1439        p_limit = 2*( 1 shl (p_bits-1) - 1);
1440        s_limit = 2*( 1 shl (s_bits-1) - 1);
1441        guard   = 3;

1442        procedure define_setpoint(var d : drive_type; k : integer);
1443        function fun(bits,limit,count : integer) : float;
1444        const
1445          one : longint = 1; { force long integer calculations }
1446        begin {fun}
1447          fun := (one shl (16-bits))*5/32768*(count-(limit div 2));
1448        end; {fun}
1449        begin {define_setpoint}
1450          with d do
1451            begin
1452              case cal.state of
1453                cal_p : fp_setpoint :=  fun(p_bits,p_limit,cal.p_stats.n);
1454                cal_s : fp_setpoint :=  fun(s_bits,s_limit,cal.s_stats.n)/64;
1455              end;
1456              set_filter(padc_filter,fp_setpoint,precision_adc_filter_length*k,guard);
1457            end;
1458        end; {define_setpoint}

1459        procedure display_finish_time;
1460        var
1461          number_of_samples,ts,tm,th,fth,ftm : longint;
1462          Hour,Min,Sec,Sec100 : word;
1463        begin {display_finish_time}
1464          number_of_samples := 4*precision_adc_filter_length + 2*guard +
1465                               (precision_adc_filter_length+guard)*(p_limit+s_limit);
```

```
1466          if drive[1].loop_time.filter.valid then
1467            ts := round(drive[1].loop_time.filter.mean/18.2*number_of_samples)
1468          else
1469            ts := round(number_of_samples*ticks_loop_max/18);
1470          tm := ts div 60;
1471          if (ts mod 60)>30 then
1472            inc(tm);
1473          th := tm div 60;
1474          GetTime(Hour,Min,Sec,Sec100);
1475          ftm := (Min+tm) mod 60;
1476          fth := (Hour+th+(Min+tm) div 60) mod 24;
1477          grey_text;
1478          put_string(42,23,'Calibration until '+copy(time_string(fth,ftm,0),1,5)+
1479                           ' approx.');
1480          white_text;
1481        end; {display_finish_time}

1482      var
1483        i : integer;
1484      begin {calibrate_cycle}
1485        for i := 1 to max_drive do
1486          with drive[i].cal do
1487            case state of
1488              cal_idle   :  ;
1489              cal_start  :
1490                begin
1491                  set_stats(p_stats);
1492                  state := cal_p;
1493                  define_setpoint(drive[i],2);
1494                  drive[i].s_trim := 0;
1495                  if i=1 then
1496                    display_finish_time;
1497                end;
1498              cal_p      :
1499                begin
1500                  if drive[i].padc_filter.valid then
1501                    begin
1502                      enter_stats(p_stats,drive[i].p_code,
1503                                  drive[i].padc_filter.mean);
1504                      if p_stats.n=p_limit/2 then { record in middle of run }
1505                        p_stats.temperature_in := drive[i].fp_temperature_in;
1506                      if p_stats.n<=p_limit then
1507                        define_setpoint(drive[i],1)
1508                      else
1509                        begin
1510                          apply_stats(p_stats);
1511                          set_stats(s_stats);
1512                          state := cal_s;
1513                          define_setpoint(drive[i],2);
1514                        end;
1515                    end;
1516                end;
1517              cal_s      :
1518                begin
1519                  if drive[i].padc_filter.valid then
1520                    begin
1521                      enter_stats(s_stats,drive[i].s_code,
1522                                  drive[i].padc_filter.mean);
1523                      if s_stats.n=s_limit/2 then { record in middle of run }
1524                        s_stats.temperature_in := drive[i].fp_temperature_in;
1525                      if s_stats.n<=s_limit then
1526                        define_setpoint(drive[i],1)
1527                      else
1528                        begin
1529                          apply_stats(s_stats);
1530                          state := cal_finished;
1531                          with drive[i] do
1532                            begin
1533                              drive[i].fp_setpoint := 0;
1534                              set_filter(padc_filter,fp_setpoint,
1535                                         precision_adc_filter_length,3);
1536                            end;
1537                        end;
1538                    end;
```

```
1539              end;
1540            cal_finished :  ;
1541            end;
1542        end; {calibrate_cycle}

1543        procedure general_initialise;
1544        begin
1545          ExitSave := ExitProc;
1546          ExitProc := @MyExit;
1547          cursor_off;
1548          openwindow(1,1,80,25,' COIL CONTROLLER ',white,white);
1549          draw_main_form;
1550          loop_count := 0;
1551          reply_count := 0;
1552          tries := 0;
1553          done := false;
1554          precision_adc_filter_length := 10;
1555          ticks_clock := ticks;
1556        end; {general_initialise}

1557        begin {main}
1558          general_initialise;
1559          initialise_drive(drive[1],1);
1560          initialise_drive(drive[2],2);
1561          initialise_host(host,3);
1562          restore_all_parameters;
1563          if drive[1].p.valid and drive[2].p.valid then
1564            set_source(host_pc)
1565          else
1566            set_source(calibrate);
1567          repeat
1568            keyboard_activity;
1569            case source of
1570              host_pc :  begin
1571                           get_host_command(host);
1572                           execute_host_command(host);
1573                         end;
1574              calibrate : begin
1575                            if (drive[1].cal.state=cal_finished) and
1576                               (drive[2].cal.state=cal_finished) then
1577                              begin
1578                                drive[1].p.valid := true;
1579                                drive[2].p.valid := true;
1580                                write_all_parameters;
1581                                set_source(host_pc);
1582                              end
1583                            else
1584                              calibrate_cycle
1585                          end;
1586            end;
1587            control_loop;
1588            inc(tries);
1589            { gotoxy(10,21); write(loop_count:6,reply_count:6,tries:8); }
1590            report_current_time;
1591          until done;
1592          clrscr;
1593          halt; {close files/coms via MyExit}
1594        end. {main}
```

```
1595    Unit ccsup;
1596    { Unit for supporting keyboard, window and screen operation
1597      copied from suplv19.pas under development by D.C. Dyer
1598    }
1599    interface
1600    uses dos,crt,win;

1601    type
1602      float    = real;
1603      char_set = set of char;
1604      data_block_type = record
1605                          v  : float;
1606                          vs : string[12];
1607                          vv : boolean
1608                        end;
1609      line_string  = string[80];
1610      TitleStrPtr  = ^TitleStr;
1611      WinRecPtr    = ^WinRec;
1612      WinRec       = record
1613                       Next: WinRecPtr;
1614                       State: WinState;
1615                       Title: TitleStrPtr;
1616                       TitleAttr, FrameAttr: Byte;
1617                       Buffer: Pointer;
1618                     end;

1619    const
1620      BELL = 7;
1621      BS   = 8;
1622      LF   = 10;
1623      CR   = 13;
1624      TAB  = 9;
1625      ESC  = 27;
1626      STAB = 256+15;
1627      F1   = 256+59;  F2   = 256+60;  F3   = 256+61;  F4   = 256+62;  F5   = 256+63;
1628      F6   = 256+64;  F7   = 256+65;  F8   = 256+66;  F9   = 256+67;  F10  = 256+68;
1629      F11  = 256+133; F12  = 256+134;
1630      SF1  = 256+84;  SF2  = 256+85;  SF3  = 256+86;  SF4  = 256+87;  SF5  = 256+88;
1631      SF6  = 256+89;  SF7  = 256+90;  SF8  = 256+91;  SF9  = 256+92;  SF10 = 256+93;
1632      SF11 = 256+135; SF12 = 256+136;
1633      CF1  = 256+94;  CF2  = 256+95;  CF3  = 256+96;  CF4  = 256+97;  CF5  = 256+98;
1634      CF6  = 256+99;  CF7  = 256+100; CF8  = 256+101; CF9  = 256+102; CF10 = 256+103;
1635      CF11 = 256+137; CF12 = 256+138;
1636      AF1  = 256+104; AF2  = 256+105; AF3  = 256+106; AF4  = 256+107; AF5  = 256+107;
1637      AF6  = 256+109; AF7  = 256+110; AF8  = 256+111; AF9  = 256+112; AF10 = 256+113;
1638      AF11 = 256+137; AF12 = 256+138;
1639      ALT1 = 256+120; ALT2 = 256+121; ALT3 = 256+122; ALT4 = 256+123; ALT5 = 256+124;
1640      ALT6 = 256+125; ALT7 = 256+126; ALT8 = 256+127; ALT9 = 256+128; ALT0 = 256+129;
1641      ALTA = 256+30;  ALTB = 256+48;  ALTC = 256+46;  ALTD = 256+32;  ALTE = 256+18;
1642      ALTF = 256+33;  ALTG = 256+34;  ALTH = 256+35;  ALTI = 256+23;  ALTJ = 256+36;
1643      ALTK = 256+37;  ALTL = 256+38;  ALTM = 256+50;  ALTN = 256+49;  ALTO = 256+24;
1644      ALTP = 256+25;  ALTQ = 256+16;  ALTR = 256+19;  ALTS = 256+31;  ALTT = 256+20;
1645      ALTU = 256+22;  ALTV = 256+47;  ALTW = 256+17;  ALTX = 256+45;  ALTY = 256+21;
1646      ALTZ = 256+44;

1647      Home       = 256+71;  UpArrow    = 256+72;  PgUp       = 256+73;
1648      LeftArrow  = 256+75;  RightArrow = 256+77;  Endkey     = 256+79;
1649      DownArrow  = 256+80;  PgDn       = 256+81;  Insertkey  = 256+82;
1650      Delkey     = 256+83;  CPrtSc     = 256+114; CLeftArrow = 256+115;
1651      CRightArrow = 256+116; CEndkey   = 256+117; CPgDn      = 256+118;
1652      CHome      = 256+119; CPgUp      = 256+132;

1653      number_set     : char_set = ['0'..'9','.','-'];
1654      filename_set   : char_set = ['A'..'Z','a'..'z','0'..'9','_','^','$','~','!',
1655                                   '#','%','&','-','{','}','(',')','@','`',chr(44)];
1656      all_set        : char_set = [chr(32)..chr(126)];

1657    var
1658      page_number    : byte;
1659      relative_mode  : boolean;
1660      last_key_code  : integer;
1661      TopWindow      : WinRecPtr;
1662      WindowCount    : Integer;
1663      Regs           : Registers;
```

```
1664        procedure beep;
1665        function  extended_read_key : integer;
1666        procedure get_key;
1667        procedure inkey;
1668        function  test_key(key_code : integer) : boolean;
1669        function  spaces(number : byte) : string;
1670        function  strip_spaces(s : line_string) : string;
1671        function  date_string(year,month,day : word) : string;
1672        function  time_string(hour,minute,second : word) : string;
1673        function  htod(t : char) : byte;
1674        function  decimal(hx : string) : longint;
1675        procedure expand_window;
1676        procedure contract_window;
1677        procedure wrap_up( var n : integer; limit : integer);
1678        procedure wrap_down( var n : integer; limit : integer);
1679        procedure set_video_mode(mode : byte);
1680        procedure blink_enable(flag : boolean);
1681        procedure set_page(n : byte);
1682        procedure set_cursor(x,y : byte);
1683        procedure get_char(x,y : byte; var character,attribute : byte);
1684        procedure put_char(character, attribute : byte);
1685        procedure put_string(x,y : byte; s : string);
1686        procedure put_string2(x1,y1:byte;s1:string; x2,y2:byte;s2:string);
1687        procedure put_number(x,y : byte; n : float; f1,f2 : byte);
1688        procedure put_string_and_number(x1,y1:byte;s1:string;
1689                                        x2,y2:byte; n:float; f1,f2 :byte);
1690        procedure change_attribute(x,y,new_attribute : byte);
1691        procedure highlight_line(xmin,xmax,y,new_attribute : byte );
1692        procedure write_boarder_char(x,y,character_index,attribute : byte);
1693        procedure horizontal_line(xa,xb,y : integer ; width,attribute : byte);
1694        procedure vertical_line(x,ya,yb : integer ; width,attribute : byte);
1695        procedure left_line(x,ya,yb,width,attribute : byte);
1696        procedure right_line(x,ya,yb,width,attribute : byte);
1697        procedure top_line(xa,xb,y,width,attribute : byte);
1698        procedure bottom_line(xa,xb,y,width,attribute : byte);
1699        procedure draw_box(xmin,ymin,xmax,ymax,width,attribute : byte);
1700        procedure update(var n : data_block_type; v : float; f1,f2 : byte);
1701        procedure edit_line(xa,xb,y,next_attribute : byte; var s : line_string;
1702                            allowed_set : char_set);
1703        procedure get_full_string(xa,xb,y,next_attribute : byte; var s : line_string;
1704                                  allowed_set : char_set);
1705        procedure edit_number(xa,xb,y,edit_colour, f1,f2 : byte; var d : data_block_type);
1706        procedure get_number(xa,xb,y,edit_colour : byte; f1,f2 : byte;
1707                             var d : data_block_type);
1708        procedure full_window;
1709        procedure clear_screen_area(xa,ya,xb,yb : byte);
1710        procedure ActiveWindow(Active: Boolean);
1711        procedure OpenWindow(X1, Y1, X2, Y2: Byte; T: TitleStr; TAttr, FAttr: Byte);
1712        procedure CloseWindow;

1713      implementation

1714      var
1715        character, attribute : byte;
1716        procedure beep;
1717      begin
1718        Sound(500); Delay(25); NoSound;
1719      end;

1720      function extended_read_key : integer;
1721      { Returns 0     if no key is pressed
1722                n     if an ASCII key is pressed
1723                256+n for extended keys like F1 or Alt-1 etc.
1724                See Pages 135 and 478 of The IBM PC and PS/2 by Peter Norton }
1725      var
1726        Ch   : char;
1727        Code : integer;
1728      begin
1729        Code := 0;
1730        Ch   := #0;
1731        if KeyPressed then
1732          begin
1733            Ch := ReadKey;
```

```
1734            if Ch = #0 then
1735              begin
1736                Code := 256;
1737                Ch   := ReadKey;
1738              end;
1739          end;
1740        extended_read_key := Code + ord(Ch);
1741      end;

1742      procedure get_key;
1743      { assigns value to last_key_code }
1744      begin
1745        last_key_code := extended_read_key;
1746      end;

1747      procedure inkey;
1748      { waits for key and returns extended code }
1749      begin
1750        repeat
1751          last_key_code := extended_read_key;
1752        until last_key_code<>0;
1753      end;

1754      function test_key(key_code : integer) : boolean;
1755      { compares last_key_code with parameter }
1756      begin
1757        get_key;
1758        if last_key_code=key_code then
1759          test_key := true
1760        else
1761          test_key := false;
1762      end;

1763      function spaces(number : byte) : string;
1764      { returns a string of spaces }
1765      var
1766        i : integer;
1767        t : line_string;
1768      begin
1769        for i  := 1 to number do t[i] := ' ';
1770        t[0]   := chr(number);
1771        spaces := t;
1772      end;

1773      function  strip_spaces(s : line_string) : string;
1774      { remove leading and trailing spaces }
1775      var
1776        ts : line_string;
1777      begin
1778        ts := s;
1779        while copy(ts,1,1)=' ' do
1780          ts := copy(ts,2,length(ts)-1);
1781        while copy(ts,length(ts),1)=' ' do
1782          ts := copy(ts,1,length(ts)-1);
1783        strip_spaces := ts;
1784      end;

1785      function  date_string(year,month,day : word) : string;
1786      const
1787        month_names : array[1..12] of string[3] = (
1788                      'Jan','Feb','Mar','Apr','May','Jun',
1789                      'Jul','Aug','Sep','Oct','Nov','Dec');
1790      var
1791        ts,t : line_string;
1792      begin
1793        str(Day:2,ts);
1794        ts := ts+' '+month_names[month]+' ';
1795        str(Year,t); ts := ts+copy(t,3,2);
1796        date_string := copy(ts+spaces(8),1,9);
1797      end;

1798      function  time_string(hour,minute,second : word) : string;
1799      var
1800        ts,t : line_string;
```

```
1801      begin
1802        ts := '';
1803        str(Hour,t);   if length(t)=1 then t := '0'+t; ts := ts+t+':';
1804        str(Minute,t); if length(t)=1 then t := '0'+t; ts := ts+t+':';
1805        str(Second,t); if length(t)=1 then t := '0'+t; ts := ts+t;
1806        time_string := ts;
1807      end;

1808      function htod(t : char) : byte;
1809      { convert a hexadeximal character to byte valye 0..15 }
1810      begin
1811      case t of
1812        '0'..'9' : htod := ord(t)-ord('0');
1813        'A'..'F' : htod := ord(t)-ord('A')+10;
1814        else
1815          begin
1816            write(chr(7));
1817            htod := 0;
1818          end;
1819        end { case }
1820      end;

1821      function decimal(hx : string) : longint;
1822      { convert hexadecimal string into decimal }
1823      var
1824        i    : integer;
1825        temp : longint;
1826      begin
1827        temp := 0;
1828        for i := 1 to length(hx) do
1829          temp := temp*16+htod(hx[i]);
1830        decimal := temp;
1831      end;

1832      procedure expand_window;
1833      begin
1834        dec(windmin,$0101);
1835        inc(windmax,$0101);
1836      end;

1837      procedure contract_window;
1838      begin
1839        inc(windmin,$0101);
1840        dec(windmax,$0101);
1841      end;

1842      procedure wrap_up( var n : integer; limit : integer);
1843      begin
1844        if n<limit then
1845          inc(n)
1846        else
1847          n := 1;
1848      end;

1849      procedure wrap_down( var n : integer; limit : integer);
1850      begin
1851        if n>1 then
1852          dec(n)
1853        else
1854          n := limit;
1855      end;

1856      procedure set_video_mode(mode : byte);
1857      begin
1858        with Regs do
1859          begin
1860            AH := 0; { Set mode }
1861            AL := mode;
1862            Intr(16,Regs);
1863          end;
1864      end;

1865      procedure blink_enable(flag : boolean);
1866      begin
```

```
1867        with Regs do
1868          begin
1869            AH := 16; {Colour palette interface}
1870            AL := 3;  {Blink option}
1871            if flag then
1872              BL := 1   {Blink on}
1873            else
1874              BL := 0;  {Blink off}
1875            Intr(16,Regs);
1876          end;
1877      end;

1878      procedure set_page(n : byte);
1879      { selects a page and makes it active }
1880      begin
1881        with Regs do
1882          begin
1883            AH := 5; { Set active page number }
1884            page_number := n;
1885            AL := n;
1886            Intr(16,Regs);
1887          end;
1888      end;

1889      procedure set_cursor(x,y : byte);
1890      begin
1891        with Regs do
1892          begin
1893            AH := 2; { Set cursor position }
1894            if relative_mode then
1895              begin
1896                DH := y+hi(WindMin)-1;
1897                DL := x+lo(WindMin)-1;
1898              end
1899            else
1900              begin
1901                DH := y;
1902                DL := x;
1903              end;
1904            BH := page_number;
1905            Intr(16,Regs);
1906          end;
1907      end;

1908      procedure get_char(x,y : byte; var character,attribute : byte);
1909      begin
1910        set_cursor(x,y);
1911        with Regs do
1912          begin
1913            AH := 8; { read character and attribute }
1914            BH := page_number;
1915            Intr(16,Regs);
1916            character := AL;
1917            attribute := AH;
1918          end;
1919      end;

1920      procedure put_char(character, attribute : byte);
1921      begin
1922        with Regs do
1923          begin
1924            if attribute=0 then
1925              AH := 10 { write character }
1926            else
1927              AH := 9; { write character and attribute }
1928            AL := character;
1929            BH := page_number;
1930            BL := attribute;
1931            CX := 1;
1932            Intr(16,Regs);
1933          end;
1934      end;

1935      procedure put_string(x,y : byte; s : string);
```

```
1936      var
1937        i : integer;
1938      begin
1939        if length(s)<>0 then
1940          for i := 1 to length(s) do
1941            begin
1942              set_cursor(x+i-1,y);
1943              put_char(ord(s[i]),textattr);
1944            end;
1945      end;

1946      procedure put_string2(x1,y1:byte;s1:string; x2,y2:byte;s2:string);
1947      begin
1948        put_string(x1,y1,s1);
1949        put_string(x2,y2,s2);
1950      end;

1951      procedure put_number(x,y : byte; n : float; f1,f2 : byte);
1952      var
1953       s : line_string;
1954      begin
1955        str(n:f1:f2,s);
1956        put_string(x,y,s);
1957      end;

1958      procedure put_string_and_number(x1,y1:byte;s1:string;
1959                                      x2,y2:byte; n:float; f1,f2 :byte);
1960      begin
1961        put_string(x1,y1,s1);
1962        put_number(x2,y2,n,f1,f2);
1963      end;

1964      procedure change_attribute(x,y,new_attribute : byte);
1965      var
1966        old_x,old_y,ch,attr : byte;
1967      begin
1968        old_x := wherex;
1969        old_y := wherey;
1970        get_char(x,y,ch,attr);
1971        put_char(ch,new_attribute);
1972        gotoxy(old_x,old_y);
1973      end;

1974      procedure highlight_line(xmin,xmax,y,new_attribute : byte);
1975      var
1976        x : integer;
1977      begin
1978        for x := xmin to xmax do change_attribute(x,y, new_attribute);
1979      end;

1980      procedure write_boarder_char(x,y,character_index,attribute : byte);
1981      var
1982        next_character, current_character, current_attribute : byte;
1983      const
1984        character_table : array[1..12] of byte =
1985            (218,191,217,192,196,179,201,187,188,200,205,186);
1986        merge_table : array[1..12,1..40] of byte =
1987            ((195,197,181,215,210,184,185,199,187,188,
1988              208,190,194,195,197,194,195,194,197,198,
1989              199,200,201,202,203,204,209,206,216,208,
1990              209,210,211,198,213,214,215,216,197,218),
1991             (180,180,181,182,183,184,185,182,187,188,
1992              189,181,191,197,197,194,197,194,197,198,
1993              215,200,201,202,203,204,209,206,207,208,
1994              209,210,208,198,213,214,215,216,180,194),
1995             (180,180,181,182,183,181,185,182,187,188,
1996              189,190,180,193,193,197,197,193,197,198,
1997              215,200,201,202,203,204,207,206,207,208,
1998              216,210,208,212,198,210,215,216,217,197),
1999             (195,197,181,215,210,181,185,199,187,188,
2000              208,190,197,192,193,197,195,193,197,198,
2001              199,200,201,202,203,204,207,206,207,208,
2002              216,210,211,212,198,214,215,216,193,195),
2003             (197,197,181,215,210,184,185,215,187,188,
```

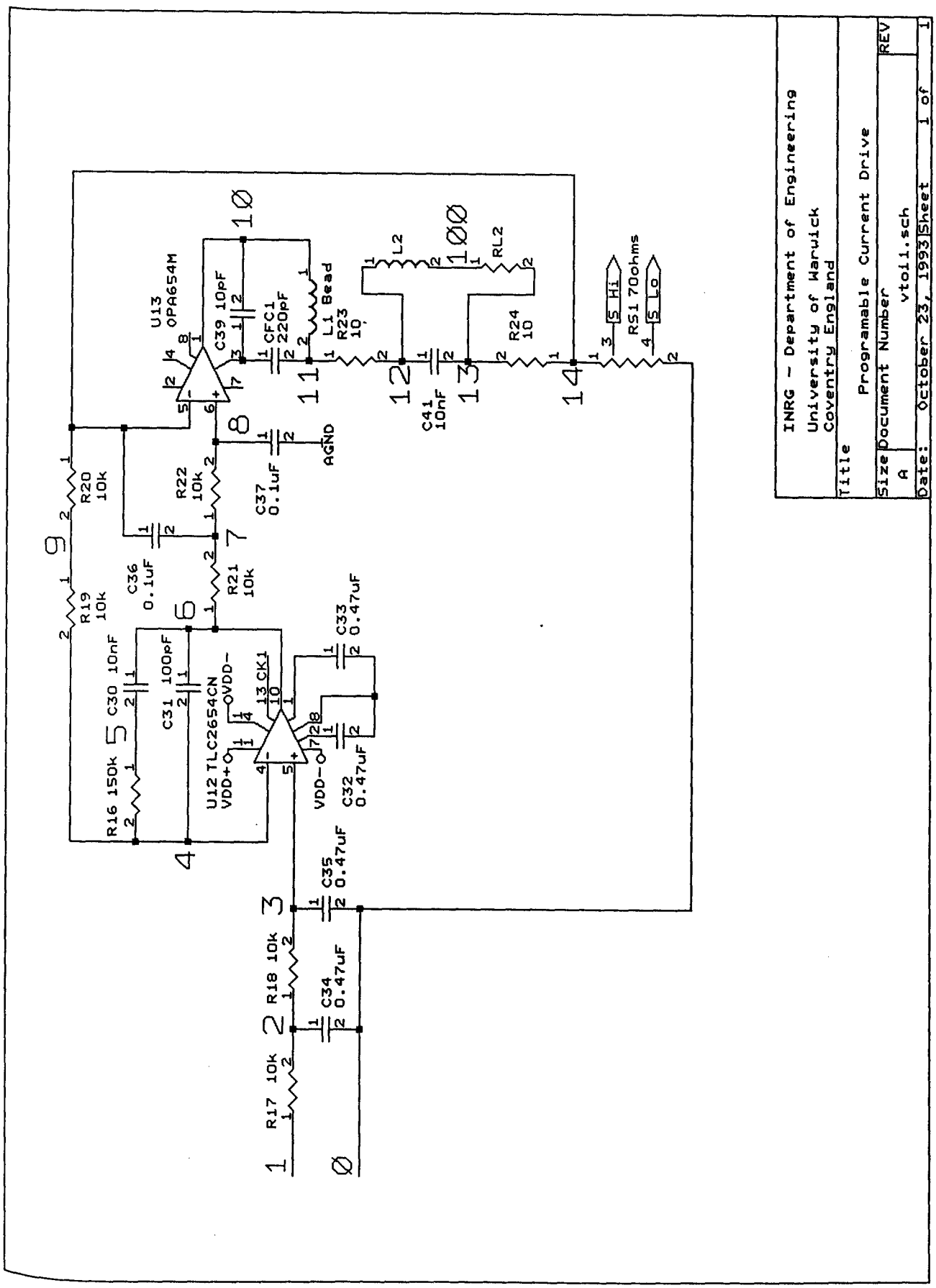
```
2004              208,190,194,193,193,194,197,196,197,198,
2005              215,200,201,202,203,204,205,206,207,208,
2006              209,210,208,212,213,210,215,216,193,194),
2007             (179,180,181,182,183,181,185,186,187,188,
2008              189,181,180,195,197,197,195,197,197,198,
2009              199,200,201,202,203,204,216,206,216,208,
2010              216,210,211,198,198,214,215,216,180,195),
2011             (201,201,203,204,201,203,206,204,203,206,
2012              204,203,201,201,201,201,201,201,201,201,
2013              204,204,201,206,203,204,203,206,203,204,
2014              203,201,204,201,201,201,204,203,201,201),
2015             (187,187,187,185,187,187,185,185,187,185,
2016              185,187,187,187,187,187,187,187,187,202,
2017              185,206,203,206,203,206,203,206,203,185,
2018              203,187,185,203,203,187,185,203,187,187),
2019             (188,188,188,185,185,188,185,185,185,188,
2020              188,188,188,188,188,188,188,188,188,202,
2021              185,202,206,202,206,206,202,206,202,188,
2022              202,185,188,202,203,185,185,202,188,188),
2023             (200,200,202,204,204,202,206,204,206,202,
2024              200,202,200,200,200,200,200,200,200,200,
2025              204,200,204,202,206,204,202,206,202,202,
2026              202,203,200,200,200,204,204,202,200,200),
2027             (216,216,216,206,203,209,206,206,203,202,
2028              202,207,209,207,207,209,216,205,216,216,
2029              206,202,203,202,203,206,205,206,207,202,
2030              209,203,202,207,209,203,206,216,207,209),
2031             (186,182,185,182,182,185,185,186,185,185,
2032              182,185,182,199,215,215,199,215,215,204,
2033              199,204,204,206,206,204,206,206,206,215,
2034              206,215,199,204,204,199,215,206,182,199)
2035              );
2036        begin
2037          get_char(x,y,current_character,current_attribute);
2038          case current_character of
2039            179..218 :
2040              next_character := merge_table[character_index,current_character-178];
2041            else
2042              next_character := character_table[character_index];
2043          end;
2044          put_char(next_character,attribute);
2045        end;

2046        procedure horizontal_line(xa,xb,y : integer ; width,attribute : byte);
2047        var
2048          i : integer;
2049        begin
2050          if xa<=xb then
2051            for i := xa to xb do write_boarder_char(i,y,5+6*(width-1),attribute);
2052        end;

2053        procedure vertical_line(x,ya,yb : integer ; width,attribute : byte);
2054        var
2055          i : integer;
2056        begin
2057          if ya<=yb then
2058            for i := ya to yb do write_boarder_char(x,i,6+6*(width-1),attribute);
2059        end;

2060        procedure left_line(x,ya,yb,width,attribute : byte);
2061        var
2062          offset : shortint;
2063        begin
2064          offset := 6*(width-1);
2065          write_boarder_char(x,ya,1+offset,attribute);
2066          write_boarder_char(x,yb,4+offset,attribute);
2067          vertical_line(x,ya+1,yb-1,width,attribute);
2068        end;

2069        procedure right_line(x,ya,yb,width,attribute : byte);
2070        var
2071          offset : shortint;
2072        begin
2073          offset := 6*(width-1);
```

```
2074        write_boarder_char(x,ya,2+offset,attribute);
2075        write_boarder_char(x,yb,3+offset,attribute);
2076        vertical_line(x,ya+1,yb-1,width,attribute);
2077      end;

2078      procedure top_line(xa,xb,y,width,attribute : byte);
2079      var
2080        offset : shortint;
2081      begin
2082        offset := 6*(width-1);
2083        write_boarder_char(xa,y,1+offset,attribute);
2084        write_boarder_char(xb,y,2+offset,attribute);
2085        horizontal_line(xa+1,xb-1,y,width,attribute);
2086      end;

2087      procedure bottom_line(xa,xb,y,width,attribute : byte);
2088      var
2089        offset : shortint;
2090      begin
2091        offset := 6*(width-1);
2092        write_boarder_char(xa,y,4+offset,attribute);
2093        write_boarder_char(xb,y,3+offset,attribute);
2094        horizontal_line(xa+1,xb-1,y,width,attribute);
2095      end;

2096      procedure update(var n : data_block_type; v : float; f1,f2 : byte);
2097      begin
2098        n.v := v;
2099        str(v:f1:f2,n.vs);
2100        n.vv := true;
2101      end;

2102      procedure draw_box(xmin,ymin,xmax,ymax,width,attribute : byte);
2103      begin
2104        top_line(xmin,xmax,ymin,width,attribute);
2105        bottom_line(xmin,xmax,ymax,width,attribute);
2106        left_line(xmin,ymin,ymax,width,attribute);
2107        right_line(xmax,ymin,ymax,width,attribute);
2108      end;

2109      procedure edit_line(xa,xb,y,next_attribute : byte; var s : line_string;
2110                          allowed_set : char_set);
2111      var
2112        i : integer;
2113        ts,ta : line_string;
2114        x,character,attribute : byte;
2115        done,first : boolean;

2116        procedure write_string;
2117          var
2118            i : integer;
2119          begin
2120            for i := x to xb do
2121              begin
2122                set_cursor(i,y);
2123                put_char(ord(ts[i-xa+1]),0);
2124              end;
2125            set_cursor(x,y)
2126          end;

2127          procedure move_right;
2128          begin
2129            if (x<xb) then x := x+1;
2130          end;

2131          procedure move_left;
2132          begin
2133            if x>xa then x := x-1;
2134          end;

2135          procedure move_end;
2136          begin
2137            x := xb;
2138            while (copy(ts,x-xa+1,1)=' ') and (x>=xa) do x := x-1;
```

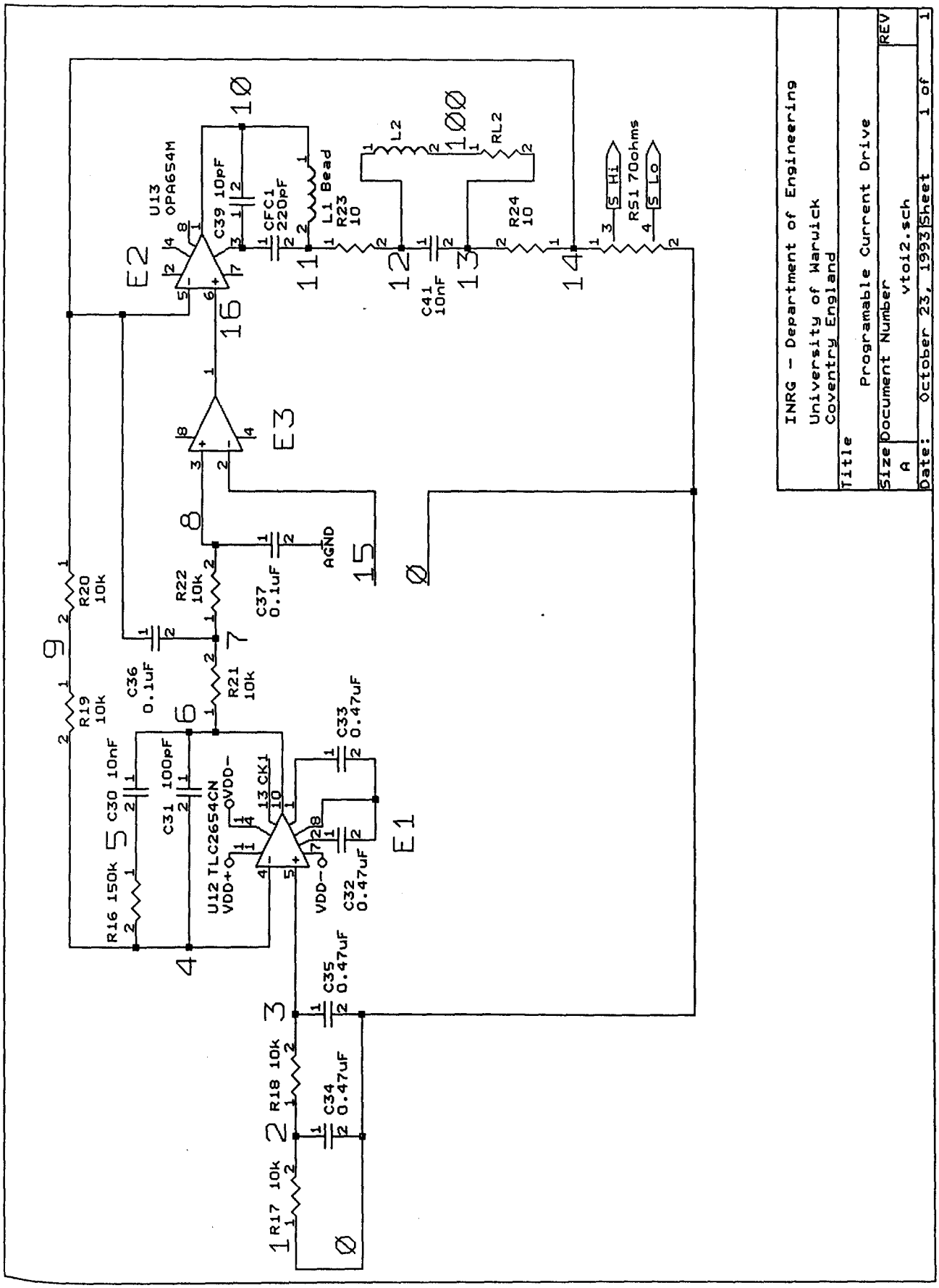
```
2139            move_right;
2140          end;

2141          procedure delete_char;
2142          begin
2143            ts := copy(ts,1,x-xa)+copy(ts,x-xa+2,length(ts)-(x-xa+1))+' ';
2144            write_string;
2145          end;

2146          procedure restore_attributes;
2147          var
2148            i : integer;
2149          begin
2150            for i := xa to xb do
2151              begin
2152                set_cursor(i,y);
2153                put_char(ord(ts[i-xa+1]),ord(ta[i-xa+1]));
2154              end;
2155            set_cursor(xa,y);
2156          end;

2157        begin
2158          ts := '';
2159          ta := '';
2160          for i := xa to xb do
2161            begin
2162              get_char(i,y,character,attribute);
2163              ts := ts+chr(character);
2164              ta := ta+chr(attribute);
2165              put_char(character,next_attribute);
2166            end;
2167          x := xa;
2168          done := false;
2169          first := true;
2170          repeat
2171            set_cursor(x,y);
2172            inkey;
2173            case last_key_code of
2174              Home       : x := xa;     { home   }
2175              LeftArrow  : move_left;   { left   }
2176              RightArrow : move_right;  { right  }
2177              Endkey     : move_end;    { right edge of string }
2178              Delkey     : delete_char; { delete }
2179              BS           : begin     { delete left }
2180                               move_left;
2181                               delete_char;
2182                             end;
2183              127          : begin     { move to far left and clear }
2184                               x := xa;
2185                               ts := spaces(xb-xa+1);
2186                               write_string;
2187                             end;
2188              32..255      : if chr(last_key_code) in allowed_set then
2189                               begin   { allowed character }
2190                                 if first and (ts<>spaces(xb-xa+1)) then
2191                                     ts := spaces(xb-xa+1);
2192                                 insert(chr(last_key_code),ts,x-xa+1);
2193                                 ts := copy(ts,1,length(ts)-1);
2194                                 write_string;
2195                                 move_right;
2196                               end;
2197              CR,ESC,256..512 : done := true;
2198            end;
2199            first := false;
2200          until done;
2201          s := ts;
2202          s := strip_spaces(ts);
2203          put_string(xa,y,copy(s+spaces(xb-xa+1),1,xb-xa+1));
2204          restore_attributes;
2205          put_string(xa,y,copy(s+spaces(xb-xa+1),1,xb-xa+1));
2206          set_cursor(xa,y);
2207        end;

2208        procedure get_full_string(xa,xb,y,next_attribute : byte; var s : line_string;
```

```
2209                                allowed_set : char_set);
2210      begin
2211        repeat
2212          edit_line(xa,xb,y,next_attribute, s, allowed_set);
2213        until pos(' ',s)=0;
2214      end;

2215      procedure edit_number(xa,xb,y,edit_colour, f1,f2 : byte; var d : data_block_type);
2216      var
2217        i,code : integer;
2218        done : boolean;
2219        ts : line_string;
2220      begin
2221        done := false;
2222        repeat
2223          edit_line(xa,xb,y,edit_colour,ts,number_set);
2224          ts := strip_spaces(ts);
2225          if ts='' then
2226            begin
2227              done := true;
2228              with d do
2229                begin
2230                  vv := false; v := 0; vs := ''
2231                end;
2232            end
2233          else
2234            begin
2235              val(ts,d.v,code);
2236              if code=0 then
2237                begin
2238                  str(d.v:f1:f2,d.vs);
2239                  if length(d.vs)<=(f1) then
2240                    begin
2241                      d.vv := true;
2242                      write(d.vs);
2243                      done := true;
2244                    end;
2245                end;
2246            end;
2247        until done;
2248      end;

2249      procedure get_number(xa,xb,y,edit_colour : byte; f1,f2 : byte;
2250                           var d : data_block_type);
2251      begin
2252        repeat
2253          edit_number(xa,xb,y,edit_colour,f1,f2,d);
2254        until d.vv or (length(d.vs)=0);
2255      end;

2256      procedure full_window;
2257      begin
2258        window(1,1,80,25);
2259      end;

2260      procedure clear_screen_area(xa,ya,xb,yb : byte);
2261      var
2262        wmin, wmax : word;
2263      begin
2264        wmin := windmin;
2265        wmax := windmax;
2266        window(xa,ya,xb,yb);
2267        clrscr;
2268        window(lo(wmin),hi(wmin),lo(wmax),hi(wmax));
2269      end;

2270      procedure ActiveWindow(Active: Boolean);
2271      begin
2272        if TopWindow <> nil then
2273          begin
2274            UnFrameWin;
2275            with TopWindow^ do
2276              if Active then
2277                FrameWin(Title^, DoubleFrame, TitleAttr, FrameAttr)
```

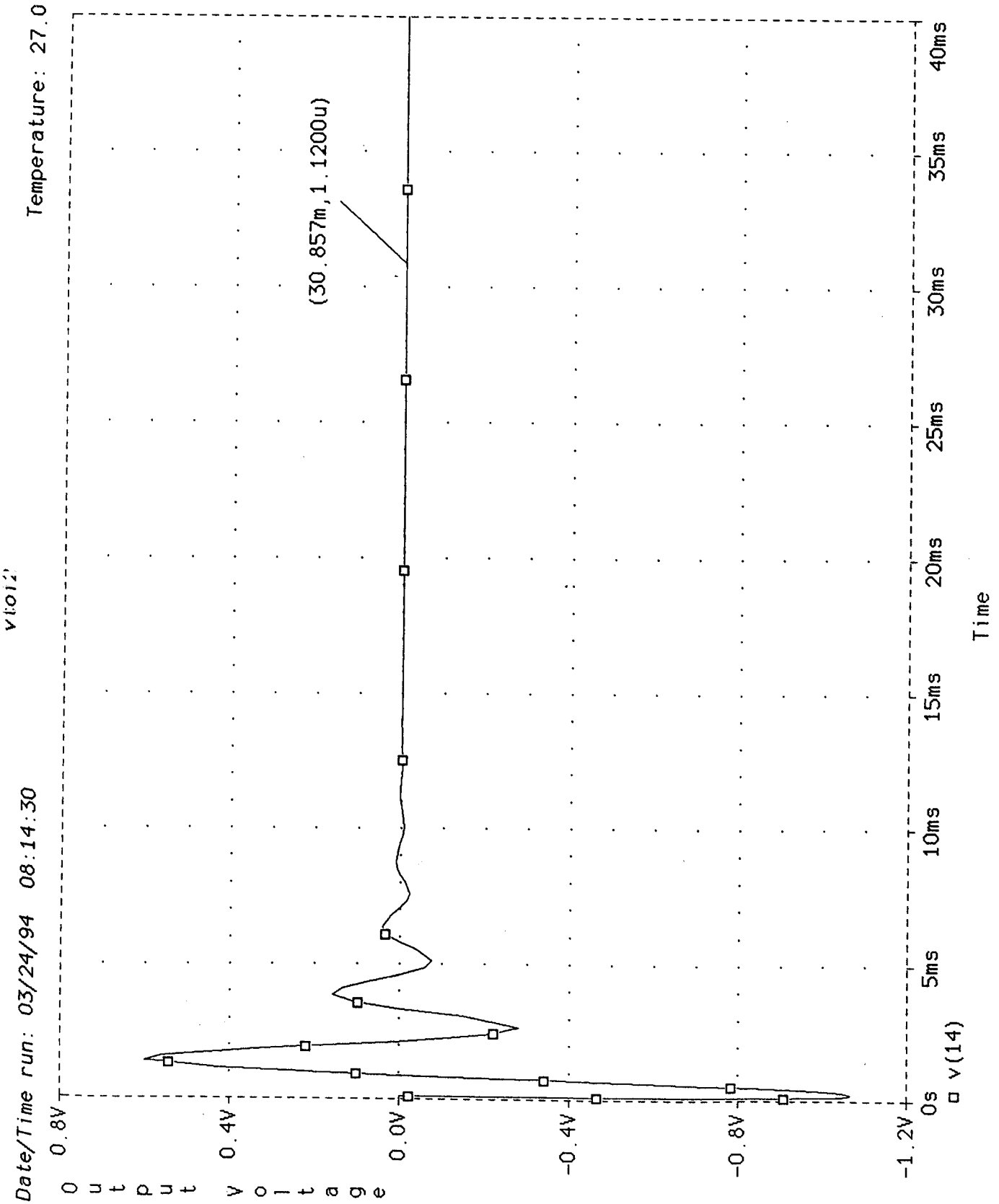
```
2278              else
2279                FrameWin(Title^, SingleFrame, FrameAttr, FrameAttr);
2280            end;
2281      end;

2282      procedure OpenWindow(X1, Y1, X2, Y2: Byte; T: TitleStr;
2283                           TAttr, FAttr: Byte);
2284      var
2285        W: WinRecPtr;
2286      begin
2287        ActiveWindow(False);
2288        New(W);
2289        with W^ do
2290          begin
2291            Next := TopWindow;
2292            SaveWin(State);
2293            GetMem(Title, Length(T) + 1);
2294            Title^ := T;
2295            TitleAttr := TAttr;
2296            FrameAttr := FAttr;
2297            Window(X1, Y1, X2, Y2);
2298            GetMem(Buffer, WinSize);
2299            ReadWin(Buffer^);
2300            FrameWin(T, DoubleFrame, TAttr, FAttr);
2301          end;
2302        TopWindow := W;
2303        Inc(WindowCount);
2304        TextAttr := TAttr;
2305        ClrSCr;
2306      end;

2307      procedure CloseWindow;
2308      var
2309        W: WinRecPtr;
2310      begin
2311        if TopWindow <> nil then
2312          begin
2313            W := TopWindow;
2314            with W^ do
2315              begin
2316                UnFrameWin;
2317                WriteWin(Buffer^);
2318                FreeMem(Buffer, WinSize);
2319                FreeMem(Title, Length(Title^) + 1);
2320                RestoreWin(State);
2321                TopWindow := Next;
2322              end;
2323            Dispose(W);
2324            ActiveWindow(True);
2325            Dec(WindowCount);
2326        end;
2327      end;

2328      { initialisation }
2329      begin
2330        set_page(0);
2331        relative_mode := true;
2332        last_key_code := 0;
2333        TopWindow     := nil;
2334        WindowCount   := 0;
2335      end.
```

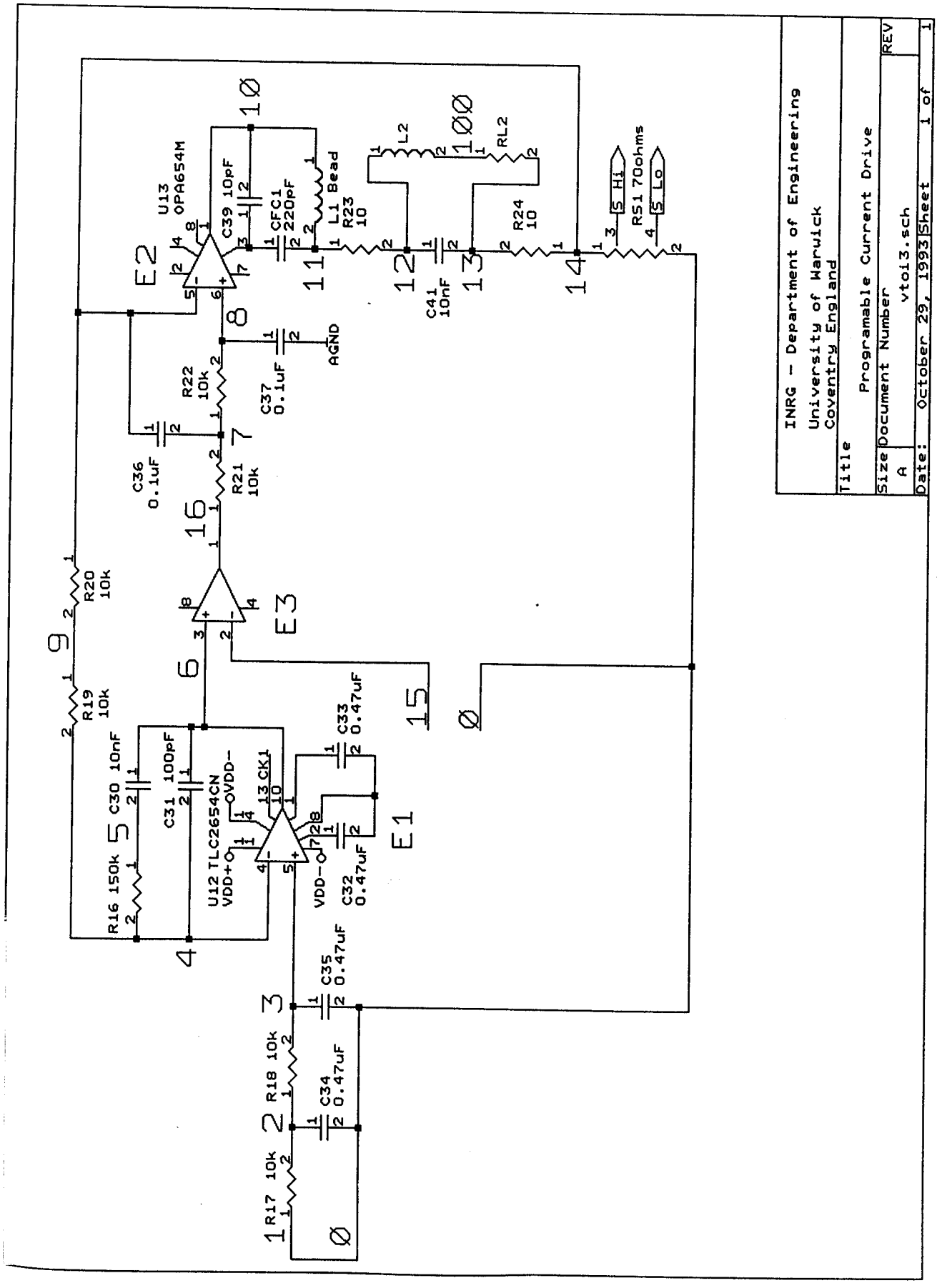

INRG - Department of Engineering
University of Warwick
Coventry England
Title
Programable Current Drive
Size
A
Document Number
vtoi1.sch
REV
Date: October 23, 1993
Sheet 1 of 1
U13 OPA654M
C39 10pF
CFC1 220pF
L1 Bead
R23 10
L2
RL2
R24 10
C41 10nF
5 Hi
RS1 70ohms
5 Lo
AGND
C37 0.1uF
R22 10k
R20 10k
C36 0.1uF
R21 10k
R19 10k
C30 10nF
C31 100pF
R16 150k
U12 TLC2654CN
VDD+
VDD-
CK1
C33 0.47uF
C32 0.47uF
C35 0.47uF
C34 0.47uF
R18 10k
R17 10k

```
vtoil
*
* Output Stage of Precision Current Drive based DR1DRV1.sch rev Q2

.OPT ACCT LIST NODE OPTS NOPAGE RELTOL=.001

* Small-signal transfer function calculation assuming
* VIN is the input and V(14), the voltage at node 14, is the output.
.TF V(14) VIN

* AC analysis.  The real and imaginary response of the circuit
* is calculated as the inputs are swept from 0.1 hertz to 10kHz
* The only AC input this circuit has is VIN.  Linear analysis.
.AC DEC 20 0.1HZ 10000HZ

* Calculate transient response in 20 microsecond steps for 0.2s
.TRAN 20U 0.2

* The following statements describe the circuit to PSpice.
* VIN is the input. It has an amplitude during AC analysis
* of 1 volt, and changes from 0 to 1 volt for transient analysis
VIN 1 0 AC 1 PULSE(0 1 0.1M 0 0 1 1)

R17 1 2 10K
R18 2 3 10K
C34 2 0 0.47UF
C35 3 0 0.47UF
C31 6 4 100PF
R16 5 4 150K
C30 6 5 0.01UF
R19 9 4 10K
R20 14 9 10K
R21 6 7 10K
R22 7 8 10K
C36 14 7 0.1UF
C37 8 0 0.1UF
L1 10 11 10UH
R23 11 12 10
L2 12 100 800UH
RL2 100 13 40
C41 12 13 0.01UF
R24 13 14 10
RS1 14 0 70

E1 6 0 3 4 5MEG
E2 10 0 8 14 12000

.PROBE
.END
```

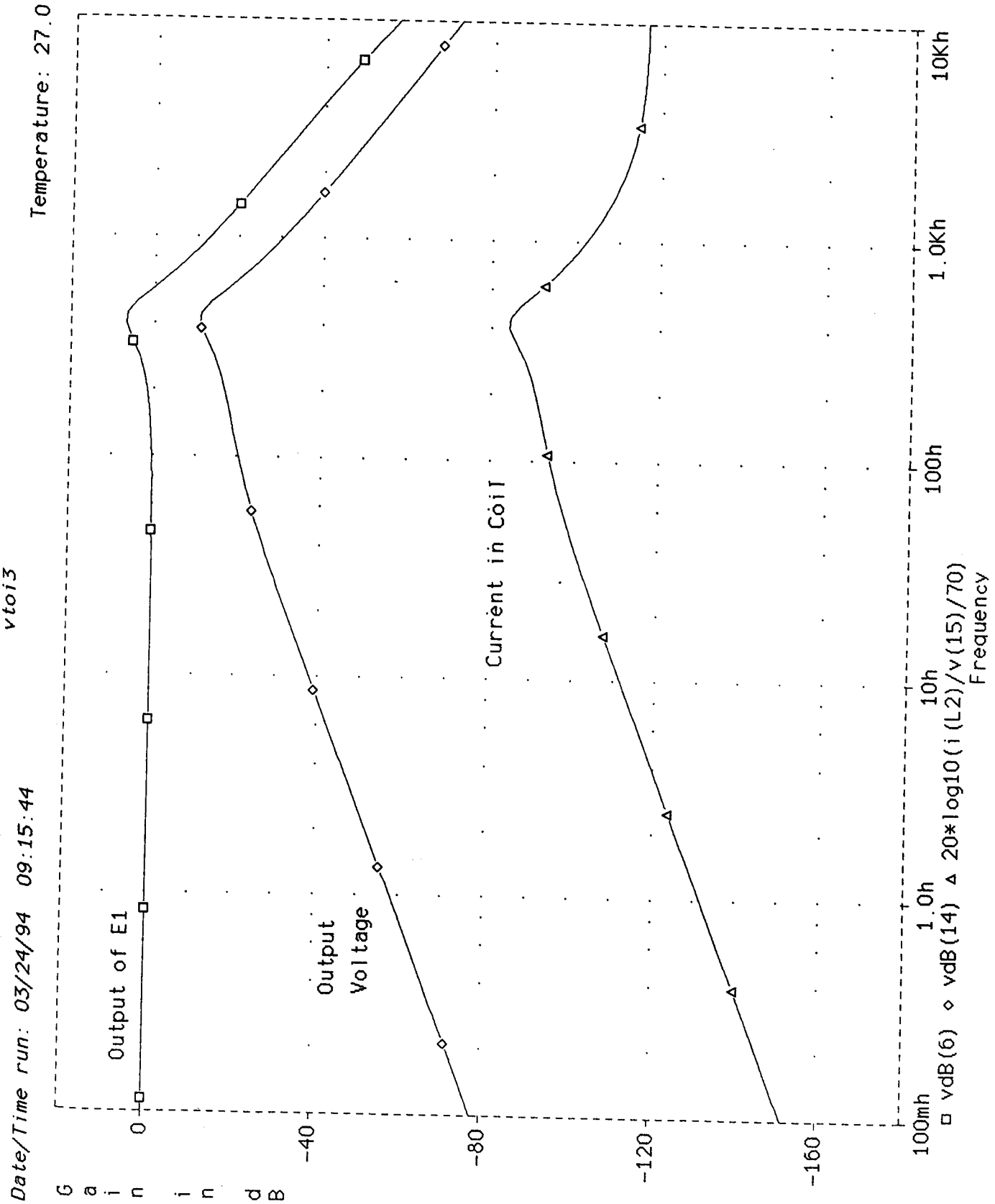

INRG - Department of Engineering
University of Warwick
Coventry England
Title
Programable Current Drive
Size
A
Document Number
vtoi2.sch
REV
Date: October 23, 1993 Sheet 1 of 1
U13 OPA654M
C39 10pF
CFC1 220pF
L1 Bead
R23 10
L2
RL2
R24 10
C41 10nF
S Hi
S Lo
RS1 70ohms
C37 0.1uF
AGND
R22 10k
R20 10k
R19 10k
C36 0.1uF
R21 10k
C30 10nF
C31 100pF
U12 TLC2654CN
VDD+
VDD-
CK1
C33 0.47uF
C32 0.47uF
C35 0.47uF
C34 0.47uF
R18 10k
R17 10k
R16 150k
E1
E2
E3

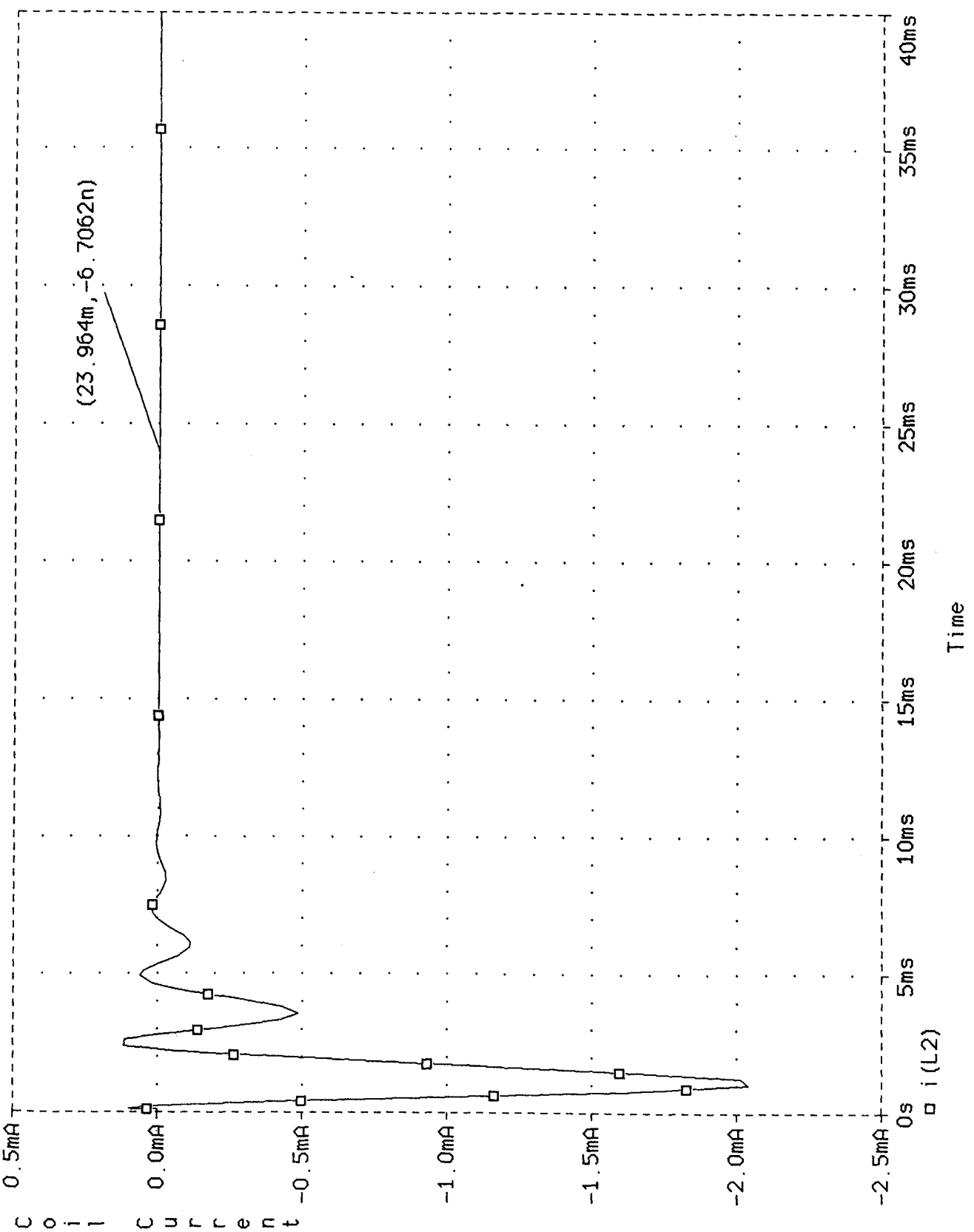
```
vtoi2
*
* Output Stage of Precision Current Drive based on DR1DRV1.sch rev Q2
* Servo loop has additional 'noise' source

.OPT ACCT LIST NODE OPTS NOPAGE RELTOL=.001

* Small-signal transfer function calculation assuming
* VIN is the input and V(14), the voltage at node 14, is the output.
.TF V(14) VIN

* AC analysis.  The real and imaginary response of the circuit
* is calculated as the inputs are swept from 0.1 hertz to 10kHz
* Only AC input this circuit has is VIN.  This is a linear analysis.
.AC DEC 20 0.1HZ 10000HZ

* Calculate transient in 20 microsecond steps for 0.1 seconds
.TRAN 20U 0.1

* The following statements describe the circuit to PSpice.
* VIN is the input. It has an amplitude during AC analysis
* of 1 volt, and changes from 0 to 1 volt for transient analysis
* VIN 1 0 AC 1 PULSE(0 1 0.1M 0 0 1 1)
* VD 15 0 0
* voltage disturbance within loop
VD 15 0 AC 1 PULSE(0 1 0.1M 0 0 1 1)
VIN 1 0 0
R17 1 2 10K
R18 2 3 10K
C34 2 0 0.47UF
C35 3 0 0.47UF
C31 6 4 100PF
R16 5 4 150K
C30 6 5 0.01UF
R19 9 4 10K
R20 14 9 10K
R21 6 7 10K
R22 7 8 10K
C36 14 7 0.1UF
C37 8 0 0.1UF
L1 10 11 10UH
R23 11 12 10
L2 12 100 800UH
RL2 100 13 40
C41 12 13 0.01UF
R24 13 14 10
RS1 14 0 70
RD1 15 0 10M
RD2 16 0 10M

E1 6 0 3 4 5MEG
E2 10 0 16 14 12000
* E3 allows inclusion of disturbance within servo loop at input to E2
E3 16 0 8 15 1

.PROBE
.END
```


Temperature: 27.0
(30.857m,1.1200u)
Date/Time run: 03/24/94 08:14:30
0.8V
0.4V
0.0V
-0.4V
-0.8V
-1.2V
Output voltage
0s
5ms
10ms
15ms
20ms
25ms
30ms
35ms
40ms
Time
v(14)


INRG - Department of Engineering
University of Warwick
Coventry England
Title
Programable Current Drive
Size
A
Document Number
vtoi3.sch
REV
Date: October 29, 1993
Sheet 1 of 1
U13 OPA654M
E2
C39 10pF
CFC1 220pF
L1 Bead
R23 10
L2
RL2
R24 10
C41 10nF
S Hi
S Lo
RS1 70ohms
C37 0.1uF
AGND
R22 10k
C36 0.1uF
R21 10k
E3
R20 10k
R19 10k
R16 150k
C30 10nF
C31 100pF
U12 TLC2654CN
VDD+
VDD-
CK1
C33 0.47uF
C32 0.47uF
E1
C35 0.47uF
R18 10k
C34 0.47uF
R17 10k

```
vtoi3
*
* Output Stage of Precision Current Drive based on DR1DRV1.sch rev Q2
* 'Noise' may be added to output of E1

.OPT ACCT LIST NODE OPTS NOPAGE RELTOL=.001

* Small-signal transfer function calculation assuming
* VIN is the input and V(14), the voltage at node 14, is the output.
.TF V(14) VIN

* AC analysis. The real and imaginary response of the circuit
* is calculated as the inputs are swept from 0.1 hertz to 10kHz
* Only AC input is VIN.  This is a linear analysis.
.AC DEC 20 0.1HZ 10000HZ

* Calculate transient in 20 microsecond steps for 0.2 seconds
.TRAN 20U 0.2

* The following statements describe the circuit to PSpice.
* VIN is the input. It has an amplitude during AC analysis
* of 1 volt, and changes from 0 to 1 volt for transient analysis
* VIN 1 0 AC 1 PULSE(0 1 0.1M 0 0 1 1)
* VD 15 0 0
* voltage disturbance within loop at output of E1
VIN 1 0 0
VD 15 0 AC 1 PULSE(0 1 0.1M 0 0 1 1)

R17 1 2 10K
R18 2 3 10K
C34 2 0 0.47UF
C35 3 0 0.47UF
C31 6 4 100PF
R16 5 4 150K
C30 6 5 0.01UF
R19 9 4 10K
R20 14 9 10K
R21 16 7 10K
R22 7 8 10K
C36 14 7 0.1UF
C37 8 0 0.1UF
L1 10 11 10UH
R23 11 12 10
L2 12 100 800UH
RL2 100 13 40
C41 12 13 0.01UF
R24 13 14 10
RS1 14 0 70
RD1 15 0 10M

E1 6 0 3 4 5MEG
E2 10 0 8 14 12000
* E3 allows disturbance within servo loop at output of E1
E3 16 0 6 15 1

.PROBE
.END
```


Date/Time run: 03/24/94 09:15:44
vtoi3
Temperature: 27.0
Output of E1
Output Voltage
Current in Coil
Gain in dB
0
-40
-80
-120
-160
100mh
1.0h
10h
100h
1.0Kh
10Kh
Frequency
vdB(6)
vdB(14)
20*log10(i(L2)/v(15)/70)


Temperature: 27.0
vtoi3
Date/Time run: 03/24/94 09:15:44
(23.964m,-6.7062n)
Coil Current
0.5mA
0.0mA
-0.5mA
-1.0mA
-1.5mA
-2.0mA
-2.5mA
0s
5ms
10ms
15ms
20ms
25ms
30ms
35ms
40ms
Time
i(L2)

```
scvtoi.cir
* Voltage-to-current converter using a switched capacitor

.OPT ACCT LIST NODE OPTS NOPAGE RELTOL=.001

* Calculate transient response in 50 microsecond steps for 0.06s
.TRAN 50U 0.06

* model almost ideal switch
.model sno vswitch(RON=100 ROFF=1E+10 VON=0.51 VOFF=0.49)
VIN 100 0 PULSE(1.5 3.5 0.01 0 0 1 2) ; bias => -1 to +1
VBIAS 5 0 2.5V    ; typically half range
VCL1 20 0 PULSE(0 1 100U 0 0 300U 1M) ; 1kHz square wave
VCL2 21 0 PULSE(0 1 600U 0 0 300U 1M) ; non-overlapping
* VCL1 20 0 PULSE(0 1 100U 0 0 200U 1M) ; 1kHz square wave
* VCL2 21 0 PULSE(0 1 400U 0 0 500U 1M) ; non-overlapping
.IC V(4,5)=-1 V(6,7)=-1 V(8,9)=-1 v(2,3)=2.7 ;initial bias conditions
E1 3 0 1 2 1E7    ; amplifier
RIN 100 1 22k     ; input filter
.PARAM CF=1
CIN 1 0 {CF}      ; filter capacitor
.STEP PARAM CF LIST 0.1UF 0.22UF 0.33uF 0.47UF
R1 2 4 1k         ; Integration time constant
C1 3 2 1uF
C2 4 5 1uF        ; hold capacitor
C3 6 7 1uF        ; sample capacitor
C4 8 9 1uF        ; feedback filter capacitor
R2 10 8 100       ; feedback filter resistor
R3 11 9 100       ; feedback filter resistor
L1 3 12 500UH     ; coil
RE1 12 10 5       ; exagerated end effect of Rref
RE2 11 0 5        ; exagerated end effect of Rref
RREF 10 11 50     ; 50 ohm reference resistor
SW1 6 4 20 0 sno ; normally open switch
SW2 7 5 20 0 sno ; normally open switch
SW3 8 6 21 0 sno ; normally open switch
SW4 9 7 21 0 sno ; normally open switch
.PROBE
.END
```

ANALOG DEVICES

High Accuracy, 22-Bit Integrating A/D Converter

AD1175K

FEATURES

High Resolution: 22 Bits
Wide Dynamic Range: 133 dB
Low Nonlinearity:
- **Integral: ±0.5 ppm max**
- **Differential: ±0.5 LSB max**

High Stability:
- **Gain: ±1 ppm/°C max**
- **Zero: ±0.5 mV/°C max**
- **INL: ±0.01 ppm/°C**
- **DNL: ±0.0025 ppm/°C**

High Throughput Rate: 20 Conversions/Second
Microprocessor Compatible Interface
Compact Modular Package

APPLICATIONS

Data Acquisition Systems
Scientific Instruments
Medical Instruments
Weighing Systems
Automatic Test Equipment
Test and Measurement Equipment

GENERAL DESCRIPTION

The AD1175 is a very high resolution integrating A/D converter intended for applications that require the highest possible accuracy without sacrificing conversion speed, board space or modest pricing. This converter provides the performance of large benchtop or rack mount instruments in a compact, modular package.

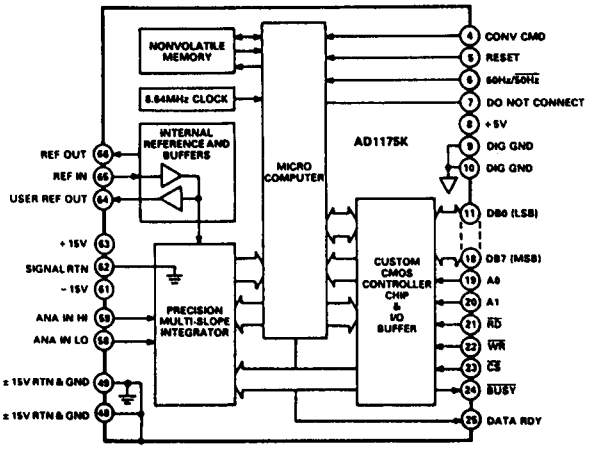
The AD1175 utilizes an auto-zeroed, multislope, integrating principle that features 22-bit resolution with extremely low nonlinearity (Integral: ±0.5 ppm max and Differential: ±0.5 LSB max). Temperature stability is specified at ±0.5 ppm/°C maximum for gain (exclusive of reference), ±0.5 μV/°C maximum for zero, ±0.01 ppm/°C for integral nonlinearity, and ±0.0025 ppm/°C for differential nonlinearity.

The integration time is user selectable for maximum, line frequency noise rejection at either 60 Hz or 50 Hz. The conversion rate is 20 or 16 per second respectively, which is many times faster than benchtop instruments of similar performance.

The nominal full-scale input range is ±5 V; however, rated accuracy is specified for inputs up to 10% over nominal, yielding a total dynamic range of greater than 4.6 million to 1. The analog input is a high impedance, high CMRR, true differential input pair. The input low operates within ±100 mV of analog ground and is used to sense signal low (at the source) to minimize ground loop problems.

The output of the AD1175 consists of four addressable 8-bit bytes (STATUS and 3 DATA) presented at an 8-bit tri-stated port with standard chip select.

AD1175K FUNCTIONAL BLOCK DIAGRAM



Several modes of operation are available and allow writing to one of several addressable locations to program gain and offset, or to initiate a conversion.

The AD1175 requires no external components and operates from ±15 V dc and +5 V dc power. All digital inputs and outputs are LSTTL compatible. The 3.7" × 5.2" × 0.53" metal case package provides excellent electrostatic and electromagnetic shielding.

PRODUCT HIGHLIGHTS

1. The unparalleled dynamic range, accuracy, linearity and stability of the AD1175 represent a breakthrough for an A/D converter offering small size and modest cost. Only large, expensive benchtop meters offer similar performance.
2. The AD1175 converts approximately ten times as fast as digital meters with like performance.
3. The microprocessor interface of the AD1175 provides for straightforward operation, but with the features required for optimum system performance. Simple commands control offset adjust, gain adjust, external offset null and initiate conversions. The output bytes indicate input polarity, off-scale condition and a variety of additional status information.
4. The AD1175 is a complete A/D converter including a precision internal reference, clock and integration capacitor. Offset and coarse gain adjust are bus controlled, while user accessible trim potentiometers allow fine gain adjust and ±full scale balance adjust.
5. Conversions may be made using either the offset and coarse gain settings stored in internal nonvolatile memory, or new settings made via the bus. The nonvolatile memory may be updated on command with the new settings.

SPECIFICATIONS (typical @ +25°C, V_S = ±15 V, V_D = +5 V unless otherwise specified)

Model	AD1175K
RESOLUTION	22-Bits +10% Overrange (4,600,000 Counts) min
DYNAMIC RANGE	133 dB
ACCURACY	
Integral Nonlinearity[1]	±0.5 ppm FSR[2], max
Differential Nonlinearity (@ 22 Bits)	±0.5 LSB, max
Total Noise (Ref to Input, 95% Confidence)	5 μV p-p max
STABILITY	
Gain TC (Excluding Reference)	±1 ppm RDG/°C, max
Zero TC	±0.5 μV/°C, max
Integral Nonlinearity TC	±0.01 ppm FSR[2]/°C
Differential Nonlinearity TC	±0.0025 ppm FSR[2]/°C
POWER SUPPLY REJECTION RATIO (±15 V)	±5 ppm FSR[2]/V
WARMUP TIME	
Relative Accuracy (for Rated Performance)	15 Minutes
Full Rated Performance	45 Minutes
REFERENCE	
External Reference In	
For Rated Performance	+6.95 V ±2%[3]
Maximum Input (Operating Only)	+9.6 V
Reference Output	
Voltage	+6.95 ±2%
Output Resistance	250 Ω
Temperature Coefficient	±0.4 ppm/°C (±0.8 ppm/°C, max)
Drift with Time[4]	
1st 15 Days Operating	±1 ppm/Day
After 15 Days Operation	±25 ppm $\sqrt{1000}$ hrs., max
Noise, 0.01 Hz to 10 Hz (95% Confidence)	1 ppm p-p, max
User Reference Output	
Gain (Referred to Reference In)	1.000 to 1.012[5]
Current	±2 mA, max
Stability: Temperature Coefficient	±1 μV/°C, max
THROUGHPUT RATE[6]	
@ Integrate Time of 1/30 sec (60 Hz)	20 conversions/sec
@ Integrate Time of 1/25 sec (50 Hz)	16 conversions/sec
ANALOG INPUT CHARACTERISTICS	
Voltage Range[7,8]	±5 V Bipolar
Max V_{IN}H (at Input Hi, Without Damage)	±12 V
Max V_{IN}L (at Input Lo, Without Damage)	±3 V
Max V_{IN}LR (Input Lo, for Rated Performance)	±100 mV
Input Resistance (Input Hi, or Input Lo)	1000 MΩ
Input Bias Current, Input Hi or Input Lo (+10°C to +50°C)	±10 nA, typ, ±40 nA max
Input Bandwidth[9]	
Small Signal	2.0 MHz
Large Signal	150 kHz
CMRR at dc to 60 Hz	80 dB, min
ADJUSTMENTS	
Offset (Programmable)	
Range	±75 mV
Resolution	1 LSB Steps
Gain-Coarse (Programmable)[8]	
Range	<4.7V to >5.6 V
Resolution	0.009% Steps
Gain-Fine Range[5,8]	±0.006% FS
Gain-Balance (±Full Scale) Range[5]	±0.005% FS
DIGITAL LEVELS	
Inputs	
Low	0.8 V max
High	2.0 V min
Outputs	
Low (@ 4 mA)	0.45 V max
High (@ 100 μA)	2.4 V min
POWER REQUIREMENTS	
Supply Voltages (for Rated Accuracy)	
$\pm V_S$	±15 V (±0.3 V each)
$+V_D$	+5 V (−0.2 V to +0.4 V)
Supply Current Drain	
@ ±15 V	
After Warm-Up	+55 mA, −70 mA
During Warm-Up	150 mA
@ +5 V	175 mA
ENVIRONMENTAL	
Rated Performance	10°C to +50°C, 70% RH
Operating	0 to +70°C
Storage	−25°C to +70°C
MECHANICAL	
Size	3.7" × 5.2" × 0.53" max
Shielding	Electrostatic, 6 Sides Electromagnetic, 5 Sides
Weight	170 grams

NOTES

[1]Integral Nonlinearity is specified over the entire input span (NOMINAL FULL-SCALE + 10% Overrange). It is specified using the "End Point" definition, where the error is measured after removing the offset error and the gain errors at plus and minus full scale.

[2]FSR means Full Scale Range which = 10 volts.

[3]Single ended, ground referred.

[4]Average trend line.

[5]Adjustment is performed via user accessible 10-turn trim potentiometer.

[6]Integration Time is selectable to either 1/30 sec for 60 Hz rejection, or 1/25 sec for 50 Hz rejection.

[7]The Nominal Analog Input Voltage Range is ±5V, but the AD1175 may be calibrated for input voltages from ±4.7 V to ±5.6 V and maintain specified accuracy over the entire range, including a 10% on-scale overrange. Therefore, input voltages of up to ±6.16 V will be accurately converted when calibrated for ±5.6 V Nominal input.

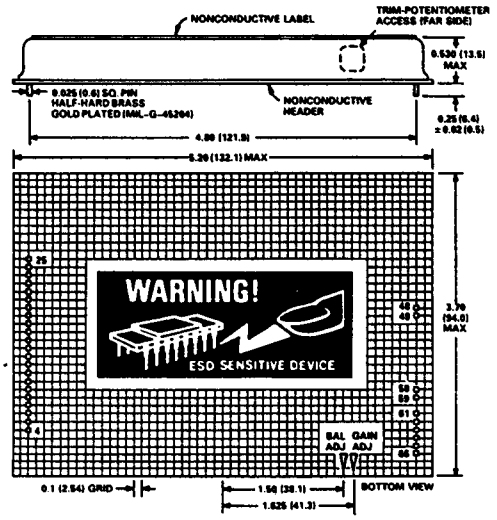
[8]Converter section GAIN is digitally adjustable, via the data bus, in steps of 0.009% from <4.7 to >5.6V FS. A user accessible 10-turn trim potentiometer is also provided for fine GAIN adjust (±0.006% range). All units are factory calibrated for ±5 V Nominal Full Scale to within ±50 μV.

[9]Input Bandwidth specifications are for true integration without clipping.

Specifications subject to change without notice.

OUTLINE DIMENSIONS

Dimensions shown in inches and (mm).



NOTE: SEE PAGE 3-161 FOR RECOMMENDED SOCKET.
SEE PAGE 3-166 FOR EVALUATION BOARD.

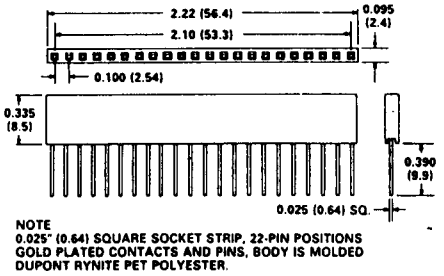
ASSEMBLY INSTRUCTIONS

CAUTION: This module is not an embedded assembly and is not hermetically sealed. Do not subject to a solvent or water-wash process that would allow direct contact with free liquids or vapors. Entrapment of contaminants may occur, causing performance degradation and permanent damage. Install after any clean/wash process and then only spot clean by hand.

PIN DESCRIPTIONS

PIN	SIGNAL	DESCRIPTION
4	CONV CMD	External Convert Command
5	RESET	Reset Internal Microcomputer Following Power-Up
6	60Hz/$\overline{50Hz}$	When Set Low, Integration Time is 1/25 sec When Set High, Integration Time is 1/30 sec
7	DO NOT CONNECT	Used Only for Factory Test
8	+5V	Digital Power Supply
9, 10	DIG GND	Digital Ground (Both Pins are Tied Together Internally)
11-18	DB0-DB7	Bidirectional Data Bus (LSB-MSB)
19	A0	Address, Bit Zero
20	A1	Address, Bit One
21	$\overline{RD}$	READ
22	$\overline{WR}$	WRITE
23	$\overline{CS}$	CHIP SELECT
24	$\overline{BUSY}$	BUSY, Responding to a Bus Command
25	DATA RDY	DATA READY
48, 49	±15V RTN & GND	Analog Power Ground and Case (Tied Together Internally)
58	ANA IN LO	Analog Input, Low
59	ANA IN HI	Analog Input, High
61	−15V	Negative Analog Power Supply
62	SIGNAL RTN	Signal Return (Non-Current Carrying Ground)
63	+15V	Positive Analog Power Supply
64	USER REF OUT	Buffered Output of Reference at REF IN
65	REF IN	Reference Input, Normally Connected to REF OUT
66	REF OUT	Internal +6.95V Reference Output, Unbuffered

SAMTEC Part Number SSQ-122-03-G-S (2 Each Required Per AD1175)
Available direct from the manufacturer or through distributors.



ARCHITECTURAL OVERVIEW

The AD1175 is a complete, precision analog-to-digital converter. It consists of three major elements: a linearized, auto-zeroed integrator, a single-chip microcomputer, and a custom CMOS controller/bus interface chip. (See Figure 1 AD1175 Functional Block Diagram.)

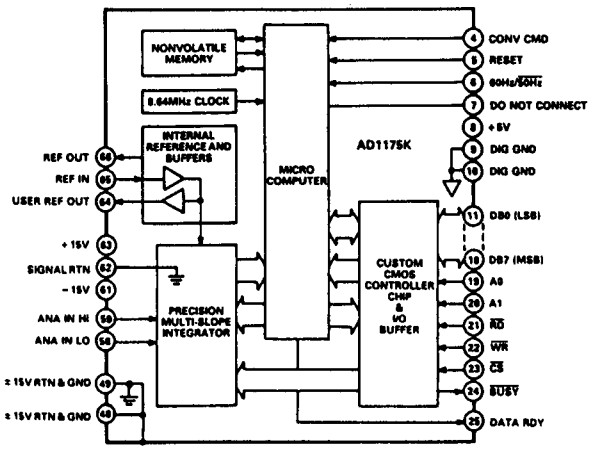


Figure 1. AD1175 Functional Block Diagram

The conversion process is similar to the classic dual-slope technique, where the input signal is integrated during a whole number of line cycles (for line noise rejection) and then a digital measurement is made of the time required for a known reference voltage to drive the integrator output back to zero (i.e., to zero charge). Since the process begins with zero charge in the integrator, and also ends there, we can express this function as follows:

CHARGE IN = CHARGE OUT

$$\text{WHERE CHARGE} = \int_0^t I\,dt = \frac{1}{R}\int_0^t V\,dt$$

$$\text{OR} \ldots \frac{1}{R_{INT}}\int_0^{T_{SIG}} V_{SIG}\,dt = \frac{1}{R_{INT}}\int_0^{T_{REF}} V_{REF}\,dt$$

$$\text{OR} \ldots \int_0^{T_{SIG}} V_{SIG}\,dt = V_{REF} \times T_{REF} \text{ (SINCE } V_{REF} = \text{CONSTANT)}$$

$$\text{OR} \ldots \frac{\int_0^{T_{SIG}} V_{SIG}\,dt}{T_{SIG}} = \text{AVG. } [V_{SIG}] = \frac{V_{REF} \times T_{REF}}{T_{SIG}}$$

$$\text{HENCE} \ldots \frac{\text{AVG. } [V_{SIG}]}{V_{REF}} = \frac{T_{REF}}{T_{SIG}} \begin{cases} \text{WHERE } T_{REF} \text{ IS MEASURED AND} \\ V_{REF} \text{ \& } T_{SIG} \text{ ARE CONSTANT} \end{cases}$$

Principle of Dual-Slope Conversion

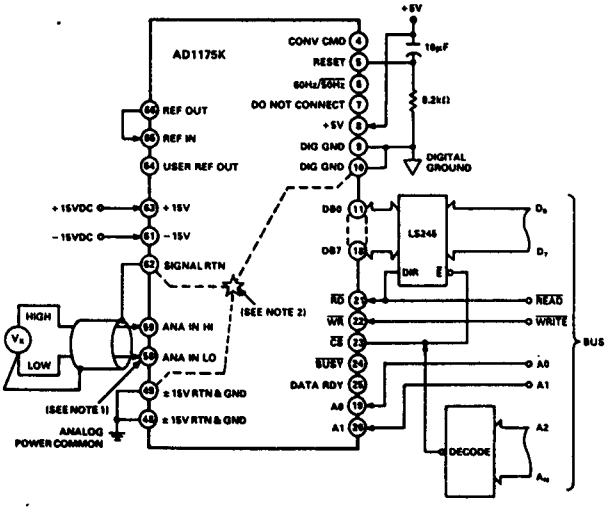
Therefore, the ratio of the signal measured (its average value) to the reference voltage, is equal to the ratio of the *measured time* (to force the integrator back to zero charge) to the signal integration time (which is held constant).

The AD1175 repeats the above sequence ten times during the first 33-1/3 milliseconds of each conversion for a 60 Hz integrate selection (40 milliseconds for a 50 Hz integrate selection). The 10 individual readings together with the result of a final, slow (about 6 ms) vernier reference integration are summed. The numeric result is then placed in the addressable output latches and DATA is indicated as AVAILABLE. During the next ten milliseconds, the integrator is reset and AUTO-ZERO nulls out offset errors in preparation for the next conversion.

The device status is indicated by the addressable STATUS byte (busy, converting, data available, etc.). DATA READY and $\overline{\text{BUSY}}$ are also indicated by logic levels at Pins 25 and 24, respectively.

SIGNAL INPUT CONNECTIONS

The ANA IN HI and ANA IN LO pins comprise a true, high-impedance, high CMRR, differential input pair. ANA IN LO must be within ±100 mV of SIGNAL RTN (Pin 62). The ANA IN LO pin is used to remote sense the source low (ground) to minimize system ground current related errors. Both *HI AND LO* SIGNALS MUST HAVE A BIAS CURRENT PATH BACK TO *SIGNAL RTN*. Figure 2 details the proper connections.



NOTES
1. BOTH HIGH AND LOW SIGNALS MUST HAVE A BIAS CURRENT PATH BACK TO GROUND AT THE AD1176. "ANA IN LO" SHOULD REFERENCE TO GROUND (SIGNAL RTN) AT THE SIGNAL SOURCE, VIA A MINIMUM OF RESISTANCE.
2. "DIG GND" AND "±15V RTN & GND" ARE STAR CONNECTED WITHIN THE CONVERTER, AND INTENDED TO BE SEPARATE OUTSIDE OF THE CONVERTER. HOWEVER, IF ±15V AND +5V POWER SHARE A SINGLE COMMON RETURN, THEN THAT COMMON MUST BE CONNECTED TO THE "±15V RTN & GND" PIN WHICH MUST BE CONNECTED VIA HEAVY COPPER TO THE "DIG GND" PIN. "SIGNAL RTN" (PIN 62) IS THE "NON-CURRENT CARRYING" GROUND, ONLY TO BE USED AS SHOWN AND AS GROUND REFERENCE FOR AN EXTERNALLY SUPPLIED REFERENCE SOURCE.

Figure 2. AD1175 Bus Driven Interface

Printed circuit board layout should insure that both analog inputs (Pins 58 and 59) are guarded by copper which is tied to SIGNAL RTN (Pin 62) on the front *and* back of the board.

Note that an offset error of up to one LSB per 120 Ω of source impedance can occur, due to input bias current, which may approach 20 nA at elevated temperatures.

REFERENCE CONNECTIONS

A very stable 6.95 V ±2% internal reference is filtered and brought out to REF OUT (Pin 66) of the converter. This output should be tied to REF IN (Pin 65) to accomplish the specifications for initial absolute accuracy. REF OUT is a high impedance output and should not be loaded in any way other than by REF IN (Pin 65). A buffered version of the reference applied to REF IN, and that which is used by the converter, is available at USER REF OUT (Pin 64).

When making ratiometric measurements, where the source excitation is derived from the converter reference, use the reference signal present at USER REF OUT (Pin 64). The load applied to Pin 64 should not exceed two milliamps. If an external reference source is to be used, it should be applied to REF IN (Pin 65).

POWER SUPPLIES AND GROUNDS

The power supply pins are all well bypassed internally to their respective common or ground pins. The converter is very tolerant of dc and low frequency noise (≤100 s of Hz) on any of the supplies, as evidenced in the power supply rejection specifications. High frequency noise (≥1 MHz) in excess of 10 mV on the ±15 V supplies could, however, degrade the converter's performance.

To avoid large, digital-rate, circulating ground currents, the system's analog supply common and that of the digital supply should be kept separate and then tied together at the converter by a heavy track (to supplement that which is internal to the converter) from ±15 V RTN & GND (Pins 48 and 49) to DIG GND (Pins 9 & 10).

If the logic supply and analog supply share a single common, then that common should be brought to ±15 V RTN & GND (Pins 48 and 49) and then from these pins a heavy track should be run to DIG GND (Pins 9 & 10).

RESET (Pin 5; Input)

After power-up and before access may be made to the converter, a reset of the internal microcomputer must be accomplished. The RESET (Pin 5) may be driven from an external source, such as may exist in most computer-based systems, or it may be connected to a simple RC circuit which will automatically generate a reset sequence upon power-up. See Figure 2 for the recommended circuit.

When driven from an external source, RESET must be held high for a minimum of 3 microseconds, but must not terminate before the +5 V logic supply and the ±15 V analog supply have been stable (>+4.7 V, and >±11 V) for 300 microseconds.

60 Hz/ $\overline{50\ Hz}$ (Pin 6; Input)

Pin 6 of the module selects either 33-1/3 milliseconds or 40 milliseconds for the signal integration time. This input is internally pulled up to 5 V via 10 kΩ and may be left open for 60 Hz normal mode rejection. The pin should be connected to Digital Ground for operation in a 50 Hz line frequency environment.

CONV CMD (Pin 4; Input)

A negative logic transition on this input causes a MODCON conversion to occur (see CALIBRATION section). A minimum hold time of 1.5 μs is required at both the High and the Low states, to operate properly. The $\overline{BUSY}$ output (Pin 24) will not respond, and BUSY (Bit 0) of the STATUS word will *not* be indicated, but all other bits of the STATUS word will be active. DATA RDY (Pin 25) will occur per Figure 8.

This input is provided to allow externally triggered conversions which will use the temporarily programmed gain and offset values (or the start-up defaults if no changes have been made).

DATA RDY (Pin 25; Output)

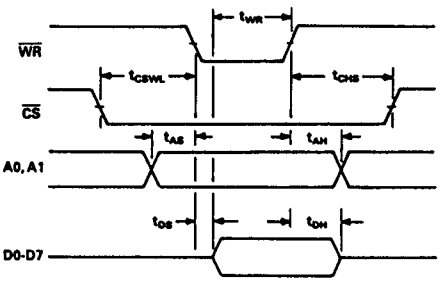
This signal will go to logic "1" when any conversion's new data has become stable in the output latches. It will remain high for the duration of the auto-zero phase (about 10 milliseconds) and go low at the end of that phase (at the end of BUSY).

$\overline{BUSY}$ (Pin 24; Output)

When a COMMAND byte is written to the microprocessor compatible port, this line is set low and remains low for the duration of the converter's response to that command. It is the opposite state of the BUSY bit within the STATUS byte.

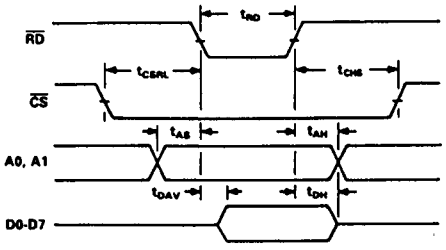
THE BUS INTERFACE

The AD1175's 8-bit microprocessor-compatible interface consists of an 8-bit, latched ,tri-stated, bidirectional port and its associated control lines: Chip Select ($\overline{CS}$), READ ($\overline{RD}$), WRITE ($\overline{WR}$) and two address bits (A1 and A0). Timing requirements for the bus interface are shown in Figure 3, and the operation of the interface is shown in Figure 4.



PARAMETER	DESCRIPTION	MIN	MAX	UNITS
t_{WR}	$\overline{WR}$ Pulse Width	0.1	20	μs
t_{CSWL}	Chip Select to $\overline{WR}$ Low	0		ns
t_{CHS}	Chip Select Hold Time	0		ns
t_{AS}	Address Setup time	10		ns
t_{AH}	Address Hold Time	0		ns
t_{DS}	Data Setup Time	−∞	5	μs
t_{DH}	Data Hold Time	20		ns

Write Cycle Timing Requirements



PARAMETER	DESCRIPTION	MIN	MAX	UNITS
t_{RD}	$\overline{RD}$ Pulse Width	150		ns
t_{CSRL}	Chip Select to $\overline{RD}$ Low	0		ns
t_{CHS}	Chip Select Hold Time	0		ns
t_{AS}	Address Setup Time	10		ns
t_{AH}	Address Hold Time	0		ns
t_{DAV}	Data Valid Time		100	ns
t_{DH}	Data Hold Time		80	ns

Read Cycle Timing Requirements

Figure 3. Interface Timing Requirements

	$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A1	A0	FUNCTION
READ	L	L	H	H	H	High CONV Data Byte READ
	L	L	H	H	L	Mid CONV Data Byte READ
	L	L	H	L	H	Low CONV Data Byte READ
	L	L	H	L	L	STATUS READ
WRITE	L	H	L	H	H	Unused
	L	H	L	H	L	Unused
	L	H	L	L	H	PARAMETER WRITE
	L	H	L	L	L	COMMAND WRITE
	X	H	H	X	X	DEVICE NOT SELECTED
	H	X	X	X	X	

NOTE THAT X = DON'T CARE

Figure 4. Bus Control Functions

	BIT # MSB 23	22	21	20	19	18	17	16	15	14	13	//	5	4	3	2	1	LSB 0	HEX
POS. OVERLOAD	1	1	1	1	1	1	1	1	1	1	1	//	1	1	1	1	1	1	FF,FF,FF
+ 1.25 × FULL SCALE	0	1	1	0	1	0	0	0	0	0	0	//	0	0	0	0	0	0	68,00,00
+ FULL SCALE	0	1	1	0	0	0	0	0	0	0	0	//	0	0	0	0	0	0	60,00,00
+ 1/2 SCALE	0	1	0	1	0	0	0	0	0	0	0	//	0	0	0	0	0	0	50,00,00
ZERO	0	1	0	0	0	0	0	0	0	0	0	//	0	0	0	0	0	0	40,00,00
− 1/2 SCALE	0	0	1	1	0	0	0	0	0	0	0	//	0	0	0	0	0	0	30,00,00
− FULL SCALE	0	0	1	0	0	0	0	0	0	0	0	//	0	0	0	0	0	0	20,00,00
− 1.25 × FULL SCALE	0	0	0	1	1	0	0	0	0	0	0	//	0	0	0	0	0	0	18,00,00
NEG. OVERLOAD	1	1	1	1	1	1	1	1	1	1	1	//	1	1	1	1	1	1	FF,FF,FF

Figure 5. Data Format

3

OUTPUT DATA FORMAT

The result of a conversion is made available in three 8-bit bytes (addressed as shown in Figure 4). The numeric result is presented as an offset binary number, where the offset value is equal to 2e22 (40,00,00 Hex), i.e., zero volts input yields this numerical output. Therefore, the nominal plus and minus full scale are 2e22 ±2e21, or 60,00,00 Hex and 20,00,00 Hex, respectively. For inputs greater than approximately 1.3× nominal full scale, the converter will indicate an overload error (Bit 5 of the STATUS byte) and will also flag the occurrence by forcing all "1s" in the conversion result, i.e., FF,FF,FF Hex. Bit 23 (MSB) cannot be a "1" for any legitimate conversion result, so that bit is used to flag an overload. The data format is depicted in Figure 5.

COMMAND BYTE

The COMMAND BYTE allows eight different instructions to be given. Five of these will require that a parameter be loaded into the PARAMETER* register prior to writing the command register. The commands are written at address 00 (ADDRESS lines A1 and A0, Pins 20 and 19, respectively) while a parameter is written to address 01. See Figure 4 for Bus Control Functions. Figure 8 details command timing requirements.

The commands are described below, preceded by an opcode name and the digital code (in hex). Figure 6 summarizes each command and its execution time.

DEFCON [00]

DEFault CONversion initiates a conversion, using the gain and offset values which are stored in the nonvolatile memory (power-up defaults).

MODCON [01]

MODified CONversion initiates a conversion using the gain and offset values which have been modified (since power-up) as in commands 02 through 07 below.

NEWOS [02]

NEW OffSet subtracts the result of the last conversion from all subsequent MODCON conversions, i.e., *acquire a new system offset*. The maximum range of this offset is 65,536 codes (= ±75 mV). Attempts to acquire an offset outside of this range will be ignored and BIT 5 and BIT 6 (Overload and command byte ERRor) will be set in the STATUS byte.

INCROS [03]

INCRease OffSet alters the offset (in LSBs) used by MODCON in the *positive* direction by a number between zero and 255 (decimal), which has already been written to PARAMETER*.

*The PARAMETER register retains the last word written to it. Any subsequent commands will repeatedly use that PARAMETER until it is updated.

This may be performed repeatedly until a maximum offset of +75 mV has been reached, as indicated by an Overload/BIT 5 response in the STATUS byte.

DECROS [04]

DECRease OffSet alters the offset (in LSBs) used by MODCON in the negative direction by a number between zero and 255 (decimal), which has already been written to PARAMETER*. This may be performed repeatedly until a maximum offset of −75 mV has been reached, as indicated by an Overload/BIT 5 response in the STATUS byte.

INCGAN [05]

INCrease GAiN by $N \times 0.01\%$, where N (a decimal number between 0 and 255) has already been written to PARAMETER*. This may be performed repeatedly until a maximum gain (<4.7 V full scale) has been reached, as indicated by an Overload/BIT 5 response in the STATUS byte. Further INCGAN commands will have no other effect.

DECGAN [06]

DECrease GAiN by $N \times 0.01$, where N (a decimal number between 0 and 255) has already been written to PARAMETER*. This may be performed repeatedly until a minimum gain (>5.6 V full scale) has been reached, as indicated by an Overload/BIT 5 response in the STATUS byte. Further DECGAN commands will have no other effect.

UPDATE [07]

Takes the current modified gain and offset values and writes them to nonvolatile memory as the new start-up defaults. To enable this function, decimal 165 (A5 in hex) must first be loaded into PARAMETER* – failure to do so will result in an ERRor (BIT 6) response in the STATUS byte.

Note: Codes other than 00 through 07 will do nothing, except cause an ERRor (BIT 6) response in the STATUS byte.

MNEMONIC	FUNCTIONAL DESCRIPTION	EXECUTION TIME (APPROXIMATE)
DEFCON	Initiate a Conversion Using the Power-Up Default Offset and Gain	50ms
MODCON	Initiate a Conversion Using the Modified Offset and Gain Values	50ms
NEWOS	Subtract System Offset (Last Conv. Result) from All MODCON Conversions	120µs
INCROS	Increase the Offset Used by MODCON Conversions	110µs
DECROS	Decrease the Offset Used by MODCON Conversions	110µs
INCGAN	Increase the Gain Used by MODCON Conversions	135µs
DECGAN	Decrease the Gain Used by MODCON Conversions	135µs
UPDATE	Write Most Recent Modified Offset & Gain Values to Nonvolatile Memory	48ms

Figure 6. Synopsis of Commands

THE STATUS BYTE

The STATUS byte contains eight bits of information about the current status of the AD1175. This byte may be examined by the host processor at any time. The individual bits in the status byte are assigned the following functions:

BIT 0 The BUSY bit is always set when the COMMAND BYTE is written, and cleared when the initiated routine has terminated. BUSY is also indicated at $\overline{\text{BUSY}}$ (Pin 24) of the module.

BIT 1 The CONVerting bit is set when the converter is in the active process of converting and computation. It is initiated by writing DEFCON or MODCON to the COMMAND-BYTE, or by a negative transition at CONV CMD (Pin 4).

BIT 2 The Data AVailable bit indicates that a new conversion is complete and the result is in the output latches. This bit sets to "1" at the conclusion of the converting process and remains "1" for the remainder of the minimum AUTO-ZERO time (about 10 milliseconds). It is reset to "0" at the end of BUSY.

BIT 3 The MODified bit, when set to "1," means that modified gain and offset values are being used for the current conversion; i.e., a conversion initiated by MODCON or an external signal at CONV CMD (Pin 4).

BIT 4 The VALue bit responds to COMMANDS 02 through 07 by setting to "1" at the end of BUSY, and remains until the next write to the COMMAND byte. This bit signals that a gain or offset value used by MODCON has been altered, or that the current MODCON gain and offset values have been loaded to nonvolatile memory as the new power-up defaults.

BIT 5 The Overload bit will be set following any conversion where the integrator has been exposed to an overload voltage. Following commands 03 through 06, it indicates that a parameter (gain or offset) has been incremented to its maximum or minimum possible value (note that further attempts to increment that parameter will not cause an overflow or underflow). Also, following NEWOS (02) command, this bit implies that an attempt was made, and ignored, to acquire an offset outside of the allowable range of ±75 mV.

BIT 6 The ERRor bit indicates one of the following: *1.* A COMMAND-BYTE was written which was not within the allowable range of 00 to 07. *2.* An update (07) command was attempted without the KEY number (165 decimal) having first been written to PARAMETER at ADDRESS 01. *3.* A NEWOS (02) command was attempted for a value outside the permissible range of ±32,768 codes (>75 mV) from zero.

BIT 7 The WaRMUP bit flags the three second time-out taken by the converter following RESET, to allow the reference and auto-zero circuits to settle. The converter will *not* convert during this time.

B7	B6	B5	B4	B3	B2	B1	B0
WRMUP	ERR	OL	VAL	MOD	DAV	CONV	BUSY

Figure 7. The Status Byte

CALIBRATION

The AD1175 is factory calibrated for plus and minus full scale (2e21) to be within ±50 μV of five volts, absolute. Since the converter will operate within specifications for inputs up to ten percent over nominal full scale, those inputs between ±5.5 V will be converted accurately. (See Figure 9 for typical linearity vs. input voltage.)

To correct for system offset voltage (particularly larger offset voltages – up to ±75 mV) the NEWOS (03) command subtracts the result of the last conversion from all subsequent MODCON conversions. If source noise is a concern when making the offset adjustment, follow a single NEWOS command with multiple MODCON conversions, average the results and adjust offset incrementally using the INCROS (03) or DECROS (04) commands.

The INCGAN 05 and DECGAN 06 commands are the *coarse gain* increment and decrement controls, respectively. The minimum gain attainable will require greater than 5.6 V to achieve a full-scale output. At maximum gain, less than 4.7 V will be required to yield a full-scale indication. The user accessible GAIN ADJ potentiometer is the vernier, or *fine gain* trim (10 turns, with a total adjustment range of about ±0.006 FS).

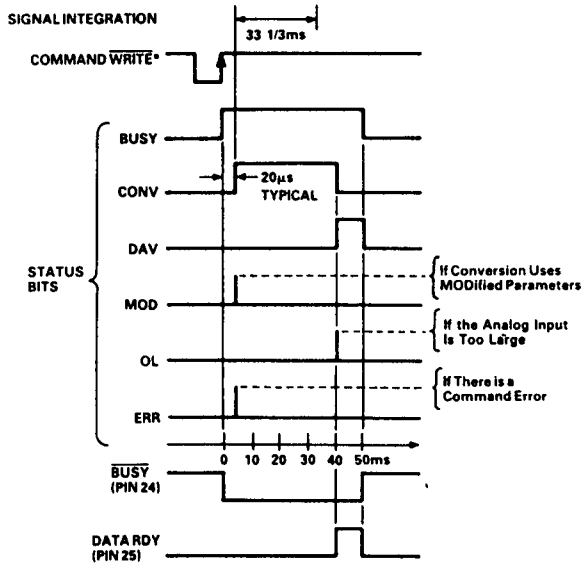
The modified offset and gain resulting from commands 02, 03, 04, 05 and 06 are used only when conversions are initiated by MODCON (command 01), or conversions triggered by a negative logic transition at the CONV CMD (Pin 4 of the converter). This pin requires a minimum hold time of 1.5 μs at both the High and the Low states in order to operate properly.

The GAIN ADJ potentiometer changes the overall gain for both positive and negative inputs. The BAL ADJ potentiometer changes the gain for positive inputs only and allows setting of plus and minus full-scale tracking to within ±1 ppm.

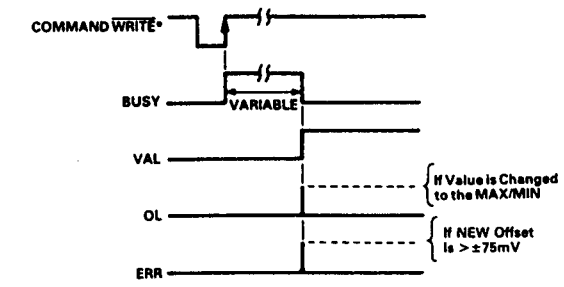
To Calibrate the AD1175:

1. Attach a calibration source and set its output to zero volts.
2. Perform MODCON conversions and null out any observed offset (via external computation, or by executing one or more of the AD1175's offset controlling commands: INCROS, DECROS and NEWOS).
3. Set the GAIN ADJ potentiometer fully clockwise (10 turns, i.e., maximum gain).
4. Apply a negative full-scale calibration voltage (−4.7 V to −5.6V).
5. Using the INCGAN or DECGAN command, coarse adjust the gain such that a subsequent MODCON conversion yields a result just larger than minus full scale. In other words, a subsequent DECGAN by 01 would just yield a result that is less than or equal to minus full scale.
6. Adjust the GAIN ADJ potentiometer to yield the precise value desired by turning counterclockwise and observing conversion results. When the correct gain is reached, rotate the potentiometer about 3 degrees in the opposite direction to remove the tension from its wiper.
7. Switch the polarity of the calibration source to positive.
8. Adjust the BAL ADJ potentiometer to yield the same gain as that achieved in Step 6 above.
9. Save the new offset value and coarse gain value, if you want them to become the power-up defaults, by performing UPDATE (Command 07).

Note: See the COMMAND BYTE section for details of command operation.

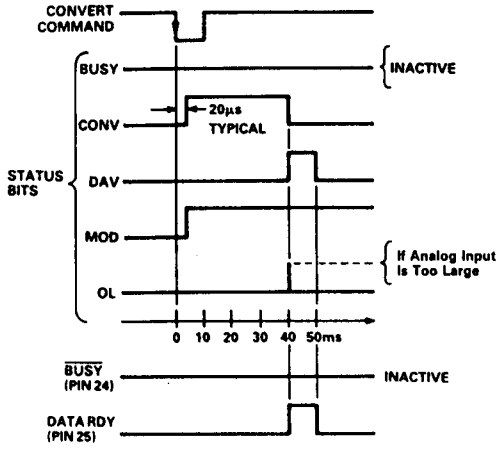


a. COMMAND BYTE Initiated Conversion



*NOTE: COMMAND WRITE Always Causes Rewrite of the Entire STATUS Byte. For Example: If the Overload Bit (OL) is Set as the Result of a Conversion, It Will Remain Set in the STATUS Byte Until the Next COMMAND WRITE.

b. COMMAND BYTE Initiated Change to Gain and/or Offset



c. CONVERT COMMAND (Pin 4) Initiated Conversion

Figure 8. Command Timing Requirements

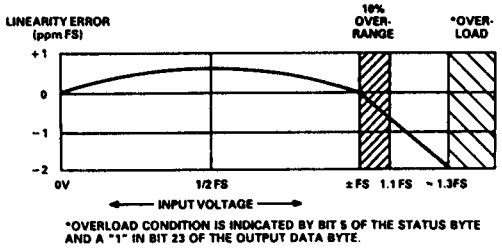


Figure 9. Typical Linearity Transfer Function

FACTORY TESTING

Each AD1175 converter is factory calibrated via test apparatus designed and constructed by Analog Devices. The heart of the test system is a digitally programmable voltage reference capable of sub-ppm accuracy and stability. Calibration of the test system is verified daily using the highest precision instruments commercially available, e.g., FLUKE* model 720A Kelvin Varley voltage divider (accurate to within ±0.1 ppm[1]) and model 732A dc secondary voltage standard (accurate to within ±1.5 ppm of the international volt[1]).

IBM PC INTERFACE

Figure 10 is an example of an AD1175/IBM interface suitable for the IBM PC, XT or AT** personal computers. In this case, the AD1175 is interfaced in the I/O space; a DIP switch controls the specific location of the AD1175 within the available address space.

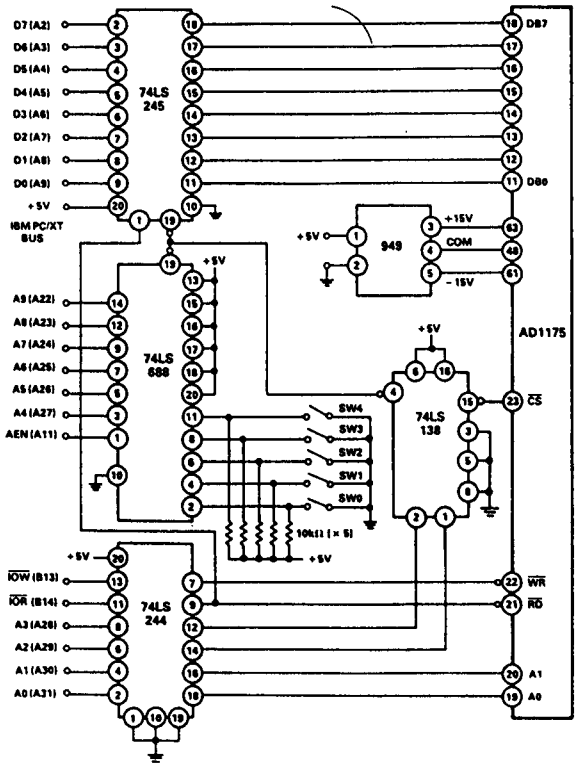


Figure 10. AD1175 to IBM PC/XT/AT Interface

*FLUKE is a registered trademark of John Fluke Manufacturing Company, Inc.

**IBM PC/XT/AT is a trademark of International Business Machines Corp.

[1]Traceable to the NATIONAL BUREAU OF STANDARDS.

INTERFACING TO AN 8051 MICROCONTROLLER

Figure 11 shows how the AD1175 may be interfaced to an 8051 microcontroller using a technique commonly called "byte banging," where the control lines and data bus of a device are manipulated under firmware control. This "byte banging" technique can be adapted to most microprocessors and is useful in situations where a conventional bus structure is either nonexistent or unavailable for use.[1]

The AD1175's data bus is connected to the 8051 using I/O lines P2.0 through P2.7. The address lines A0 and A1 are connected to I/O lines P1.0 and P1.1 respectively. The RD/ and WR/ lines are connected to P1.2 and P1.3. The CS/ line of the AD1175 is grounded when it is the only device connected to the 8051, but multiple AD1175s could easily be connected in the same way if each CS/ line were separately controlled.

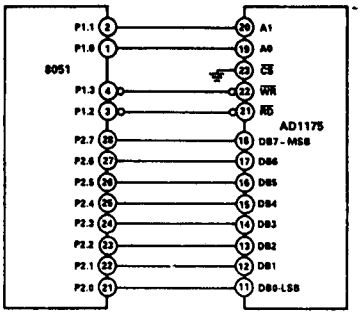


Figure 11. Simple AD1175 to 8051 Interface

To initialize the interface, first write "1"s to the port pins connected to the data bus and the RD/ and WR/ control lines. This puts the 8051 I/O lines into a lightly "pulled up" state, simulating a tri-stated condition on the bus to insure that neither RD/ nor WR/ are selected:

```
INIT:   SETB  P1.2        ;DISABLE RD/
        SETB  P1.3        ;AND WR/
                          ;
        MOV   P2, #OFFH   ;SET P2 TO ALL ONES
```

To write a command to the AD1175, first set the state of the P1.1 and P1:0 lines for the address corresponding to the byte to be written to (00=COMMAND BYTE, 01=PARAMETER). Set the P2 port to the command data, then strobe the WR/ line by first clearing the P1.3 line and then setting it:

```
WRCMD:  CLR   P1.0        ;FIRST CLEAR A0 AND A1
        CLR   P1.1        ;TO POINT TO CMD BYTE
                          ;
        MOV   P2, #00     ;00 IS THE OPCODE FOR
                          ;A DEFAULT MODE
                           CONVERSION
                          ;
        CLR   P1.3        ;STROBE THE WR/ LINE
        SETB  P1.3        ;ONE TIME
                          ;
        MOV   P2, #OFFH   ;SET DATA BUS TO
                          ;ALL ONES
```

To read a byte from the AD1175, first set the P1.0 and P1.1 lines to point to the address of the byte desired. Bring the RD/line low, reading the contents of P2. Return the RD/ line high:

```
RDSTAT: CLR   P1.0        ;POINT TO STATUS BYTE
        CLR   P1.1        ;
                          ;
        CLR   P1.2        ;BRING RD/ LINE LOW
        MOV   A,P2        ;READ CONTENTS OF BUS
        SETB  P1.2        ;RESTORE RD/ LINE HIGH
```

[1]Note that the 8051 microcontroller *does* contain a conventional bus structure; the "byte banging" interface shown here is presented as an example of an alternative technique.

AC5005

. . . an IBM PC/XT/AT Compatible Evaluation Board for the AD1175K

FEATURES

Compatible to IBM PC/XT/AT* or Equivalent
Menu-Driven Demonstration Software
Full Documentation
- **Example Listings of BASIC Programs**
- **Schematic**
- **Assembly Drawing**

Complete Set of Tools to Evaluate the AD1175K 22-Bit Resolution Integrating A/D Converter

APPLICATIONS

Laboratory DVM
Product Test and Measurement
Analytical Instrumentation
Material Analysis
Seismic Analysis

GENERAL DESCRIPTION

The AC5005 is an evaluation board for Analog Devices' AD1175K and is designed to plug directly into the backplane of an IBM PC/XT/AT and compatibles. The AC5005 is offered as a support tool to enable users to easily and quickly evaluate Analog Devices' AD1175K 22-bit multi-slope integrating A/D converter. The AC5005 comes with a demonstration program written in BASICA that completely exercises the functions of the AD1175K and emulates a 6 1/2 digit DVM. The onboard multiplexer allows selection via software from four differential analog input channels. A set of ten digital I/O lines are available to the user for control of lamps and actuators as well as to test switch positions. The AC5005 plugs directly into an IBM PC or compatible. Armed with an IBM PC and an AC5005 evaluation board, the user is ready to execute the demonstration program and evaluate the operation of the AD1175K.

A user's guide provides the user with all the information required to put the AC5005/AD1175K pair into operation. The schematic of the AC5005 is provided as an example of how to interface the AD1175K to a computer bus. The AC5005 is very easy to configure. It has one set of DIP switches to select the board's base address and one set of jumpers to select either 50 Hz or 60 Hz line cycle. All the tools needed to evaluate the AD1175K come with the AC5005. There is even a short example program listing written in BASIC to demonstrate the ease of programming the AD1175K.

PRODUCT HIGHLIGHTS

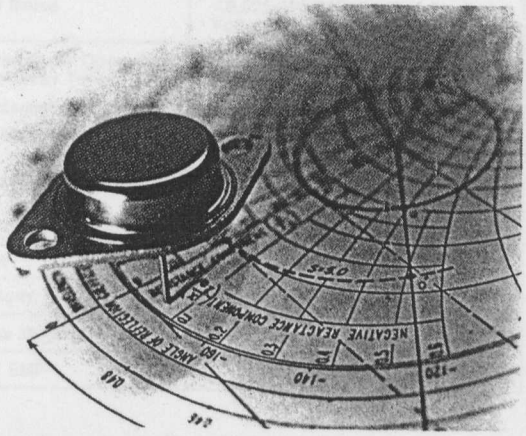
1. Plugs directly into IBM PC/XT/AT or compatibles.
2. Evaluates the AD1175K 22-bit multi-slope integrating A/D converter without having to build a breadboard or prototype.
3. Comes complete with software and programming examples to exercise all of the AD1175K's functions and emulate a 6 1/2 digit DVM.
4. AC5005 schematic and assembly drawing are provided to be used as examples of how to interface theAD1175K to a microprocessor bus.
5. Turnkey solution for laboratory measurement and analytical instrumentation.

*IBM PC/XT/AT is a trademark of International Business Machines Corp.

VISHAY
...to be precise®

Hermetically-sealed Precision Power and Current Sensing Resistors

VISHAY MODELS VHP-3, VHP-4



The basic features of Vishay Bulk Metal® foil resistors such as tight resistance tolerance, fast response time, low TCR, and exceptional long-term stability are available for power-circuit applications. Non-inductive design, current sensing applications, deflection amplifiers, constant current power supplies, forced balance electronic scales, graphic display computers, character generation on CRT's, electron beam controls etc. The standard heat-sinking TO3-configuration case can be screw mouted directly on a metal chassis for maximum heat-transfer capability. All-welded construction ensures maximum reliability.

Hermetically-sealed metal case protects against environmental conditions and insures exclient long-term stability.

FEATURES

- Temperature Coefficient of Resistance — Nominal TCR[3]: +0.6ppm/°C (0°C to +25°C); −0.6ppm/°C (+25°C to +60°C); +2.2ppm/°C (−55°C to +25°C); −1.8ppm/°C (+25°C to +125°C)
- Standard TCR Spread from Nominal[4]: ±1.5 ppm/°C (0°C to +25°C and +25°C to +60°C); ±2.0ppm/°C (−55°C to +25°C and +25°C to +125°C)
- Maximum TCR Spread from Nominal[5]: ±2.5ppm/°C (0°C to +25°C and +25°C to +60°C)
  ±2.3ppm/°C (-55°C to +25°C and +25°C to +125°C)
- Selected TCR Tracking: 0.5ppm/°C
- Load-Life Stabilty:
  0.01% Maximum ΔR, 3 watts on heat sink at 25°C.
  0.05% Maximum ΔR, 10 watts on heat sink at 25°C.
  For 3 watts, free air, ΔR's above not applicable.
- Shelf life stability 5ppm/year
- Power Rating:
  10 watts on heat sink at 25°C.
  3 watts, free air at 25°C.
- Resistance Tolerance (Initial Resistance Accuracy): ±0.01% tightest to ±5.0% loosest
- Resistance range: 0.05Ω to 80KΩ
- Current Noise: <0.025 μV (RMS)/Volt of Applied Voltage
- Thermal EMF:
  0.5 V/°C (lead effect), 4 μ V/watt (power effect).
- Non-Inductive Design: Standard

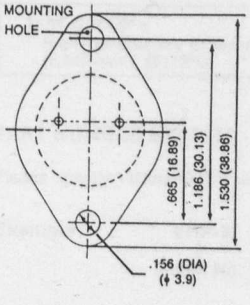
Res. Range	VHP-3*	
	Tightest Resistance Tolerance[1]	TCR
50Ω to <80K	±0.01%	±5 ppm/°C
25Ω to <50Ω	±0.02%	±7 ppm/°C
10Ω to <25Ω	±0.05%	±10 ppm/°C
5Ω to <10Ω	±0.1%	±13 ppm/°C
2Ω to <5Ω	±0.25%	±20 ppm/°C
1Ω to <2Ω	±0.5%	±25 ppm/°C
0.5Ω to <1Ω	±1.0%	±50 ppm/°C
0.25Ω to <0.5Ω	±2.0%	
0.1Ω to <0.25Ω	±5.0%	
0.05Ω to <0.1Ω	N.A.	N.A.

*Weight = 15 gms. max. †VHP-4 available only up to 500Ω.

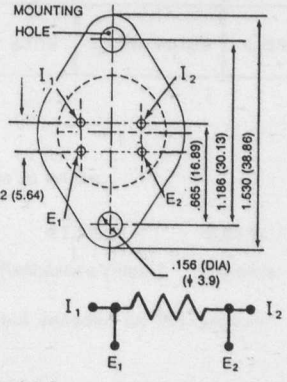
Res. Range	VHP-4*†	
	Tightest Resistance Tolerance[1]	TCR
10Ω to <500Ω	±0.01%	±5 ppm/°C
5Ω to <10Ω	±0.02%	±6 ppm/°C
2Ω to <5Ω	±0.05%	±8 ppm/°C
1Ω to <2Ω	±0.1%	±10 ppm/°C
0.5Ω to <1Ω	±0.25%	±15 ppm/°C
0.25Ω to <0.5Ω	±0.5%	±20 ppm/°C
0.1Ω to <0.25Ω	±1.0%	±25 ppm/°C
0.05Ω to <0.1Ω	±2.0%	±30 ppm/°C

STANDARD IMPRINTING AND DIMENSIONS

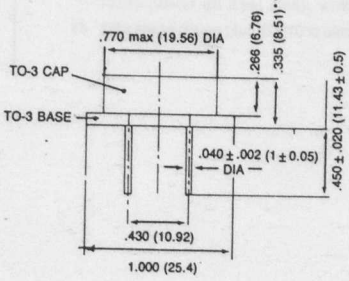
VHP-3



VHP-4

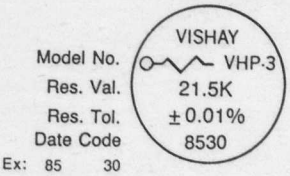


Note: Dimensions are in inches (millimeters)



**.375 (9.53) — VHP-4

STANDARD MARKING ARRANGEMENT



Case not active; insulators not required; mounting hardware not supplied.

"VHP-3" SPECIFICATIONS[2]/TABLE II

Stability Load-Life at 2000 Hours Shelf-Life	0.05% Maximum ΔR under full rated power of 10 Watts at 25°C 0.0005% (5ppm) Maximum ΔR/Year
Power Rating at 25°C At Ambient Temperature (See Graph II)	10 Watts or 3 Amps[6]—Heat Sink[7] 3 Watts or 3 Amps[6]—Free Air
Current Noise	<0.025μV (RMS)/Volt of Applied Voltage (−32dB)
High-Frequency Operation Rise/Decay Time Inductance[8] (L) Capacitance (C)	 1.0ns @ 1kΩ 0.1μH Maximum; 0.08μH Typical 1.0pF Maximum; 0.5pF Typical
Voltage Coefficient	<.1ppm/V[9]
Operating Temperature Range	−55°C to +175°C
Hermeticity	10^{-7} Atmospheric cc/sec.
Maximum Working Voltage[10]	600V
Thermal EMF[11]	0.5 μV/°C max. (lead effect) 4.0 μV/watt (power effect)

POWER RESISTOR ENVIRONMENTAL PERFORMANCE COMPARISON/TABLE III

Test	Method Para[12]	Mil-R-39009B Limits	VHP-3 Typical[2] Test Data[16]
TEST GROUP I Conditioning	4.7.2	±0.2% +0.05Ω	0.03%
TEST GROUP II Resistance Temperature Characteristic[13] (-55°C to +125°C)	4.7.4	<1Ω: ±100ppm/°C; 1Ω to 19.6Ω: ±50ppm/°C; ≥20Ω; ±30ppm/°C	See Graph VI
Temperature[14] (2 hours @ 175°C)	4.7.5	±0.5% +0.05Ω	0.01%
Low-temperature Storage	4.7.17	±0.3% +0.05Ω	0.005%
DWV (750V @ atmos. press.)	4.7.6	±0.2% +0.05Ω	0.005%
Insulation Resistance	4.7.7	10^4MΩ	>10^4MΩ
Low-temperature Operation	4.7.8	±0.3% +0.05Ω	0.01%
Momentary overload[15] (5 sec. @ 750V rms)	4.7.9	±0.3% +0.05Ω	0.03%
Moisture Resistance	4.7.10	±0.5% +0.05Ω	0.02%
Terminal Strength	4.7.11	±0.2% +0.05Ω	0.005%
TEST GROUP III Shock - Medium Impact (Post-test DWV @ 750V)	4.7.12	±0.2% +0.05Ω	0.005%
Vibration - High-Frequency (Post-test DWV @ 750V)	4.7.13	±0.2% +0.05Ω	0.005%
TEST GROUP IV Life - 10 watts for 2,000 hours @ 25°C	4.7.14	±1.0% +0.05Ω	0.01%
(80% power for 2,000 hours @ 70°C)	—	—	0.04%
TEST GROUP V High-Temperature Exposure (2,000 hours @175°C)	4.7.15	±1.0% +0.05Ω	0.05%

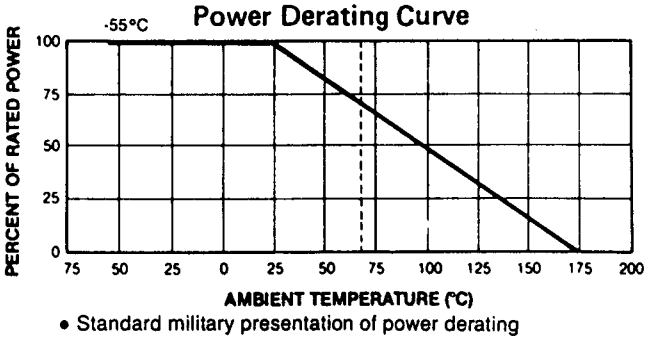
PART NUMBER SYSTEM

Please specify Vishay VHP-3 resistors as follows:

Example: **VHP-3** (Model No.) **21.5K** (Resistance Value) **0.01%** (Tolerance)

Resistance Value Designation is not encoded on this product.

GRAPH II
Power Derating Curve



• Standard military presentation of power derating

NOTES

1. Standard Resistance tolerances—±0.01%; ±0.02%; ±0.05%; ±0.1%; ±0.25%; ±0.5%; ±1.0%; ±2.0%; ±5.0% (±0.005% tolerance available on special order).
2. Maximum is 1.0% A.Q.L. standard for all specifications except TCR. (For TCR information, see notes 3-7.)
   Typical is a designers reference which represents that 85% of the units supplied, over a long period of time, will be at least the figure shown or better.
3. Vishay Nominal TCR is defined as the chord slopes of the relative resistance/temperature (RT) curve from 0°C to +25°C and +25°C to +60°C ("Instrument" Range); and from −55°C to +25°C and +25°C to +125°C ("Military" Range). These specifications and the definition of Nominal TCR apply to all resistance values including low-value resistors.
4. Vishay Standard TCR Spread is defined as a designers reference which represents that at least 92% of the units, and 82% of the lots, supplied by Vishay will be within the stated band centered on the nominal curve. This definition of the Vishay Standard TCR Spread from Nominal applies to all resistance values. However, as the resistance value decreases below 60Ω, the Vishay Standard TCR Spread from Nominal specification starts to increase. (See Graph VIII page 3).
5. Vishay Maximum TCR Spread is defined as the 3σ (sigma) limit of a normal Gaussian distribution (99.73% of a production lot) which is within a band, centered on the nominal curve. This Vishay Maximum TCR Spread is no greater than ±2.5ppm/°C from nominal throughout the full temperature range. This definition of the Vishay Maximum TCR Spread from Nominal applies to all resistance values. However, as the resistance value decreases below 80Ω, the Vishay maximum TCR Spread from Nominal specification starts to increase. (See Graph VIII page 3).
6. Whichever is lower.
7. Heat sink chassis dimensions and requirements per Mil-R-39009/1B:

	Inches	mm
L:	6.00	152.4
W:	4.00	101.6
H:	2.00	50.8
T:	0.04	1.0

8. Inductance (L) due mainly to the leads.
9. The resolution limit of existing test equipment (within the measurement capability of the equipment, or "essentially zero.")
10. Not to exceed power rating of resistor.
11. μV/°C relates to EMF due to lead temperature difference and μV/watt due to power applied to the resistor.
12. The VHP-3 test data as compared to Mil-R-39009 is shown for illustration purposes. Vishay test conditions that deviate from the Mil test method are noted within parentheses.
13. Maximum specifications. See VHP-3 and VHP-4 Tabulation p. 12.
14. Maximum ambient temperature rating is +175°C.
15. Maximum overload rating is 15 Watts (5x rated power in free air; 1.5x rated power on heat sink), with applied voltage not to exceed 750V.
16. ΔR's are as shown plus .0001Ω to allow for measurement errors at low resistance values.