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 ±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

#### Summary

<table>
<thead>
<tr>
<th>1 Application of precision coil/magnet actuators</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Coil/magnet geometries and actuators</td>
<td>1</td>
</tr>
<tr>
<td>1.2 X-ray interferometry</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Coil arrangements for translation and twist</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Requirements for a programmable current source</td>
<td>7</td>
</tr>
<tr>
<td>1.5 (Un)suitability of commercial programmable current sources</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2 Design ideas, problems and constraints</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Generic problems in high-precision electronic instrumentation</td>
<td>12</td>
</tr>
<tr>
<td>2.1.1 Drift with temperature</td>
<td>13</td>
</tr>
<tr>
<td>2.1.2 Drift with time</td>
<td>15</td>
</tr>
<tr>
<td>2.1.3 Noise</td>
<td>15</td>
</tr>
<tr>
<td>2.1.4 Interference in mixed digital/analogue circuits</td>
<td>16</td>
</tr>
<tr>
<td>2.2 Applicability of published designs for programmable current sources</td>
<td>17</td>
</tr>
<tr>
<td>2.2.1 Review of - Programmable current supply for inductive load (1986)</td>
<td>18</td>
</tr>
<tr>
<td>2.2.2 Review of - Microcomputer-controlled, programmable current source for NMR measurements at very low temperature (1991)</td>
<td>18</td>
</tr>
<tr>
<td>2.2.3 Review of - Development of a programmable current source (1993)</td>
<td>19</td>
</tr>
<tr>
<td>2.3 Design ideas for a suitable 1 ppm DAC</td>
<td>20</td>
</tr>
<tr>
<td>2.3.1 Pulse width modulation (1:10&lt;sup&gt;6&lt;/sup&gt;)</td>
<td>21</td>
</tr>
<tr>
<td>2.3.2 Multiple level pulse-width-modulated DAC (1:64)</td>
<td>22</td>
</tr>
<tr>
<td>2.3.3 Coarse and fine DACs</td>
<td>24</td>
</tr>
<tr>
<td>2.4 Minimising risk in the NPL/DTI project</td>
<td>25</td>
</tr>
<tr>
<td>2.4.1 Coil self-heating effects and cures</td>
<td>26</td>
</tr>
<tr>
<td>2.4.2 Embedded versus sub-system controllers</td>
<td>27</td>
</tr>
<tr>
<td>2.5 Proposed design</td>
<td>28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 Drive circuits in detail</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Power supplies</td>
<td>31</td>
</tr>
<tr>
<td>3.2 Microcomputer</td>
<td>33</td>
</tr>
<tr>
<td>3.3 Precision analogue-to-digital converter module</td>
<td>34</td>
</tr>
</tbody>
</table>
3.4 Coil drive circuit 35
3.5 Heater/Drive 2 circuit 38
3.6 Auxiliary devices 39
  3.6.1 Power supply for chopper-stabilised amplifiers 39
  3.6.2 Clock generator for chopper-stabilised amplifiers 39
  3.6.3 Monitor ADC and temperature sensors 40
  3.6.4 Isolated serial link 41
  3.6.5 Non-volatile memory 42
  3.6.6 Configuration switches 42
3.7 Component layout and Printed Circuit Board 43

4 Embedded Software 48
  4.1 Initialisation 49
  4.2 Serial interface and command structure 50
  4.3 Timer interrupt and service routines 51
  4.4 Command decoder and command execution 52
  4.5 Floating-point interpreter 57

5 Slave PC Software 60
  5.1 Main program 60
  5.2 Screen presentations 62
  5.3 Calibration 65
  5.4 Communication ports and interrupt driven buffers 70
  5.5 Closed-loop control strategy 71
  5.6 Keyboard operation 72
  5.7 Messages from and to host computer 72
  5.8 Utility functions and procedures 73

6 Performance Measures 74
  6.1 Noise sources and predicted performance 75
    6.1.1 Noise in voltage reference and DAC's 75
    6.1.2 Response of Voltage-to-Current converter 76
    6.1.3 Precision current sensing resistor RS1 80
  6.2 Measured responses 81
    6.2.1 Precision ADC 81
    6.2.2 Power-on/off transients 84
    6.2.3 Step response 85
    6.2.4 Noise in the coil 86
List of Figures

Figure 1.1 - Flexure mechanism with coil/magnet actuator 2
Figure 1.2 - Silicon monolith [1.14] 5
Figure 1.3 - Silicon monolith with two magnets [1.18] 5
Figure 1.4 - Coil Arrangements 6
Figure 1.5 - Jumping fringes 9
Figure 1.6 - Commercial current sources [1.20 to 1.23] 10
Figure 2.1 - Commercial DACs 20
Figure 2.2 - Pulse-width-modulated DAC 21
Figure 2.3 - Multiple-level PWM N-Bit DAC 23
Figure 2.4 - Piecewise linear DAC 25
Figure 2.5 - Proposed heating jacket 27
Figure 2.6 - General arrangement of equipment 28
Figure 2.7 - Block diagram of drive 29
Figure 3.1 - Connections to Precision ADC Module 34
Figure 3.2 - DAC(s) and summing amplifier 36
Figure 3.3 - Circuit for voltage-to-current converter 37
Figure 3.4 - Clock timing for chopper-stabilised amplifiers 40
Figure 3.5 - Top-view of assembled PCB 45
Figure 3.6 - Oblique-view of chassis and PCB 46
Figure 3.7 - Appearance of completed system 47
Figure 4.1 - Port allocation 50
Figure 4.2 - Command address table 53
Figure 4.3 - Preparing primary DAC 54
Figure 4.4 - Interpreter operations 58
Figure 4.5 - Example of using interpreter 59
Figure 5.1 - Main program listing 61
Figure 5.2 - Screen when Source = HOST PC 63
Figure 5.3 - Screen when entering CALIBRATION 66
Figure 5.4 - Typical recorded calibration data 67
Figure 5.5 - Coding for calibrate-cycle 68
Figure 5.6 - Combined DACs during calibration 69
Figure 5.7 - Assignment of communication channels 70
Figure 6.1 - Component noise 75
Figure 6.2 - Voltage-to-Current converter simulated by PSPICE 76
Figure 6.3 - vtoi1 Transient response 77
Figure 6.4 - vtoil Frequency response 78
Figure 6.5 - vtoi2 Frequency response 79
Figure 6.6 - Noise of OPA654 80
Figure 6.7 - Precision ADC readings 82
Figure 6.8 - Precision ADC readings and filtered values 83
Figure 6.9 - Low frequency noise of filtered ADC readings 83
Figure 6.10 - Power on and off transients 84
Figure 6.11 - Step response 85
Figure 6.12 - High gain test amplifier 87
Figure 6.13 - Noise spectra of coil-current 88
Figure 6.14 - MATLAB script files for spectral analysis 89
Figure 6.15 - Frequency response of test amplifier 90
Figure 6.16 - Error in ramped output 91
Figure 6.17 - NPL Monolith - 2.1 fringes/mV or 0.67nm/mV - G3 93
Figure 6.18 - NPL Monolith - 2.1 fringes/mV or 0.67nm/mV - G4 94
Figure 6.19 - Warwick (stiff) Monolith - 0.067 fringes/mV or 20pm/mV - G5 94
Figure 6.20 - Warwick (stiff) Monolith - 0.067 fringes/mV or 20pm/mV - G6 95
Figure 7.1 - OPA177E and OPA627BM specifications 98
Figure 7.2 - Use of external voltmeter 100
Figure 7.3 - Voltage halving and inversion 101
Figure 7.4 - Differential to single-ended conversion 102
Figure 7.5 - Switched capacitor V-to-I with bias 103
Figure 7.6 - Response of switched capacitor V-to-I 104
Figure 7.7 - Response of switched capacitor V-to-I 105
Figure 7.8 - Deglitched and demultiplexing a DAC 106
Figure 7.9 - Demultiplexed DAC waveforms 107
Figure 7.10 - Better demultiplexing 108
Figure 7.11 - Interpolation in a time demultiplexed DAC 109
Figure 7.12 - Using a 10-bit PWM DAC as the coarse DAC 110
Figure 7.13 - Power on/off protection 111
Figure 7.14 - Proposal for dual drive 112
Figure 8.1 - Circuit depicting DACs with different time constants 115
Figure 8.2 - Responses when DACs have different time constants 116
Figure 8.3 - Embedded diagnostic amplifier 117
Figure 8.4 - Alternative communication schemes 119
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analogue to digital converter</td>
</tr>
<tr>
<td>BCD</td>
<td>Binary coded decimal</td>
</tr>
<tr>
<td>cps</td>
<td>counts per second</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital to analogue converter</td>
</tr>
<tr>
<td>dc</td>
<td>Direct current</td>
</tr>
<tr>
<td>DNL</td>
<td>Differential non-linearity</td>
</tr>
<tr>
<td>DTI</td>
<td>Department of trade and industry</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Electrically erasable programmable read only memory</td>
</tr>
<tr>
<td>FSO</td>
<td>Full scale output</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institution of Electrical and Electronic engineers</td>
</tr>
<tr>
<td>INL</td>
<td>Integral nonlinearity</td>
</tr>
<tr>
<td>LSB</td>
<td>Least significant bit</td>
</tr>
<tr>
<td>MSB</td>
<td>Most significant bit</td>
</tr>
<tr>
<td>NPL</td>
<td>National physical laboratory</td>
</tr>
<tr>
<td>NS</td>
<td>National semiconductor corporation</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed circuit board</td>
</tr>
<tr>
<td>ppm</td>
<td>parts per million</td>
</tr>
<tr>
<td>PTB</td>
<td>Physikalisch-Technische Bundesanstalt</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse width modulation</td>
</tr>
<tr>
<td>PZT</td>
<td>Piezo-electric transducer</td>
</tr>
<tr>
<td>RAM</td>
<td>Random access memory</td>
</tr>
<tr>
<td>RF</td>
<td>Radio frequency</td>
</tr>
<tr>
<td>rms</td>
<td>Root mean square</td>
</tr>
<tr>
<td>THD</td>
<td>Total harmonic distortion</td>
</tr>
</tbody>
</table>
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 in 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 \( \frac{dH}{dr} \), \( \frac{d^2H}{dr^2} \) and \( \frac{d^3H}{dr^3} \) 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](image)

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](image)

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 $10 \, \mu m$ 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 $1 \, \mu m$ 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 $\pm 10 \, \mu m$ 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 ($\pm 1/2$ 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 ($\pm 1/4$ 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 $\pm \frac{1}{4}$ fringe is unnecessarily tight, but on a binary scale it is the next best range when $\pm \frac{1}{2}$ fringe is rejected.

![Diagram of Fringe Jumping](image)

**Figure 1.5 - Jumping fringes**

Since the range of $10 \, \mu m$ contains 31949 fringes, which maps closely to a 15-bit scale with 32768 intervals, the corresponding required integral non-linearity (INL) is $\pm \frac{1}{4}$ of the least significant bit (LSB). Expressed as a percentage of full scale this is $\pm 0.0007\%$, 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 $\pm \frac{1}{2}$ LSB, such as part DAC729KH by Burr Brown.

The lower bound of $\pm 0.0007\%$ 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/$^\circ$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.

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Range</th>
<th>Step</th>
<th>Res.</th>
<th>Temp.Co</th>
<th>Linearity</th>
<th>Noise</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keithley</td>
<td>224</td>
<td>±100mA</td>
<td>50µA</td>
<td>12</td>
<td>?</td>
<td>?</td>
<td>100ppm</td>
<td>Non-Inductive load</td>
</tr>
<tr>
<td>Fluke</td>
<td>PM2831</td>
<td>±1A</td>
<td>12</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>2mA</td>
<td></td>
</tr>
<tr>
<td>HP</td>
<td>6625A</td>
<td>500mA</td>
<td>100µA</td>
<td>14</td>
<td>?</td>
<td>?</td>
<td>0.1mA</td>
<td></td>
</tr>
<tr>
<td>Marconi</td>
<td>103A</td>
<td>±10mA</td>
<td>100nA</td>
<td>20</td>
<td>?</td>
<td>0.005%</td>
<td>?</td>
<td>DC Reference</td>
</tr>
</tbody>
</table>

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 10μV 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 $3\sigma$ 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 ±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 than 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 $\mu$V/°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 $\mu$V 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 \( 5 \times 10^{-5} \). 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 \( \pm 10 \text{ ppm/°C} \) is given as typical. Selection of the 'C' grade part would have offered monotonicity to 15 bits and a maximum temperature coefficient of \( \pm 15 \text{ ppm/°C} \).

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.

<table>
<thead>
<tr>
<th>Device</th>
<th>Res. Mono.</th>
<th>Gain ±ppm</th>
<th>Power</th>
<th>Manufacturer</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC-HP16B</td>
<td>16</td>
<td>15</td>
<td>675</td>
<td>Datel</td>
<td>Resolution too low</td>
</tr>
<tr>
<td>AD11458BG</td>
<td>16</td>
<td>0.1</td>
<td>2.5</td>
<td>Analog Devices</td>
<td>Lowest power and drift</td>
</tr>
<tr>
<td>AD1139J</td>
<td>18</td>
<td>4</td>
<td>825</td>
<td>Analog Devices</td>
<td>'Best' DAC found</td>
</tr>
<tr>
<td>AD1860</td>
<td>18</td>
<td>25</td>
<td>110</td>
<td>Analog Devices</td>
<td>PCM Audio DAC</td>
</tr>
<tr>
<td>DAC1138K</td>
<td>18</td>
<td>NA</td>
<td>900</td>
<td>Analog Devices</td>
<td>Requires external reference to achieve quoted drift.</td>
</tr>
<tr>
<td>DAC72USHK</td>
<td>18</td>
<td>5</td>
<td>1200</td>
<td>Burr Brown</td>
<td>Monotonic to 16 bits, but high power</td>
</tr>
<tr>
<td>PCM63P</td>
<td>20</td>
<td>16/17</td>
<td>225</td>
<td>Burr Brown</td>
<td>PCM Audio DAC</td>
</tr>
</tbody>
</table>

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 \( \text{REF} = 2^{19} \) a square wave is produced with a 1:1 mark to space ratio, and a mean value of \( V_{\text{ref}}/2 \) appears after the filter along with unfiltered ripple.

![Figure 2.2 - Pulse-width-modulated DAC](image)
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 \( \text{REF}=1 \) 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 \( \text{msb}=0 \), \( \frac{V_{\text{ref}}}{2} \) and 0v are used, and when \( \text{msb}=1 \), \( V_{\text{ref}} \) and
\( \frac{V_{\text{ref}}}{2} \) 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 \( 0 \leq n < (2^N-1) \) and \( 0 \leq m < (2^M-1) \) and suppose

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

then average output = \( V_{\text{out}} \) is given by

\[
V_{\text{out}} = \frac{f(n) \cdot (2^M-m) + f(n+1) \cdot m}{2^M}
\]

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

then when

\[
m = 0 \quad V_{\text{out}} = \frac{n \cdot V_{\text{ref}}}{2^N}
\]

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

It can be seen that, for any \( n \), held constant, the output voltage varies linearly with \( m \) from the \( n \)th 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 \(1/2^{15}\) 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 suggests 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.

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.

![Diagram](image)

**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 (Vtol). This single modification (innovation) has the effect of making the whole design insensitive to errors and drifts in the performance of the Vtol, 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].

![Block diagram of drive](image)

**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 RSI, 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 RSI. However all these effects are deterministic and repeatable, so by measuring the voltage across the sensing terminals of RSI 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 be 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](image_url)

- 34 -
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 1/64th 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 320 kΩ represents an acceptable compromise, and results in a ratio of 1:64.

![Figure 3.2 - DAC(s) and summing amplifier](image)

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 ±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](image)

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 ±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](image)

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 $\pm 13.5$ 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 $\pm 1.2$ v.

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 than 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 $+5$v and $-5$v 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.

- 41 -
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 be 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 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 config 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.
<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Circuit Label</th>
<th>Port ID Label</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>C0</td>
<td>PA0</td>
<td>DBO (Data Bus LSB)</td>
</tr>
<tr>
<td>10</td>
<td>C1</td>
<td>PA1</td>
<td>DBO</td>
</tr>
<tr>
<td>9</td>
<td>C2</td>
<td>PA2</td>
<td>DB2</td>
</tr>
<tr>
<td>8</td>
<td>C3</td>
<td>PA3</td>
<td>DB3</td>
</tr>
<tr>
<td>7</td>
<td>C4</td>
<td>PA4</td>
<td>DB4</td>
</tr>
<tr>
<td>6</td>
<td>C5</td>
<td>PA5</td>
<td>DB5</td>
</tr>
<tr>
<td>5</td>
<td>C6</td>
<td>PA6</td>
<td>DB6</td>
</tr>
<tr>
<td>4</td>
<td>C7</td>
<td>PA7</td>
<td>DB7 (Data Bus MSB)</td>
</tr>
<tr>
<td>12</td>
<td>C8</td>
<td>PB0</td>
<td>NWRLB/A0</td>
</tr>
<tr>
<td>13</td>
<td>C9</td>
<td>PB1</td>
<td>NWRHB/A1</td>
</tr>
<tr>
<td>14</td>
<td>C10</td>
<td>PB2</td>
<td>NRD/NOE</td>
</tr>
<tr>
<td>15</td>
<td>C11</td>
<td>PB3</td>
<td>NWR</td>
</tr>
<tr>
<td>16</td>
<td>C12</td>
<td>PB4</td>
<td>NLDAC</td>
</tr>
<tr>
<td>17</td>
<td>C13</td>
<td>PB5</td>
<td>RESET of AD1175K</td>
</tr>
<tr>
<td>18</td>
<td>C14</td>
<td>PB6</td>
<td>50/60Hz select of AD1175K</td>
</tr>
<tr>
<td>19</td>
<td>C15</td>
<td>PB7</td>
<td>RTS auxiliary vs232 output</td>
</tr>
<tr>
<td>28</td>
<td>C16</td>
<td>PC0</td>
<td>not used</td>
</tr>
<tr>
<td>27</td>
<td>C17</td>
<td>PC1</td>
<td>-CS switch input select</td>
</tr>
<tr>
<td>26</td>
<td>C18</td>
<td>PC2</td>
<td>-CS AD1175 select</td>
</tr>
<tr>
<td>25</td>
<td>C19</td>
<td>PC3</td>
<td>-CS AD11458B Heater select</td>
</tr>
<tr>
<td>24</td>
<td>C20</td>
<td>PC4</td>
<td>-CS AD11458B Secondary DAC select</td>
</tr>
<tr>
<td>23</td>
<td>C21</td>
<td>PC5</td>
<td>-CS AD11458B Primary DAC select</td>
</tr>
<tr>
<td>22</td>
<td>C22</td>
<td>PC6</td>
<td>-CS MAX132CNG ADC select</td>
</tr>
<tr>
<td>21</td>
<td>C23</td>
<td>PC7</td>
<td>-CS RAM/Flash EEPROM select</td>
</tr>
<tr>
<td>29</td>
<td>C24</td>
<td>PDO</td>
<td>RS232 input</td>
</tr>
<tr>
<td>30</td>
<td>C25</td>
<td>PD1</td>
<td>RS232 output</td>
</tr>
<tr>
<td>31</td>
<td>C26</td>
<td>PD2</td>
<td>MOSI from MAX132CNG</td>
</tr>
<tr>
<td>32</td>
<td>C27</td>
<td>PD3</td>
<td>SCLK to MAX132CNG and Address Generator</td>
</tr>
<tr>
<td>33</td>
<td>C28</td>
<td>PD4</td>
<td>SCLK to MAX132CNG and Address Generator</td>
</tr>
<tr>
<td>34</td>
<td>C29</td>
<td>PD5</td>
<td>-SS tied to +5v</td>
</tr>
<tr>
<td>35</td>
<td>C31</td>
<td>PD7</td>
<td>Does not exist - preserves numbering</td>
</tr>
<tr>
<td>36</td>
<td>C32</td>
<td>TCAP</td>
<td>CTS auxiliary vs232 input</td>
</tr>
<tr>
<td>37</td>
<td>C33</td>
<td>TCP</td>
<td>output to LED via driver</td>
</tr>
<tr>
<td>2</td>
<td>C34</td>
<td>-IRQ</td>
<td>Interrupt tied to +5v</td>
</tr>
<tr>
<td>1</td>
<td>C35</td>
<td>-RESET</td>
<td>Reset from LM2925T regulator</td>
</tr>
</tbody>
</table>

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 \textit{samph} 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 \textit{STATE} counter which increments every timer interrupt (see \textit{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 $8 \times 0.0625$ 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 \textit{CHAN} counter, so that a new analogue channel is selected with the multiplexer U26.

The routine \textit{tims} also manages variable \textit{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 \textit{OLVL} bit in the Timer Control Register, and the values of \textit{BLSTAT}, \textit{BLCNT}, \textit{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. \textit{rdl} at 00461, and \textit{cmd} at 00491 are called in sequence near \textit{main1} at 00450, to read a line from the serial interface and then decode it. The line is terminated by \texttt{<CR>}. Use is made of a command table, \textit{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.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmdA</td>
<td>Synchronises ADC then outputs 8 numbers</td>
</tr>
<tr>
<td>cmdB</td>
<td>Repeated cmdA</td>
</tr>
<tr>
<td>cmdC</td>
<td>Repeatedly outputs PADC, averaged PADC</td>
</tr>
<tr>
<td>cmdD</td>
<td>Repeatedly outputs averaged PADC</td>
</tr>
<tr>
<td>cmdE</td>
<td>E</td>
</tr>
<tr>
<td>cmdF</td>
<td>F</td>
</tr>
<tr>
<td>cmdG</td>
<td>G</td>
</tr>
<tr>
<td>cmdH</td>
<td>Prepare Heater DAC</td>
</tr>
<tr>
<td>cmdI</td>
<td>Output a ramp to Heater DAC</td>
</tr>
<tr>
<td>cmdJ</td>
<td>J</td>
</tr>
<tr>
<td>cmdK</td>
<td>K</td>
</tr>
<tr>
<td>cmdL</td>
<td>Load all DACs - pulses -LDAC line</td>
</tr>
<tr>
<td>cmdM</td>
<td>Set MODE byte</td>
</tr>
<tr>
<td>cmdN</td>
<td>N</td>
</tr>
<tr>
<td>cmdO</td>
<td>Prepare Primary DAC</td>
</tr>
<tr>
<td>cmdP</td>
<td>Read precision ADC 20 times</td>
</tr>
<tr>
<td>cmdQ</td>
<td>Read precision ADC as 6+2 hex characters</td>
</tr>
<tr>
<td>cmdR</td>
<td>Read moving average of precision ADC in Hex</td>
</tr>
<tr>
<td>cmdS</td>
<td>Prepare Secondary DAC</td>
</tr>
<tr>
<td>cmdT</td>
<td>Enable/Disable software protection of EEPROM</td>
</tr>
<tr>
<td>cmdU</td>
<td>Read OIL switches as hex pair</td>
</tr>
<tr>
<td>cmdV</td>
<td>Read bytes from EEPROM</td>
</tr>
<tr>
<td>cmdW</td>
<td>Write a byte to EEPROM</td>
</tr>
<tr>
<td>cmdX</td>
<td>Read moving average of precision ADC in Hex</td>
</tr>
<tr>
<td>cmdY</td>
<td>Read all MAX132 channels in hex</td>
</tr>
<tr>
<td>cmdZ</td>
<td>Set some/all DACs</td>
</tr>
</tbody>
</table>

**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 *
00775 041A CD 042D cmdP jsr getpw get word parameter
00776 041D 24 00 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 98 cmdPh sei protect sequence
00805 0448 B7 00 sta PA output high byte to Primary DAC
00806 044A 13 01 bclr NWHRB,PB select high byte
00807 044C 10 01 bset NRRLB,PB but not low byte
00808 044E 18 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 87 00 sta PA output low byte to Primary DAC
00815 0457 11 01 bclr NRRLB,PB select low byte
00816 0459 12 01 bset NRHRB,PB but not high byte
00817 045B 18 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

\[ z2000400006000 \]

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 $4000000$, 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

\[ ($2000/64+$4000)\times64+400000 = 502000 \]

Which shows there to be an offset of

\[ ($502079-502000)/200000*5 = 0.000288V \approx 288\mu V \]

**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

\[ 0000BD00048A01359A01A78901D08601F1700000D5000D4Y \]
which is split as

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

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

```
0.00035 0.00214 0.14599 0.19971 0.23456 0.00039 0.00039
```

with interpretation after shifting and scaling of

```
0.0v 0.00179V 3.52454V 4.82466V 21.87°C 23.42°C
```

**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
<table>
<thead>
<tr>
<th>Code</th>
<th>Time</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td></td>
<td>☐ no operation, used to complete a byte</td>
</tr>
<tr>
<td>Group</td>
<td>1</td>
<td>nop</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>sldl</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>sstd</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>ld1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>stl</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>ld2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>add</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>sub</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>mul</td>
</tr>
<tr>
<td>A</td>
<td>1850</td>
<td>div</td>
</tr>
<tr>
<td>B</td>
<td>br</td>
<td>branch group</td>
</tr>
<tr>
<td>C</td>
<td>fun</td>
<td>function group</td>
</tr>
<tr>
<td>D</td>
<td>cnt</td>
<td>load/dec and branch counter group</td>
</tr>
<tr>
<td>E</td>
<td>adc</td>
<td>load FP1 with ADC conversion; low-nibble=channel</td>
</tr>
<tr>
<td>F</td>
<td>undefined</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Branch group</th>
<th>0</th>
<th>bra</th>
<th>branch always</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>beq</td>
<td>branch if FP1=0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>bne</td>
<td>branch if FP1&lt;&gt;0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>bpl</td>
<td>branch if FP1 sign bit = 0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>bmi</td>
<td>branch if FP1 sign bit = 1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>call</td>
<td>reserved</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>ret</td>
<td>reserved</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>exit</td>
<td>leave interpreter</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>wt</td>
<td>wait if keyboard activity</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>sym</td>
<td>synchronise with CHAN counter</td>
</tr>
<tr>
<td>A</td>
<td>bc</td>
<td>branch if bits clear</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>bs</td>
<td>branch if bits set</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>dly</td>
<td>delay in 0.1 sec intervals</td>
<td></td>
</tr>
<tr>
<td>D,E,F</td>
<td>undefined</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function group</th>
<th>0</th>
<th>clr</th>
<th>FP1=0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>abs</td>
<td>FP1=abs(FP1)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>neg</td>
<td>FP1=-FP1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>sgn</td>
<td>FP1=-1,0,+1 matching sign of FP1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4800</td>
<td>FP1=square root(FP1) [4.6] - uses FPO,1,2,3</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>sqn</td>
<td>FP1=FP1*FP2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>swp</td>
<td>FP1&lt;&gt;FP2</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>fix</td>
<td>fix FP1</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>flt</td>
<td>float FP1</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>int</td>
<td>find integer nearer zero</td>
</tr>
<tr>
<td>A</td>
<td>frac</td>
<td>remove integer part</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>out</td>
<td>output FP1 in fixed point format to RS232</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>crlf</td>
<td>output CRLF to RS232</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>spac</td>
<td>output a space to RS232</td>
<td></td>
</tr>
<tr>
<td>E,F</td>
<td>undefined</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 the reading of monitor ADC channel 2, the voltage setting for the heater drive is given by:

\[
\text{Heater voltage} = K_1 \sqrt{K_2 - \text{ADC}^2}^2
\]

\(K_1\) 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 1dx #L/256 pointer high
00609 032F A6 3D 1da #L%256 pointer low
00610 0331 CD 06BD jsr fintp interpret
...
0061x 0334 1E B9 bset ACK,MODE acknowledge
0061x 0336 B1 rts

* Formula = K1 * SQRT(K2-(ADC2)^2)
00619 0330 B9 L fc fbr,fsyn
00620 033E E2 fc fadc,2
00621 033F C5 fc ffun,fsg
00622 0340 C2 fc ffun,fneg
00623 0341 67 fc f1d2,fadd FP2=K2, then FP1=FP1+K2
00624 0342 82 fcb $82
00625 0343 C4 fc ffun,fsgt FP2*K1, then FP1=FP1*K1
00626 0344 69 fc f1d2,fmul by iteration, see [4.7]
00627 0345 B1 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
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 control_loop;
1587 inc(tries);
1588 { gotoxy(10,21); write(loop_count:6,reply_count:6,tries:8); }
1589 report current time;
1590 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 comer 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.
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 1/64th 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.
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.

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.

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)

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.

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.

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.

```
<table>
<thead>
<tr>
<th>Source=CALIBRATION</th>
<th>DRIVE 1</th>
<th>DRIVE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Floating</td>
<td>Floating</td>
</tr>
<tr>
<td>Set Point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precision</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean Dist</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyboard</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secondary</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heater Power</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amplifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amplifier 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature in °C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature out °C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean Loop Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abort Calibrate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keyboard</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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 cclv15f.dt obtained during
testing of the calibration techniques.

<table>
<thead>
<tr>
<th>Drive 1</th>
<th>Drive 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coil Resistance = 22.000</td>
<td>Coil Resistance = 22.000</td>
</tr>
<tr>
<td>Heater Resistance = 22.000</td>
<td>Heater Resistance = 22.000</td>
</tr>
<tr>
<td>Primary DAC statistics</td>
<td>Primary DAC statistics</td>
</tr>
<tr>
<td>Temperature (in) = 23.9600</td>
<td>Temperature (in) = 24.7569</td>
</tr>
<tr>
<td>Number = 127</td>
<td>Number = 127</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index Code</th>
<th>Voltage</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-32256.0</td>
<td>-4.92171597</td>
</tr>
<tr>
<td>2</td>
<td>-31744.0</td>
<td>-4.84356380</td>
</tr>
<tr>
<td>63</td>
<td>-512.0</td>
<td>-0.07792139</td>
</tr>
<tr>
<td>64</td>
<td>0.0</td>
<td>0.00018167</td>
</tr>
<tr>
<td>127</td>
<td>32256.0</td>
<td>4.92212224</td>
</tr>
</tbody>
</table>

mean x = 0.0 mean y = 0.000025976

a = 2.5975610327533E-004
b = 1.52586576621978E-004
r = 9.999999924512E-001

Secondary DAC statistics
Temperature (in) = 24.1393
Number = 63

<table>
<thead>
<tr>
<th>Index Code</th>
<th>Voltage</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-31744.0</td>
<td>-0.07546997</td>
</tr>
<tr>
<td>2</td>
<td>-30720.0</td>
<td>-0.07303190</td>
</tr>
<tr>
<td>31</td>
<td>-1024.0</td>
<td>-0.00226164</td>
</tr>
<tr>
<td>32</td>
<td>0.0</td>
<td>0.00018215</td>
</tr>
<tr>
<td>63</td>
<td>31744.0</td>
<td>0.07679677</td>
</tr>
</tbody>
</table>

mean x = 0.0 mean y = 0.00107441

a = 2.06016358875072E-004
b = 2.38446421095551E-006
r = 9.9999992838470E-001

Secondary DAC statistics
Temperature (in) = 24.8809
Number = 63

<table>
<thead>
<tr>
<th>Index Code</th>
<th>Voltage</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-31744.0</td>
<td>-0.07461419</td>
</tr>
<tr>
<td>2</td>
<td>-30720.0</td>
<td>-0.07218027</td>
</tr>
<tr>
<td>31</td>
<td>-1024.0</td>
<td>-0.00139141</td>
</tr>
<tr>
<td>32</td>
<td>0.0</td>
<td>0.00105715</td>
</tr>
<tr>
<td>63</td>
<td>31744.0</td>
<td>0.07679677</td>
</tr>
</tbody>
</table>

mean x = 0.0 mean y = 0.0000107441

a = 1.07441062018943E-003
b = 2.38446859352594E-006
r = 9.99999937323992E-001

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,
begin {calibrate_cycle}
  for i := 1 to max_drive do
    with drive[i].cal do
      case state of
        cal_idle
          begin
            set_stats(p_stats);
            state := cal_p;
          end;
        cal_start
          begin
            define_setpoint(drive[i],2);
            drive[i].s_trim := 0;
            if i=1 then
              display_finish_time;
            end;
            cal_p
              begin
                if drive[i].padc_filter.valid then
                  begin
                    enter_stats(p_stats,drive[i].p_code,
                      drive[i].padc_filter.mean);
                    if p_stats.n=p_limit/2 then { record in middle of run }
                      p_stats.temperature_in := drive[i].fp_temperature_in;
                    end;
                  end;
                end;
              end;
        cal_s
          begin
            if drive[i].padc_filter.valid then
              begin
                enter_stats(s_stats,drive[i].s_code,
                  drive[i].padc_filter.mean);
                if s_stats.n=s_limit/2 then { record in middle of run }
                  s_stats.temperature_in := drive[i].fp_temperature_in;
                end;
              end;
          end;
        cal_finished
          begin
            apply_stats(s_stats);
            state := cal_finished;
            with drive[i] do
              begin
                drive[i].fp_setpoint := 0;
                set_filter(padc_filter,fp_setpoint,
                  precision_adc_filter_length,3);
              end;
          end;
      end;
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_{pdc} = a + b \times \text{DAC}_p, where \text{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.

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 $V_{pdc} = a + b \times \text{DAC}_s$, where \text{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.

Figure 5.6 - Combined DACs during calibration
and scaling which is close to the theoretical value of \( \frac{5}{32768/64} = 2.384185 \times 10^{-6} \).

The error was attributed to the tolerances of resistors R10 and R11. Coefficients \( a \) and \( b \) are stored and used in the form \( DAC_s = \frac{\text{Offset\_voltage}-a}{b} \) by the closed-loop tracking algorithm in order to compute the control code for the secondary DAC.

When both drives enter the \texttt{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 \texttt{read(f,ch)}, where \( f \) has been previously assigned to a communications port via \texttt{assignDD} at line 635 in the procedure \texttt{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

<table>
<thead>
<tr>
<th>Channel</th>
<th>PC I/O</th>
<th>PC IRQ</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM1</td>
<td>$3F8</td>
<td>4</td>
<td>Drive 1</td>
</tr>
<tr>
<td>COM2</td>
<td>$2F8</td>
<td>3</td>
<td>Drive 2</td>
</tr>
<tr>
<td>COM3</td>
<td>$3E8</td>
<td>5</td>
<td>Host PC</td>
</tr>
</tbody>
</table>

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 ±5μV 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

<table>
<thead>
<tr>
<th>Component Number</th>
<th>Component Type</th>
<th>Noise $\mu$V pk-pk</th>
<th>Bandwidth Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td>REF1</td>
<td>AD586LQ</td>
<td>4</td>
<td>0.1 to 10</td>
</tr>
<tr>
<td>U6,U7</td>
<td>AD1145BG</td>
<td>5</td>
<td>0.1 to 10</td>
</tr>
<tr>
<td>U8,U10</td>
<td>TLC2654ACN</td>
<td>1.5</td>
<td>0 to 10</td>
</tr>
<tr>
<td>R9,R11</td>
<td>5k ohms</td>
<td>0.45</td>
<td>0 to 100</td>
</tr>
<tr>
<td>R10</td>
<td>320k</td>
<td>3.6</td>
<td>0 to 100</td>
</tr>
<tr>
<td>R17,R18</td>
<td>10k ohms</td>
<td>0.64</td>
<td>0 to 100</td>
</tr>
</tbody>
</table>

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]
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 $R_{11}$ to $R_{10}$ (1:64), as is the contribution from $R_{10}$ 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](image-url)
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 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 \( \frac{1}{70} = 14.2857 \) 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 \( \mu \)A (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 \( \mu \)A 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 - vto11 Frequency response](image)

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
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.
<table>
<thead>
<tr>
<th>Frequency Hz</th>
<th>rms (nV)</th>
<th>pk to pk (μV) = 5xrms @ 99% confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>from 0.01</td>
<td>to 0.10</td>
<td>946.4</td>
</tr>
<tr>
<td>from 0.10</td>
<td>to 1.00</td>
<td>700.0</td>
</tr>
<tr>
<td>from 1.00</td>
<td>to 10.00</td>
<td>517.8</td>
</tr>
<tr>
<td>from 10.00</td>
<td>to 100.00</td>
<td>383.0</td>
</tr>
<tr>
<td>from 100.00</td>
<td>to 1000.00</td>
<td>283.3</td>
</tr>
</tbody>
</table>

Figure 6.6 - Noise of OPA654

Between 1 kHz and 10 kHz white noise is dominant and the rms noise is given by \(12 \times 10^{-9} \times (10^4 - 10^0)^{0.5} = 1.14\mu V\), 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 ±2.5 ppm, with a nominal of 0 ppm! Assuming the drives are operated in a precision metrology laboratory with a temperature stability of ±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 $(2^{18}) = 262144$, 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.
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](image)

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 $4\times80/70=4.57v$ 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.

![Simulated and actual step responses](image)

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 1 kHz 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 10 Ω 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.
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.

\[ R \text{ is a } 10 \Omega \text{ resistor in series with the coil.} \]

Let \( V_R = \) voltage across \( R \), \( V_a = \) amplified voltage, 
\( g = \) gain = 469000, \( I_c = \) coil current

\[
V_R = I_c R \quad : \quad I_c = \frac{V_R}{10g} \quad \text{also} \quad +I_{max} = \frac{5}{70} \quad \Rightarrow \quad I_{range} = 2 \times \frac{5}{70}
\]

\[
\text{fraction} = \frac{I_c}{I_{range}} = \frac{V_a}{10g} \times \frac{5}{70} \times \frac{70}{5} = 0.7 \frac{V_a}{g}
\]

\[
\text{ppm} = 0.7 \times \frac{10^6}{469000} \times V_a
\]
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

![Graph A: Intrinsic amplifier noise](image)

![Graph B: SP=4.0 and fan](image)

![Graph C: Drive 1 OFF](image)

![Graph D: Drive 1 ON, SP=0.0](image)

**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.

```matlab
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 100us samples
z=[f',s./m/(n/2)];

file噪声4.m
% Load files and compute spectra
load pcdnil; a=spec3(pcdnil); clear pcdnil; % amplifier noise
load pcdn16; b=spec3(pcdn16); clear pcdn16; % mains off
load pcdn17; c=spec3(pcdn17); clear pcdn17; % SP=0
load pcdn11; d=spec3(pcdn11); clear pcdn11; % SP=4 and lead near %
% 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 %
% Plot
clf;
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'); ylabel('ppm FSO'); title('2: SP=4.0 and fan');
plot(f,d,'w-');
```

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.
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.

![Diagram of error ppm, Trim count, Prim. DAC bit 9, Increment Interval](image)

**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 \(<111>\) 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 ± 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.

![Graph showing noise spike on fringe pattern](image)

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

<table>
<thead>
<tr>
<th>Offset/V</th>
<th>Settling/s</th>
<th>Gate/s</th>
<th>Step size</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td>4.50</td>
<td>1.0</td>
<td>1.0</td>
<td>30μV Ramp and return of 100 steps</td>
</tr>
<tr>
<td>Rear</td>
<td>4.45</td>
<td></td>
<td></td>
<td>(~20pm)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Offset/V</th>
<th>Settling/s</th>
<th>Gate/s</th>
<th>Step size</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front</td>
<td>2.02</td>
<td>5.0</td>
<td>0.5</td>
<td>2μV  Very slow scan of 900 steps</td>
</tr>
<tr>
<td>Rear</td>
<td>0.78</td>
<td></td>
<td></td>
<td>(~0.4pm) in 85 minutes</td>
</tr>
</tbody>
</table>
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.
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OPA177E</strong></td>
<td></td>
</tr>
<tr>
<td>Voltage offset</td>
<td>$4 \mu V$ (typ) $10 \mu V$ (max) at $25^\circ C$</td>
</tr>
<tr>
<td>Voltage offset drift</td>
<td>$0.03 \mu V/^\circ C$ (typ), $0.1 \mu V/^\circ C$ (max)</td>
</tr>
<tr>
<td>Open loop gain</td>
<td>134 dB (min)</td>
</tr>
<tr>
<td>Input noise voltage</td>
<td>85 nV rms (typ), 1 Hz to 100 Hz</td>
</tr>
<tr>
<td>Long term drift</td>
<td>0.2 $\mu V$/month</td>
</tr>
<tr>
<td>Input bias current</td>
<td>0.5 nA (typ)</td>
</tr>
<tr>
<td>bias current drift</td>
<td>25 pA/$^\circ C$ (max)</td>
</tr>
<tr>
<td>Supply current</td>
<td>1.3 mA at ±15 V</td>
</tr>
<tr>
<td><strong>OPA627BM</strong></td>
<td></td>
</tr>
<tr>
<td>Voltage offset</td>
<td>$40 \mu V$ (typ) $100 \mu V$ (max) at $25^\circ C$</td>
</tr>
<tr>
<td>Voltage offset drift</td>
<td>$0.4 \mu V/^\circ C$ (typ), $0.8 \mu V/^\circ C$ (max)</td>
</tr>
<tr>
<td>Open loop gain</td>
<td>112 dB (min)</td>
</tr>
<tr>
<td>Input noise voltage</td>
<td>0.6 $\mu V$-p (typ), 0.1 Hz to 10 Hz</td>
</tr>
<tr>
<td>Long term drift</td>
<td>not specified</td>
</tr>
<tr>
<td>Input bias current</td>
<td>1 pA (typ) $5$ pA (max) at $25^\circ C$</td>
</tr>
<tr>
<td>bias current drift</td>
<td>doubles every $10^\circ C$</td>
</tr>
<tr>
<td>Supply current</td>
<td>±7 mA at ±15 V</td>
</tr>
</tbody>
</table>

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 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 recalibration 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](image-url)
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, $V_n = V_{dec}$. Since $V_n = \text{voltage across sensing pins of } R_{ref}$ and the coil and $R_{ref}$ are in series, the current in the coil is proportional to $V_{dec}$. C reduces the high frequency response and provides closed-loop stability.

![Diagram](image)

**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 create 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].

![Circuit Diagram](attachment:image.png)

**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 $\mu$F. 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 $(1.5-2.5)/10 = -20$ 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 $C_{in} = 0.33 \mu$F 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](image)

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 \( C_s\)1, and code \( n \) produces level \( N \) and is associated with \( C_s\)2. In the case of an interval of type-\( m \),
Cs1 is briefly subject to the previous level N, then a glitch, before settling to the desired level M. The value of Cs1 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', Cs1 will not follow immediately, and many samples will be necessary before charge redistribution is completed. While the charge on Cs1 is in flux the voltage on Cs1 need not settle to the expected level by the end of its sampling interval. The behaviour of Cs2 and Ch2 is similar but Cs2 acquires level N by the end of type-n intervals.

The LTC1043 contains two sampling circuits that operate in anti-phase and this leads directly to the implementation in Figure 7.8(b), 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 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.
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 $2^{10} \times 10^6 = 1.024 \text{ ms}$, 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. $V_{\text{ref}}$ 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

* indicates function generated with/by FPGA, which may be optically coupled to embedded microcontroller.
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 0.1 μF 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\textsubscript{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\textsubscript{ref}/2 may be derived from V\textsubscript{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
±50 mA at 12.5 V, or ±100 mA at ±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 $R_{ref} = 5/2/0.05 = 500$. Assuming the end effects of $R_{ref}$ are small compared with its value the maximum coil resistance is $(12.5 - 2.5)/0.05 = 200 \, \Omega$. 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 \( \mu \)s 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.

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 \( \mu \)s) 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 \( V_o \) is used to drive the voltage-to-current converter.

![Figure 8.1 - Circuit depicting DACs with different time constants](image-url)
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(ers) 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](image)

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](image)

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


- 121 -


[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]


[2.6] Pages 130 and 133 of [6.2]

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


[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"


[3.2] See 4.2

[3.3] Analog Devices: "AD1145BG DAC 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"


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


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


[6.7] See page 34 of [5.2]

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


[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.


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


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


[8.1] Bede scientific instruments: "XIP control program", Linsey Park, Bowburn, Durham, England DH6 5PF
<table>
<thead>
<tr>
<th>#</th>
<th>Ref</th>
<th>Part</th>
<th>Supplier</th>
<th>Function</th>
<th>From DRIMIQIA.CAL</th>
<th>No. File.sch</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADC1</td>
<td>AD1175K</td>
<td>Analog Devices</td>
<td>Feedback ADC</td>
<td>CPU and ADC</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>CK1</td>
<td>2pF</td>
<td>RS 126-124</td>
<td>CPU oscillator</td>
<td>CPU and ADC</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>CK2</td>
<td>2pF</td>
<td>RS 126-124</td>
<td>CPU oscillator</td>
<td>CPU and ADC</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>C1</td>
<td>10000uF/40v</td>
<td>RS 104-382</td>
<td>+15vADC PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>C2</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>+15vADC PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>C3</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>+15vADC PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>C4</td>
<td>10000uF/40v</td>
<td>RS 104-382</td>
<td>-15vADC PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>C5</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>-15vADC PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>C6</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>-15vADC PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>C7</td>
<td>10000uF/40v</td>
<td>RS 104-382</td>
<td>+15vDRV PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>C8</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>+15vDRV PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>C9</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>+15vDRV PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>C10</td>
<td>10000uF/40v</td>
<td>RS 104-382</td>
<td>-15vDRV PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>C11</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>-15vDRV PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>C12</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>-15vDRV PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>C13</td>
<td>15000/16v</td>
<td>RS 104-380</td>
<td>+5vDIG PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>C14</td>
<td>15000/16v</td>
<td>RS 104-380</td>
<td>+5vDIG PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>C15</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>+5vDIG PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>C16</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>+5vDIG PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>C17</td>
<td>15000/16v</td>
<td>RS 104-380</td>
<td>FAN PSU</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>C18</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>+5v bypass</td>
<td>CPU and ADC</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>C19</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>+5v bypass</td>
<td>CPU and ADC</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>C19A</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>+5v bypass</td>
<td>RS232 Chip</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>C20</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>+5v bypass</td>
<td>RS232 Chip</td>
<td>3</td>
</tr>
<tr>
<td>25</td>
<td>C20A</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>+5v bypass</td>
<td>RS232 Chip</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>C21</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U8 chopper</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>27</td>
<td>C22</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U8 chopper</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>28</td>
<td>C23</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U10 chopper</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>29</td>
<td>C24</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U10 chopper</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>C25</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U9 chopper</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>31</td>
<td>C26</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U9 chopper</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>C27A</td>
<td>1uF</td>
<td>RS 126-067</td>
<td>REF1 filter</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>33</td>
<td>C27B</td>
<td>1uF</td>
<td>RS 126-067</td>
<td>REF1 filter</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>34</td>
<td>C28</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>VDD+ bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>35</td>
<td>C29</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>VDD+ bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>36</td>
<td>C30</td>
<td>10nF</td>
<td>RS 116-812</td>
<td>U12 phase comp.</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>37</td>
<td>C31</td>
<td>100pF</td>
<td>RS 126-922</td>
<td>HF filter</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>38</td>
<td>C32</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U12 chopper</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>39</td>
<td>C33</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U12 chopper</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>C34</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>Low pass filter</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>41</td>
<td>C35</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>Low pass filter</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>42</td>
<td>C36</td>
<td>0.1uF</td>
<td>RS 116-840</td>
<td>Filter</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>43</td>
<td>C37</td>
<td>0.1uF</td>
<td>RS 116-840</td>
<td>Filter</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>44</td>
<td>C38</td>
<td>4.7uF/16v</td>
<td>FN 100-978</td>
<td>U13 +bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>45</td>
<td>C39</td>
<td>10pF</td>
<td>RS 126-809</td>
<td>U13 comp.</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>46</td>
<td>C40</td>
<td>4.7uF</td>
<td>FN 100-978</td>
<td>U13 -bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>47</td>
<td>C41</td>
<td>10nF</td>
<td>RS 116-812</td>
<td>HF filter</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>48</td>
<td>C42</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>+5v bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>49</td>
<td>C43</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>+5v bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>C44</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>+5v bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>51</td>
<td>C45</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>-15vDRV bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>52</td>
<td>C46</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>-15vDRV bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td>C47</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>+15vDRV bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>54</td>
<td>C48</td>
<td>47uF/16v</td>
<td>FN 100-884</td>
<td>+15vDRV bypass</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>55</td>
<td>C49</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U15 chopper</td>
<td>Heater/Drive2</td>
<td>5</td>
</tr>
<tr>
<td>56</td>
<td>C50</td>
<td>0.47uF</td>
<td>RS 116-862</td>
<td>U15 chopper</td>
<td>Heater/Drive2</td>
<td>5</td>
</tr>
<tr>
<td>57</td>
<td>C51</td>
<td>0.1uF</td>
<td>RS 125-733</td>
<td>U18 bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
</tr>
<tr>
<td>58</td>
<td>C52</td>
<td>10uF</td>
<td>RS 101-816</td>
<td>U18 bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
</tr>
<tr>
<td>59</td>
<td>C53A</td>
<td>1uF</td>
<td>RS 126-067</td>
<td>REF2 filter</td>
<td>Heater/Drive2</td>
<td>5</td>
</tr>
<tr>
<td>60</td>
<td>C53B</td>
<td>1uF</td>
<td>RS 126-067</td>
<td>REF2 filter</td>
<td>Heater/Drive2</td>
<td>5</td>
</tr>
<tr>
<td>61</td>
<td>C54</td>
<td>10uF</td>
<td>RS 116-812</td>
<td>Phase comp.</td>
<td>Heater/Drive2</td>
<td>5</td>
</tr>
<tr>
<td>Part</td>
<td>Supplier</td>
<td>Function</td>
<td>Sheet Name</td>
<td>No. File</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>----------</td>
<td>------------</td>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>RS 126-922</td>
<td>HF filter</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>RS 114-862</td>
<td>U12 chopper</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>RS 114-862</td>
<td>U12 chopper</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>RS 114-862</td>
<td>Low pass filter</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>RS 114-862</td>
<td>Low pass filter</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>RS 114-840</td>
<td>Filter</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>RS 114-840</td>
<td>Filter</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>FN 100-878</td>
<td>U13 +bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>RS 126-809</td>
<td>U13 comp.</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>FN 100-878</td>
<td>U13 +bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>RS 114-812</td>
<td>HF filter</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>RS 125-733</td>
<td>+5V bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>RS 125-733</td>
<td>+5V bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>FN 100-884</td>
<td>+15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>FN 100-884</td>
<td>-15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>RS 125-733</td>
<td>-15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>RS 125-733</td>
<td>-15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>RS 125-733</td>
<td>+15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>RS 125-733</td>
<td>-15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>RS 125-733</td>
<td>+15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>RS 125-733</td>
<td>-15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>RS 125-733</td>
<td>+15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>RS 125-733</td>
<td>-15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>RS 125-733</td>
<td>+15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>RS 125-733</td>
<td>+15VOR bypass</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>RS 113-285</td>
<td>U13 freq. comp.</td>
<td>Drive 1</td>
<td>4</td>
<td>DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>RS 113-285</td>
<td>U17 freq. comp.</td>
<td>Drive 1</td>
<td>4</td>
<td>DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>Linear Technology</td>
<td>Bandgap Ref.</td>
<td>Auxiliary Devices</td>
<td>6</td>
<td>DRAUX1</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>RS 108-269</td>
<td>Mains input</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>RS 108-269</td>
<td>Mains switch</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>2 pin 0.1&quot; pitch</td>
<td>Fan connector</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>2 pin 0.1&quot; pitch</td>
<td>LED connector</td>
<td>CPU and ADC</td>
<td>3</td>
<td>DRICPU1</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>7 pin 0.1&quot; pitch</td>
<td>Serial link</td>
<td>CPU and ADC</td>
<td>3</td>
<td>DRICPU1</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>3 pin 0.1&quot; pitch</td>
<td>Drive 1</td>
<td>Drive 1</td>
<td>4</td>
<td>DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>3 pin 0.1&quot; pitch</td>
<td>Drive 2</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>no component</td>
<td>Star point</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>no component</td>
<td>Star point</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>FN 108-269</td>
<td>U13 HF suppression</td>
<td>Drive 1</td>
<td>4</td>
<td>DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>FN 108-269</td>
<td>U17 HF suppression</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>RS 296-043</td>
<td>LED driver</td>
<td>CPU and ADC</td>
<td>3</td>
<td>DRICPU1</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>RS 294-457</td>
<td>VDD+ booster</td>
<td>Drive 1</td>
<td>4</td>
<td>DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>RS 294-453</td>
<td>VDD- booster</td>
<td>Drive 1</td>
<td>4</td>
<td>DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>FN 1KA820E</td>
<td>ADC rectifiers</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>FN 1KA820E</td>
<td>DRV rectifiers</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>FN 1KA820E</td>
<td>DIG rectifiers</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>FN 1KA820E</td>
<td>FAN rectifiers</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>Analog Devices</td>
<td>Drive1 Vref</td>
<td>Drive 1</td>
<td>4</td>
<td>DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>Analog Devices</td>
<td>Drive2 Vref</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>RS 648-444</td>
<td>+15VADC regulator</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>RS 648-472</td>
<td>-15VADC regulator</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>RS 648-444</td>
<td>+15VOR regulator</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>RS 648-472</td>
<td>-15VOR regulator</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>RS 648-545</td>
<td>+5VDIG regulator</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>FN 2990ACZS</td>
<td>U22 +5V regulator</td>
<td>Auxiliary Devices</td>
<td>6</td>
<td>DRAUX1</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>FN 79L0SACZ</td>
<td>U22 -5V regulator</td>
<td>Auxiliary Devices</td>
<td>6</td>
<td>DRAUX1</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>70ohms</td>
<td>N/A</td>
<td>External Coil</td>
<td>Drive 1</td>
<td>4</td>
<td>DR1DRV1</td>
</tr>
<tr>
<td>118</td>
<td>70ohms</td>
<td>N/A</td>
<td>External Coil</td>
<td>Heater/Drive2</td>
<td>5</td>
<td>DR1DRV2</td>
</tr>
<tr>
<td>119</td>
<td>RS 135-645</td>
<td>CPU oscillator</td>
<td>CPU and ADC</td>
<td>3</td>
<td>DRICPU1</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>RS 148-663</td>
<td>+15VADC load</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>RS 148-663</td>
<td>-15VADC load</td>
<td>Power Supplies</td>
<td>2</td>
<td>DRIPSU1</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>Ref Part</td>
<td>Supplier</td>
<td>Function</td>
<td>From DR1M01A.CAL</td>
<td>No. File.sch</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>----------</td>
<td>--------------</td>
<td>-----------------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>R3</td>
<td>4k7</td>
<td>RS 148-663</td>
<td>Power Supplies</td>
<td>2 DR1PSU1</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>R4</td>
<td>4k7</td>
<td>RS 148-663</td>
<td>Power Supplies</td>
<td>2 DR1PSU1</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>R5</td>
<td>10k</td>
<td>RS 148-736</td>
<td>Power Supplies</td>
<td>2 DR1PSU1</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>R6</td>
<td>4k7</td>
<td>RS 148-663</td>
<td>Power Supplies</td>
<td>2 DR1PSU1</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>R7</td>
<td>150k</td>
<td>RS 148-304</td>
<td>CPU and ADC</td>
<td>3 DRCPU1</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>R8</td>
<td>47k5IL</td>
<td>RS 140-990</td>
<td>CPU and ADC</td>
<td>3 DRCPU1</td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>RBA</td>
<td>47k5IL</td>
<td>RS 140-990</td>
<td>CPU and ADC</td>
<td>3 DRCPU1</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>R9</td>
<td>5k0</td>
<td>Vishay SI02K</td>
<td>Drive 1</td>
<td>4 DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>R10</td>
<td>320k</td>
<td>Vishay SI02K</td>
<td>Drive 1</td>
<td>4 DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>R11</td>
<td>5k0</td>
<td>Vishay SI02K</td>
<td>Drive 1</td>
<td>4 DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>R12</td>
<td>20k</td>
<td>RS 167-018</td>
<td>Voltage set</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>R13</td>
<td>11k8</td>
<td>RS 166-900</td>
<td>Voltage set</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>R14</td>
<td>20k</td>
<td>RS 167-018</td>
<td>Voltage set</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>R15</td>
<td>20k</td>
<td>RS 167-018</td>
<td>Voltage set</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>R16</td>
<td>150k</td>
<td>RS 149-026</td>
<td>U12 Phase comp.</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>R17</td>
<td>10k</td>
<td>RS 148-736</td>
<td>Low pass filter</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>R18</td>
<td>10k</td>
<td>RS 148-736</td>
<td>Low pass filter</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>R19</td>
<td>10k</td>
<td>RS 148-736</td>
<td>DC feedback</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>R20</td>
<td>10k</td>
<td>RS 148-736</td>
<td>DC feedback</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>131</td>
<td>R21</td>
<td>10k</td>
<td>RS 148-736</td>
<td>Filter</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>R22</td>
<td>10k</td>
<td>RS 148-736</td>
<td>Filter</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>R23</td>
<td>10</td>
<td>RS 148-017</td>
<td>Current limit</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>R24</td>
<td>10</td>
<td>RS 148-017</td>
<td>Current limit</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>R25</td>
<td>82k</td>
<td>RS 148-950</td>
<td>Voltage set</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>R26</td>
<td>180k</td>
<td>RS 149-048</td>
<td>Voltage set</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>R27</td>
<td>150k</td>
<td>RS 149-026</td>
<td>U12 Phase comp.</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>R28</td>
<td>10k</td>
<td>RS 148-736</td>
<td>Low pass filter</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>R29</td>
<td>10k</td>
<td>RS 148-736</td>
<td>Low pass filter</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>R30</td>
<td>10k</td>
<td>RS 148-736</td>
<td>DC feedback</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>R31</td>
<td>10k</td>
<td>RS 148-736</td>
<td>DC feedback</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>R32</td>
<td>10k</td>
<td>RS 148-736</td>
<td>Filter</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>R33</td>
<td>10k</td>
<td>RS 148-736</td>
<td>Filter</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>R34</td>
<td>10</td>
<td>RS 148-017</td>
<td>Current limit</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>R35</td>
<td>10</td>
<td>RS 148-017</td>
<td>Current limit</td>
<td>5 DR1DRV2</td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>R36</td>
<td>23k2</td>
<td>RS 167-074</td>
<td>Voltage divider</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>R37</td>
<td>1k</td>
<td>RS 165-769</td>
<td>Voltage divider</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>R38</td>
<td>23k2</td>
<td>RS 167-074</td>
<td>Voltage divider</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>R39</td>
<td>1k</td>
<td>RS 165-769</td>
<td>Voltage divider</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>R40</td>
<td>4k7</td>
<td>RS 148-663</td>
<td>U22 5v load</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>R41</td>
<td>47k5IL</td>
<td>RS 140-990</td>
<td>U19 pullup</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>R42</td>
<td>620k</td>
<td>RS 149-177</td>
<td>U22 Int. R</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>R43</td>
<td>33k2 0.1%</td>
<td>RS 167-226</td>
<td>Vref divider</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>R44</td>
<td>33k2 0.1%</td>
<td>RS 167-226</td>
<td>Vref divider</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>R45</td>
<td>33k</td>
<td>RS 148-859</td>
<td>D3 bias</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>R46</td>
<td>27k</td>
<td>RS 148-837</td>
<td>Q1 gate pulldown</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>SW1</td>
<td>SMD1P</td>
<td>RS 337-560</td>
<td>Setup switches</td>
<td>6 DRAUX1</td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>T1</td>
<td>5V 0-15 0-15</td>
<td>RS 207-807</td>
<td>ADC PSU</td>
<td>2 DR1PSU1</td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>T2</td>
<td>5V 0-15 0-15</td>
<td>RS 207-807</td>
<td>Drive PSU</td>
<td>2 DR1PSU1</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>T3</td>
<td>5V 0-15 0-15</td>
<td>RS 207-970</td>
<td>Digital/Fan PSU</td>
<td>2 DR1PSU1</td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>U1</td>
<td>MC68HC705C8S</td>
<td>Motorola</td>
<td>Microprocessor</td>
<td>3 DRCPU1</td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>U2</td>
<td>74HC164</td>
<td>RS 301-347</td>
<td>Address Generator</td>
<td>3 DRCPU1</td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>U3</td>
<td>74HC164</td>
<td>RS 301-347</td>
<td>Address Generator</td>
<td>3 DRCPU1</td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>U4</td>
<td>X28C256</td>
<td>RS 656-041</td>
<td>Memory</td>
<td>3 DRCPU1</td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>U5</td>
<td>MAX252ACHL</td>
<td>Maxim</td>
<td>Isolated RS232</td>
<td>3 DRCPU1</td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>U6</td>
<td>ADI1458G</td>
<td>Analog Devices</td>
<td>Primary DAC</td>
<td>4 DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>167</td>
<td>U7</td>
<td>ADI1458G</td>
<td>Analog Devices</td>
<td>Secondary DAC</td>
<td>4 DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>168</td>
<td>U8</td>
<td>TLC2654CN</td>
<td>FN TLC2654CN</td>
<td>U6 output</td>
<td>4 DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>169</td>
<td>U9</td>
<td>TLC2654CN</td>
<td>FN TLC2654CN</td>
<td>U7 output</td>
<td>4 DR1DRV1</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>Ref</td>
<td>Part</td>
<td>Supplier</td>
<td>Function</td>
<td>Sheetname</td>
<td>No.</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>------</td>
<td>----------</td>
<td>----------</td>
<td>-----------</td>
<td>-----</td>
</tr>
<tr>
<td>180</td>
<td>U10</td>
<td>TLC2654CN</td>
<td>FN TLC2654CN</td>
<td>Summing amplifier</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>181</td>
<td>U11A</td>
<td>TLE2062ACP</td>
<td>RS 264-765</td>
<td>VDD+ generator</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>182</td>
<td>U11B</td>
<td>TLE2062ACP</td>
<td>(A/B is dual pk.)</td>
<td>VDD- generator</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>183</td>
<td>U12</td>
<td>TLC2654CN</td>
<td>FN TLC2654CN</td>
<td>Output stage</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>184</td>
<td>U13</td>
<td>OPA654M</td>
<td>Burr Brown</td>
<td>Power Output</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>185</td>
<td>U14</td>
<td>AD1145BG</td>
<td>Analog Devices</td>
<td>Heater/Drive 2</td>
<td>Heater/Drive 2</td>
<td>5</td>
</tr>
<tr>
<td>186</td>
<td>U15</td>
<td>TLC2654CN</td>
<td>FN TLC2654CN</td>
<td>U14 output</td>
<td>Heater/Drive 2</td>
<td>5</td>
</tr>
<tr>
<td>187</td>
<td>U16</td>
<td>TLC2654CN</td>
<td>FN TLC2654CN</td>
<td>Output stage</td>
<td>Heater/Drive 2</td>
<td>5</td>
</tr>
<tr>
<td>188</td>
<td>U17</td>
<td>OPA654M</td>
<td>Burr Brown</td>
<td>Power Output</td>
<td>Heater/Drive 2</td>
<td>5</td>
</tr>
<tr>
<td>189</td>
<td>U18</td>
<td>MAX663CPA</td>
<td>Maxim</td>
<td>Voltage regulator</td>
<td>Heater/Drive 2</td>
<td>5</td>
</tr>
<tr>
<td>190</td>
<td>U19</td>
<td>EX03-20</td>
<td>FN 221-764</td>
<td>Oscillator module</td>
<td>Heater/Drive 2</td>
<td>5</td>
</tr>
<tr>
<td>191</td>
<td>U20</td>
<td>CD4040BCN</td>
<td>FN CD4040CN</td>
<td>Divider</td>
<td>Heater/Drive 2</td>
<td>5</td>
</tr>
<tr>
<td>192</td>
<td>U21</td>
<td>CD401068CN</td>
<td>FN CD401068CN</td>
<td>Inverter</td>
<td>Heater/Drive 2</td>
<td>5</td>
</tr>
<tr>
<td>193</td>
<td>U22</td>
<td>MC74HC573</td>
<td>RS 643-512</td>
<td>Buffer</td>
<td>Auxiliary Devices</td>
<td>6</td>
</tr>
<tr>
<td>194</td>
<td>U23</td>
<td>LM35CZ</td>
<td>FN LM35CZ</td>
<td>Temperature sensor</td>
<td>Auxiliary Devices</td>
<td>6</td>
</tr>
<tr>
<td>195</td>
<td>U24</td>
<td>LM35CZ</td>
<td>FN LM35CZ</td>
<td>Temperature sensor</td>
<td>Auxiliary Devices</td>
<td>6</td>
</tr>
<tr>
<td>196</td>
<td>U25</td>
<td>MAX132CNG</td>
<td>Maxim</td>
<td>Auxiliary ADC</td>
<td>Auxiliary Devices</td>
<td>6</td>
</tr>
<tr>
<td>197</td>
<td>U26</td>
<td>MAX328CPE</td>
<td>Maxim</td>
<td>Analog Multiplexer</td>
<td>Auxiliary Devices</td>
<td>6</td>
</tr>
<tr>
<td>198</td>
<td>VAR1</td>
<td>V250LA40B</td>
<td>FN V250LA40B</td>
<td>Varistor</td>
<td>Power Supplies</td>
<td>2</td>
</tr>
<tr>
<td>199</td>
<td>VR1</td>
<td>10k</td>
<td>Vishay 1260W</td>
<td>Trim resistor</td>
<td>Drive 1</td>
<td>4</td>
</tr>
<tr>
<td>200</td>
<td>VR2</td>
<td>10k</td>
<td>Vishay 1260W</td>
<td>Trim resistor</td>
<td>Heater/Drive 2</td>
<td>5</td>
</tr>
<tr>
<td>201</td>
<td>X1</td>
<td>4.0MHz</td>
<td>RS 658-895</td>
<td>CPU crystal</td>
<td>CPU and ADC</td>
<td>3</td>
</tr>
<tr>
<td>202</td>
<td>X2</td>
<td>32768Hz</td>
<td>RS 304-447</td>
<td>Aux ADC crystal</td>
<td>Auxiliary Devices</td>
<td>6</td>
</tr>
</tbody>
</table>

* Present if REFx is AD5861Q
** Present if REFx is LT1027BCN8

Annex 1: Circuit diagrams
Assemble for 6805 & 6305 family of microprocessors, ver 2.2b.

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 *
00009 *
00010 *
00011 *
00012 *
00013 *
00014 *

00015 * History Record
00016
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 *
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 A01175 before MAX132
00029 * 1.7 Revised routines fpbra to fpbl; Introduced fpbc, fpbs, fpoly
00030 * moving average sample for Precision ADC
00031 * fpadc #0..#7 = MAX132, #6 latest AD1175K, #9..#f mean AD1175K
00032 * (average of last 16 readings - see ADCBL)
00033 * Command C outputs PADC and PADCA, command D outputs 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 rdi 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 Wxxxxxy gives NACK
00041 * 1.11 Routines which control DACs are now interrupt protected, because
00042 * service routine padcs calls getst which clears PBO, PBI, and getr
00043 * leaves them high. putcm also corrups 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 *
00052 * A collection of useful macros
00053 *

00054 dely macro
00055 lda #0
00056 sta TOUT
00057 \.
00058 tst TOUT Tout is decremented every 2.5 milli seconds
00059 bne \.
00060 endm

00061 brst macro may be replaced by brset
00062 nop
00063 noc required by EVM-05
00064 fcb \*2
00065 fcb \1
00066 \.
00067 fcb \2-

- 138 - Annex 3: Embedded software
CurrentDrive Page 2, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

00066  
00068  
00069  
00070  
00071  
00072  
00073  
00075  
00076  
00077  
00079  
00080  
00081  
00082  
00083  
00084  
00085  
00086  
00087  
00088  
00089  
00090  
00091  
00092  
00094  
00095  
00096  
00097  
00098  
00099  
01000  
01001  
01002  
01004  
01005  
01006  
01007  
01008  
01010  
01011  
01012  
01013  
01014  
01016  
01017  
01018  
01019  
01020  
01022  
01023  
01024  
01025  
01026  
01028  
01029  
01030  
01031  
01032  
01033  

endm 

brcl  macr may be replaced by brclr
nop  
nop required by EVM-05
fcb \O*2+1
fcb \1
\O fcb \2-\0-1
endm
mul  macr define multiply instruction
fcb $42
endm

* partial product
pmul  macr (arg2)= (arg2) + arg0 * arg1
lda \0
ldx \1
mul
add \2
sta \2
txa
adc \2-1 high part at lower address
sta \2-1
bcc \2-1 allow for carry to higher byte
inc \2-2
\.1 equ *
endm

fc  macr c defines one or more codes
fcb \0*16+\1
ifgt NARG-2
fcb \2
endc
ifgt NARG-3
fcb \3
endc
endm

* used when defining 'fop's for interpreter jump table
fopdef macr
f\O equ (*-foptab)/2
fdb fp\O
endm

* used when defining function group for interpreter jump table
fundef macr
f\O equ (*-funtab)/2
fdb fp\O
endm

* used when defining branch group interpreter jump table
fbrdef macr
f\O equ (*-fbrtab)/2
fdb fp\O
endm
endm

* constants
BS  equ $08 Back Space
CR  equ $0D Carriage return
LF  equ $0A Line Feed
SP  equ $20 Space

* Port Addresses
PA  equ 0 Port A Data Register - I/O
PB  equ 1 Port B Data Register - I/O
PC  equ 2 Port C Data Register - I/O
PD  equ 3 Port D input register
DDRA equ 4 Data Direction Register for port A

- 139 -  Annex 3 : Embedded software
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 TCHR 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 001F OPT1 equ $1FDF option register
00159 0020 OPT2 equ $3FDF option register in 705C9 (Emulator)
00160 00A0 EEPPA equ $A0 include physical device address of EEPROM
00161 0013 ACHR equ $1A Alternate Counter High Register
00162 001B ACLR equ $1B Alternate Counter Low Register
00163 001C PR equ $1C Program Register
00164 001D COPR equ $1D COP reset register
00165 001E COPC equ $1E COP control register
00166 001F OPT1 equ $1FDF option register
00167 0020 OPT2 equ $3FDF option register in 705C9 (Emulator)

* Bit Assignments

* Timer Control Register - TCR - See P4-9
00168 0007 ICIE equ 7 Input Capture Interrupt Enable
00169 0006 OCIE equ 6 Output Compare Interrupt Enable
00170 0005 TOIE equ 5 Timer Overflow Interrupt Enable
00171 0004 IEDG equ 1 Input EDGE
00172 0003 OLVL equ 0 Output LeVeL

* Timer Status Register - TSR - See p4-10
00173 0007 ICF equ 7 Input Capture Flag
00174 0006 OCF equ 6 Output Compare Flag
00175 0005 TOF equ 5 Timer Overflow Flag

* Serial Communications Control Register 1 - SCCR1 - See p5-5
00176 0007 RB equ 7 Receive bit #8
00177 0006 T8 equ 6 Transmit bit #8
00178 0005 M equ 4 Mode bit (0->8 bits, 1->9)
00179 0004 WAKE equ 3 Wake-up enable - See 5-7

* Serial Communications Control Register 2 - SCCR2 - See p5-7
00180 0007 TIE equ 7 Transmit Interrupt Enable
00181 0006 TCIE equ 6 Transmission Complete Interrupt Enable
00182 0005 RTE equ 5 Receive Interrupt Enable
00183 0004 ILIE equ 4 Idle Line Interrupt Enable
00184 0003 TE equ 3 Transmit Enable
00185 0002 RE equ 2 Receive Enable
00186 0001 RWU equ 1 Receiver Wake-Up
00187 0000 SBK equ 0 Send Break

* Serial Communications Status Register - SCSR - See p5-9
00188 0007 TORE equ 7 Transmit ata Register Empty
00189 0006 TC equ 6 Transmission Complete
00190 0005 RDRF equ 5 Receive Data Register Full
00191 0004 IDLE equ 4 Idle detect bit
00192 0003 OR equ 3 Overrun error
00193 0002 NF equ 2 Noise Flag
00194 0001 FE equ 1 Framing Error

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

- 140 - Annex 3 : Embedded software
CurrentDrive Page 4, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

00202 0005 SCP equ 5 Serial Communications Prescaler
00203 0004 SCP 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,SCRO=1 =>
00206 0000 SCRO 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 SPRO 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 / AO
00241 0000 AO equ NWRLB
00242 0001 NWRHB equ 1 Not Write High Byte / AI
00243 0001 AI equ NWRHB
00244 0002 NRD equ 2 Not Read
00245 0003 NWR equ 3 Not Write
00246 0004 NLDAC equ 4 Not LDA/DAC
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 PCO equ 0 Not used
00254 0001 NS1S 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 MC68HC70SC8
00264 * Emulator using 'C9 part needs data direction register set up
00265 * see config
00266 0000 RX equ 0 RS232 input
00267 0001 TX equ 1 RS232 output
00268 0002 MOSI equ 2 Master In Slave Out from MAX132
00269 0003 MOSO equ 3 Master Out Slave In to MAX132

- 141 -

Annex 3 : Embedded software
`CurrentDrive Page 5, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11`

```
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 0300 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 zrams 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 1FFE 0022 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 0020 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 0038 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

- 142 - Annex 3: Embedded software`
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  FP16B RMB 1  FPI Guard Byte
00353 0060  FPEXE  rmb 1  exponent extension
00354 0061  FPEXOF rmb 1  exponent offset used in square root function
00355 0062  FCNTRI  rmb 1  counter within package
00356 0063  FPTSGN rmb 1  sign of fpi
00357 0064  FP2SGN rmb 1  sign of fpc
00358 0065  FP3SGN rmb 1  sign of result
00359 0066  FPLT  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  floex 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 ‘cfi1gb’
00372  *  lda $1111
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  ACS  rmb  4  sum for moving average
00382 00A4  ACBP  rmb  1  ADC Buffer Pointer
00383 00A5  ADCH  rmb  1
00384 00A5  ADCM  rmb  1
00385 00A7  ADCL  rmb  1
00386 00A8  SBIT  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  SAMples 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  PADCSR  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 0089  MODE  rmb  1  MODE flags
00400  *  allocation of bits in MODE byte
00401 0000  QUIET  equ 0  1=>ADC sampling starts with fsym => 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 008A  TEMPX  rmb 1  temp locations

- 143 -  Annex 3 : Embedded software
CurrentDrive Page 7, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

00406 00BB TEMPA rmb 1 Buffer Pointer for command buffer
00407 00BC BP rmb 1
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 auxiliary ram
00418 0028 CMLEN equ 40
00419 0100 CMDBUF rmb CMLEN

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 cfg 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 A01175 precision ADC
00446 0214 9D nop
00447 0215 9D nop
00448 0216 9D nop
00449 0217 1B 01 bc1r RESET,PB
00450 0219 CD 0231 main1 jsr rd1 read a line
00451 021C 1F 89 bc1r ACK,MORE anticipate not acknowledged
00452 021F CD 026A jsr cmd Command Decode and execute
00453 0221 A6 4E lda #'N' anticipate not acknowledged
00454 0223 brc1 ACK,MORE,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 0FBB jsr crlf end of command
00458 022F 20 0B 0219 bra main1

00460 * read command line into buffer
00461 0231 1F 01 rd1 bc1r RTS,PB allow characters to arrive
00462 0233 5F clr x point to start of buffer
00463 0234 CD 0F9B rd11 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 rd12 no => continue as before
00467 023D 5D ttxt if there have been no characters then ignore
00468 023E 27 F4 0234 beq rd11
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

- 144 - Annex 3 : Embedded software
00474 024D 5A  decc  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 A5 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 0289  beq  rdlz
00484 0262 A3 27  cpzx  #CMOLEN-1
00485 0264 27 CE 0234  beq  rdl1
00486 0266 5C  incx
00487 0267 2D CB 0234  bra  rdl1
00488 0269 81  rdlz  rts

00490 026A 3F BC  * command decode
00491 026B CO 028D  cmd  clrx  BP  clear buffer pointer
00492 026C CD 0280  jsr  ibnc  get next character from input buffer
00493 026F A1 41  cmp  #A  less than 'A'
00494 0271 25 14 0287  bbs  cmdz
00495 0273 A1 5A  cmp  #Z  greater than 'Z'
00496 0275 22 10 0287  bhi  cmdz
00497 0277 A0 41  sub  #A  form word offset
00498 0279 48  lsla
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 0288 033D  * a list of command from A..Z
00509 028A 035F  cmdtb  fdb  cmdA  A - synchronises ADC then outputs 8 numbers
00510 028C 0367  fdb  cmdB  B - repeated cmdA
00512 028E 0388  fdb  cmdC  C - repeatedly outputs PADC, averaged PADC
00513 0290 028C  fdb  cmdD  D - repeatedly outputs averaged PADC
00514 0292 028C  fdb  cmdE
00515 0294 028C  fdb  cmdF
00516 0296 030A  fdb  cmdH  H - Prepare Heater DAC
00517 0298 03A7  fdb  cmdI  I - Output a ramp to Heater DAC
00518 029A 028C  fdb  cmdJ
00519 029C 028C  fdb  cmdK
00520 029E 0404  fdb  cmdL  L - Load all DACs - pulses -LDAC line
00521 02A0 0410  fdb  cmdM  M - set MODE byte
00522 02A2 028C  fdb  cmdN
00523 02A4 028C  fdb  cmdO
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 0578  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 02BA 0615  fdb  cmdX  X - Read moving average of precision ADC in Hex
00533 02BC 0638  fdb  cmdY  Y - Read all MAX132 channels in hex
00534 02BA 064C  fdb  cmdZ  Z - Set some/all DACs

00536 02BC 81  * nothing to do
00537 02BC 81  null  rts

00539 02BC 81  * get next character from input buffer - x points to character just read
00540 02BC 81  * characters are folded to upper case
00541 02BC 81  * z=1 if end of buffer

- 145 -
Annex 3 : Embedded software
00542 02BD BE BC  ibnc  ldx  BP  get buffer pointer
00543 02BF 06 0100  ldx  CMDBUF,x
00544 02C2 98  clc  anticipate end of buffer
00545 02C3 27 09 02CE  beq  ibnc
00546 02C5 3C BC  inc  BP
00547 02C7 A1 60  cmp  #$60  test for lower case
00548 02C9 25 02 02CD  bca  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
00554  * get parameter
00555  * treat characters as hexadecimal and pack into a byte
00556  02CF 3F BD  getpa  clr  PARAM
00557  02D1 AD EA  02BD  bsr  ibnc  get next character
00558  02D3 24 1B  02F0  bcc  getpaz  exit with cy=0 if end of buffer
00559  02D5 AD 1C  02F3  bsr  chtb  convert ascii hex to binary
00560  02D7 24 17  02F0  bcc  getpaz  not HEX
00561  02D9 87 BD  sta  PARAM
00562  02DA AD 00  02BD  bsr  ibnc  get next character
00563  02DB 24 11  02F0  bcc  getpaz  exit with cy=0 if end of buffer
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 BD BD  add  PARAM
00570  02ED 87 BD  sta  PARAM
00571  02EF 99  sec  flag as valid by cy=1
00572  02F0 86 BD  getpaz  lda  PARAM
00573  02F2 81  rts

00575
00576  * convert ascii hex character into binary
00577  * s=1 if ok otherwise 0
00578  02F3 A1 30  cmp  #$0
00579  02F5 25 15  030C  blo  chtby  below '0', and cannot be converted
00579  02F7 A1 39  cmp  #$9
00580  02F9 22 04  02FF  bhi  chtbl  now in binary
00581  02FB A0 30  sub  #$0
00582  02FD 20 0A  0309  bra  chtbx
00583  02FF A1 41  chtbl  cmp  #$A
00584  0301 25 09  030C  blo  chtby  below 'A', and cannot be converted
00585  0303 A1 46  cmp  #$F
00586  0305 22 05  030C  bhi  chtby  above 'F', and cannot be converted
00587  0307 AD 37  sub  #$A-10
00588  0309 99  chtbx  sec  indicated conversion ok
00589  030A 20 01  0300  bra  chtbz
00590  030C 98  chtby  clr  indicates no conversion
00591  030D 81  chtbz  rts

00593
00594  030E BD BB  casb  sta  TEMPA
00595  0310 BF BA  stx  TEMPX
00596  0312 AD 00  0321  bsr  cash
00597  0314 BD 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 BD BB  lda  TEMPA
00604  031E BD BA  ldx  TEMPX
00605  0320 81  rts

00607
00608  0321 44  cash  lsra
00609  0322 44  lsra

- 146 -  Annex 3 : Embedded software
CurrentDrive Page 10, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

00610 0323 44  lsra
00611 0324 44  lsra
00612 0325 97  tax
00613 0326 06 032D  lda  cashtb,x
00614 0329 C0 OFA7  jsr  zputw
00615 032C B1  rts
00616 032D  cashtb  fcc "0123456789ABCDEF"

00618 033D AE 03  cmdA  ldx  #cAtb/256 pointer high
00619 033F A6 47  ldx  #cAtb%256 pointer low
00620 0341 CD 00B0  jsr  fintp interpret
00621 0344 1E 09  bset  ACK,MODE acknowledge
00622 0346 B1  rts

00624 0347  cAtb  fc  fcnt,1,10
00625 0348  cAtb1 fc  fbr,fsyn
00626 0349  cAtb1 fc  fbr,fsyn
00627 034A  fc  fadc,0
00628 034B  fc  ffun,fout
00629 034C  fc  fadc,1
00630 034D  fc  ffun,fout
00631 034E  fc  fadc,2
00632 0350  fc  ffun,fout
00633 0351  fc  ffun,fout
00634 0352  fc  fadc,3
00635 0353  fc  ffun,fout
00636 0354  fc  fadc,4
00637 0355  fc  ffun,fout
00638 0356  fc  fadc,5
00639 0357  fc  ffun,fout
00640 0358  fc  ffun,fout
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,fcexit

00647  * repeated cmdA
00648 035F AD DC 0330 cmdB  bsr  cmdA
00649 0361 CD 0F93  jsr  zget
00650 0364 24 F9  035F  bcc  cmdB
00651 0366 B1  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 OF93  jsr  zget
00658 036C 24 F9  0367  bcc  cmdC
00659 036E 1E 09  bset  ACK,MODE acknowledge
00660 0370 B1  rts

00662 0371 AE 03  cmdC1  ldx  #cCltb/256 pointer high
00663 0373 A6 79  lda  #cCltb%256 pointer low
00664 0375 CD 06B0  jsr  fintp interpret
00665 0378 B1  rts

00667 0379  cCltb  fc  fcnt,1,10
00668 037B  cCltb1 fc  fcnt,2,4
00669 037D  cCltb2 fc  fadc,8
00670 037E  fc  ffun,fout
00671 037F  fc  fadc,9
00672 0380  fc  ffun,fout
00673 0381  fc  fcnt,8+2,cCltb2-*=1
00674 0383  fc  ffun,fcrlf
00675 0384  fc  fcnt,8+1,cCltb1-*=1
00676 0386  fc  ffun,fcrlf
00677 0387  fc  fbr,fcexit

- 147 -  Annex 3 : Embedded software
00680 0388 AD 08 0392 cmdD bsr cmdD
00681 0388 CD 0F93 jsr zget
00682 0388 24 F9 0388 bcc cmdD
00683 038F 1E B9 bset ACK,MODE acknowledge
00684 0391 81 rts

00686 0392 AE 03 cmdD1 lda #D1tb/256 pointer high
00687 0394 86 8A lda #D1tb%256 pointer low
00688 0396 039E 06B0 jsr flntp interpret
00689 0399 81 rts

00691 039A 039C cmdD1 fc fcnt,1,10
00692 039C 039E cmdD1 fc fcnt,2,8
00693 039F 03A0 cmdD1 fc fcnt,8+2,D1tb2-1
00694 03A1 03A3 cmdD1 fc fcnt,8+1,D1tb1-1
00695 03A4 03A6 cmdD1 fc fbr,exit

00702 * test Heater DAC by outputting a RAMP
00703 03A7 3F 5E cmdI clr FPT0
00704 03A8 3F 5D clr FPT1
00705 03A9 03AE jsr getpa get high byte
00706 03A9 24 29 03D9 bcc cmdlz ignore if absent
00707 03BA 87 5B sta FPT3
00708 03BB 03BC jsr getpa get high byte
00709 03BC 24 22 03D9 bcc cmdlz ignore if absent
00710 03BD 03BD 5C sta FPT2
00711 03BE D6 5D cmdIl lda FPT1 get and output high part
00712 03BF 03BE 03EA jsr cmdHh
00713 03C0 03BE 03F7 lda FPT0 get and output low part
00714 03C1 03C3 0409 jsr cmdHl
00715 03C2 03C6 86 5E jsr cmdLs pulse -DAC line
00716 03C7 03C8 03C9 lda FPT0
00717 03CA 03CA 03CB add FPT2
00718 03CB 03CC 03CD sta FPT0
00719 03CD 03CE 86 5D lda FPT1
00720 03CE 03DE 89 5B adc FPT3
00721 03DF 03D0 87 5D sta FPT1
00722 03E0 03DE 03F3 jsr zget continue until a character has arrived
00723 03F4 03D5 24 E2 0389 bcc cmdIl
00724 03E7 1E B9 bset ACK,MODE acknowledge
00725 03D9 81 cmdlz rts

00727 * write to Heater DAC buffer
00728 03DA 038D 042D cmdH jsr getpw get word parameter
00729 03DB 038D 042A 03E9 bcc cmdHz part is missing
00730 03D0 86 8E lda PARH output high part
00731 03D1 03E1 07 03EA bsr cmdHh
00732 03E2 86 8F lda PARL
00733 03E3 03E5 10 03F7 bsr cmdHl
00734 03E7 1E B9 bset ACK,MODE acknowledge
00735 03E9 81 cmdHz rts

00737 03E9 9B cmdHh sei protect sequence
00738 03EB 87 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 N114SH,PC
00742 03F3 16 02 bset N114SH,PC
00743 03F5 9A cli allow interrupts again
00744 03F6 81 rts

00746 03F7 9B cmdHh sei protect sequence
00747 03FB 87 00 sta PA output low byte to heater DAC

Annex 3 : Embedded software
00748 03FA 11 01  bclr NWRLB,PB select low byte
00749 03FC 12 01  bset NWRRB,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 0404 AD 03 0409 cmdL bsr cmdLs
00756 0406 1E B9
00757 0408 81   rts

00760 0410 08 01 0419 bclr NLDAC,PB load all DAC's from internal latches
00761 0411 07 01 041A bset NLDAC,PB
00762 0412 06 01 041F cli allow interrupts again
00763 0414 41 01
00764 0415 81   rts

00767 0416 81   cmdM jsr getpa get parameter
00768 0417 24 04 041D bcc cmdMz ignore if absent
00769 0418 81   rts

00770 041A CD 02CF
00771 041B 24 00 0420 cmdP jsr getpw get word parameter
00772 041C 0D 04 0424 bcc cmdPz part is missing
00773 041D CD 03C jsr paneg
00774 041E 0C 04 0426 lda PARH output high part
00775 041F 26 06 0428 lda PARL
00776 0421 00 04 042A bset PARL
00777 0422 01 04 042C bcc getpwz ignore if absent
00778 0423 02 04 042E getpwz RTS

00779 0424 AD 21 0447 bsr cmdPh
00780 0425 AD 20 044E bsr cmdP1
00781 0426 AD 2A 0454 bcc getpwz ignore if absent
00782 0427 1E B9
00783 0428 81   cmdPz RTS

00784 0429 CD 02CF
00785 042A 00 04 042C lda PARH
00786 042B 04 04 042D lda PARL
00787 042C CD 02CF
00788 042D 05 03 0430 bcc getpwz ignore if absent
00789 042E 0B 03 0433 sta PARH
00790 042F CD 02CF
00791 0430 24 02 0434 bcc getpwz ignore if absent
00792 0431 07 03 0435 sta PARL
00793 0432 81   getpwz RTS

00794 0433 04 04 0436 jsr to parameter PARH:PARL
00795 0434 06 04 0438 jsr getpw get high byte
00796 0435 00 04 043A jsr getpwz get high byte
00797 0436 CD 02CF
00798 0437 04 04 043C bscc PARH
00799 0438 CD 02CF
00800 0439 AB 03 043E sta PARL
00801 043A 02 04 043F sta PARH
00802 043B 81   RTS

00803 043C 0B 04 043F jsr getpw get high byte
00804 043D 0C 04 0440 cli allow interrupts again
00805 0441 CD 02CF
00806 0442 06 04 0444 jsr to parameter PARH:PARL
00807 0443 07 03 0445 jsr getpw get high byte
00808 0444 CD 02CF
00809 0445 02 04 0446 cli allow interrupts again
00810 0446 81   RTS

00811 0447 0B 04 044C jsr getpw get high byte
00812 0448 0C 04 044D jsr getpwz get high byte
00813 0449 CD 02CF
00814 044A 06 04 044A jsr to parameter PARH:PARL
00815 044B 07 03 044B jsr getpw get high byte
00816 044C CD 02CF
00817 044D 02 04 044E cli allow interrupts again
00818 044F 81   RTS

00819 0450 0C 04 0451 jsr getpw get high byte
00820 0451 0D 04 0452 jsr getpwz get high byte
00821 0452 CD 02CF
00822 0453 06 04 0454 jsr to parameter PARH:PARL
00823 0455 07 03 0455 jsr getpw get high byte
00824 0456 CD 02CF
00825 0457 02 04 0458 cli allow interrupts again
00826 0458 81   RTS

- 149 -

Annex 3 : Embedded software
CurrentDrive Page 13, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

00822       * Read precision ADC 20 times
00823 0461 3F SE 0471 cmdQ clr FPT0 counter
00824 0463 AD 0C 0471 cmdQ1 bsr cmdR do reading
00825 0465 CD 0F8B jsr crlf
00826 0468 3C 5E inc FPT0
00827 046A B6 SE lda FPT0
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 brcI NEXT,MODE,cmdR2
00834 0475 cmdRL bsr 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 sty PADCC,x
00839 047F 5C incx
00840 0480 A3 04 cmp #4
00841 0482 26 F7 047B bne cmdR3
00842 0484 10 81 bset 0,PADCST set lb 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 048B 5C incx
00848 048C A3 04 cmp #4
00849 048E 26 F6 0488 bne cmDR4
00850 0492 1E 89 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 0496 24 0D 04A7 bcc cmdSz part is missing
00856 049A CD 043C jsr paneg
00857 049B 06 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 89 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 04A9 B3 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 87 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

- 150 -   Annex 3 : Embedded software
00884 04C2 CD 02BD
00886 04CS 24 29 04F0 bcc cmdTz no parameter
00887 04C7 A1 50 04F4 cmp #P
00888 04C9 27 09 0404 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 set should be OK for approx 1ms but protect anyway
00895 04DF A6 FF lda #SFF address to read then write
00896 04E1 87 BE sta PARH
00897 04E3 0E 01 lda #$7F
00898 04E5 CD 058F jsr eepr
00899 04E7 B7 0D sta command PARAM=contents of ($7FFF)
00900 04E9 9A 9A cli
00901 04EA 9A 9F lda cmdT2
00902 04EF CD 0527 jsr prot
00903 04E8 1E B9 cmdT2 bset ACK,MODE
00904 04F0 01 cmdTz rts
00906 04F4 9B set should be OK for approx 1ms but protect anyway
00907 04F5 9B set work through table
00908 04F6 D6 0569 unpl lda cmdTb2,x
00909 04F7 CD 00D7 jsr spiol
00910 04F8 05 9A incx
00911 04F9 D6 0569 lda cmdTb2,x
00912 04FA 05 9A jsr spiol
00913 04FB 05 9A incx
00914 04FC 05 9A lda cmdTb2,x
00915 04FD 05 9A jsr spiol
00916 04FE 05 9A incx
00917 04FF D6 0569 lda cmdTb2,x get data
00918 0500 87 00 sta PA onto bus
00919 0502 05 9A bset NRD,PA set up control lines
00920 0503 10 01 bclr NWR,PA
00921 0504 1F 02 bclr NEEP,PC and enable
00922 0505 1E 02 bset NEEP,PC
00923 0506 16 01 bset NWR,PA
00924 0507 18 0C incx
00925 0508 19 20 cmp $20
00926 0509 26 DE 04FB bne unp1
00927 050A 9A cli
delay 4
00928 050B 81 rts
00930 050C 9B set should be OK for approx 1ms but protect anyway
00931 050D 9B set work through table
00932 050E 05 9A bset cmdTb1,x
00933 050F 05 9A jsr spiol
00934 0510 05 9A incx
00935 0511 0D 0560 lda cmdTb1,x
00936 0512 CD 00D7 jsr spiol
00937 0513 05 9A incx
00938 0514 06 0560 lda cmdTb1,x
00939 0515 CD 00D7 jsr spiol
00940 0516 03 9C incx
00941 0517 0F 00 sta PA onto bus
00942 0518 14 01 bset NRD,PA set up control lines
00943 0519 17 01 bclr NWR,PA
00944 051A 1F 02 bclr NEEP,PC and enable
00945 051B 1E 02 bset NEEP,PC
00946 051C 16 01 bset NWR,PA
00947 051D 9C incx
00948 051E 9F 00 cmp $5A0
00949 051F 26 DE 0531 bne prot1
00950 0520 CD 035A jsr eepw write byte previously read
00951 0521 0A cli

- 151 -

Annex 3 : Embedded software
* read state of DIL switches and output as hex pair

```assembly
00968 0578 98 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 89 bset ACK,MODE acknowledge
00979 058E 81 rts
```

* read EEPROM at address given in PARH:PARL

```assembly
00981 058F B6 BE eepr lda PARH output address
00984 0591 CD 0D57 jsr spioi via SPI
00985 0594 B6 8F 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 15 01 bset NWR,PB
00990 059F IF 02 bclr NEEP,PC and enable
00991 05A0 B6 00 lda PA get data
00992 05A2 1E 02 bset NEEP,PC
00993 05A4 14 01 bset NRD,PB
00994 05A7 33 04 com DDRA change back to outputs
00995 05A9 81 rts
```

* write to EEPROM at address given in PARH:PARL

```assembly
00998 05A0 B6 BE eepw lda PARH output address
01001 05AC CD 0D57 jsr spioi via SPI
01002 05AF B6 8F lda PARL
01003 05B1 CD 0D57 jsr spioi
01004 05B4 B6 8D lda PARAM get data
01005 05B6 87 00 sta PA onto bus
01007 05B8 14 01 bset NRD,PB set up control lines
01008 05B9 17 01 bclr NMR,PB
01009 05BC 1F 02 bclr NEEP,PC and enable
01010 05BE 4E 02 bset NEEP,PC
01011 05C0 16 01 bset NWR,PB
01012 05C2 81 rts
```

* read a block of bytes from the EEPROM U4

```assembly
01015 05C3 CD 042D cmdV jsr getpw get word into PARH:PARL as start address
01016 05CE 24 21 05E9 bcc cmdVz part is missing
```
CurrentOrivePage16, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

01020 05CB CD 02CF jsr getpa get byte into PARAM as counter
01021 05CB 24 1C 05E9 bcc cmdVz part is missing
01022 05CD cmdVl delay 1
01023 050F 9B sei should be OK for approx 1ms but protect anyway
01024 050F CD 058F jsr eepr read EEPROM
01025 050F 9A cli allow interrupts again
01026 050F CD 030E jsr casb output in hex
01027 050F 3C BF inc PARL increment address
01028 050F 26 02 05E3 bne cmdV2
01029 050F 3C BE inc PARH
01030 050F 3A BD cmdV2 dec PARAM reduce loop counter
01031 050F 5E 6E 05CD bne cmdV1
01032 050F 3E 89 bset ACK, MODE acknowledge
01033 050F 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 050F 5A CD 0420 cmdVn jsr getpw get word parameter into PARH:PARL
01039 050F 24 25 0614 bcc cmdVz part is missing
01040 050F CD 02CF jsr getpa get byte parameter into PARAM
01041 050F 24 20 0614 bcc cmdVz part is missing
01042 050F 45 delay 1 should be OK for approx 1ms but protect anyway
01043 050F 9B sei should be OK for approx 1ms but protect anyway
01044 050F CD 05AA jsr eepw write to EEPROM
01045 0600 89 cli allow interrupts again
01046 0600 9E delay 4
01047 0600 9B sei should be OK for approx 1ms but protect anyway
01048 0600 AD 058F jsr eepr read from EEPROM
01049 0600 89 cli allow interrupts again
01050 0600 81 8D cmp PARAM see if byte was written OK
01051 0600 81 0614 bne cmdVz leave as negative acknowledge
01052 0612 1E 89 bset ACK, MODE acknowledge
01053 0614 81 cmdVz 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 interface readings.
01058 0615 cmdX delay 1
01059 0610 9B sei should be OK for approx 1ms but protect anyway
01060 0610 86 88 lda PADCA+2
01061 0620 87 BD sta PARAM
01062 0622 86 87 lda PADCA+1
01063 0624 87 BF sta PARL
01064 0626 86 86 lda PADCA
01065 0628 87 BE sta PARH copy to temporary area
01066 062A 8A cli
01067 062B CD 030E jsr casb now output as 6 hex characters
01068 062E 86 BF lda PARL
01069 0630 CD 030E jsr casb
01070 0633 86 BD lda PARAM
01071 0635 CD 030E jsr casb
01072 0638 1E 89 bset ACK, MODE acknowledge
01073 063A 81 rts

01076 * copy current MAX132 U25 readings and output in hex
01077 063B CD 0A0F cmdY jsr adccp copy block with interrupt protect
01078 063E 5F clrx
01079 063F 6E 88 cmdYl 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 cpox #ADCLEN
01083 0647 26 F8 063F bne cmdYl
01084 0649 1E 89 bset ACK, MODE acknowledge
01085 064B 81 rts

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

- 153 -

Annex 3: Embedded software
cmdZ jsr cmdS update Secondary DAC
brcl ACK,MODE,cmdZz
jsr cmdP update Primary DAC
jsr cmdH update Heater DAC
jsr cmdL load all DACs simultaneously

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

* define 'fops' ; fopdef creates table entries and offsets
* by concatenating 'f'+'arg' and 'fp'+'arg'. See macro definition
* basic floating point operations jump table

fopdef nop
fopdef sld1
fopdef sld2
fopdef ld1
fopdef st1
fopdef ld2
fopdef add
fopdef sub
fopdef mul
fopdef div
fopdef br
fopdef fun
fopdef cnt
fopdef adc

* branch group jump table
fbrdef bra
fbrdef beq
fbrdef bne
fbrdef bpl
fbrdef bmi
fbrdef call
fbrdef ret
fbrdef exit
fbrdef wt
fbrdef syn
fbrdef bc
fbrdef bs
fbrdef dly
fbrdef fnop D
fbrdef fnop E
fbrdef fnop F

* function group jump table
fundef clr
fundef abs
fundef neg
fundef sgn
fundef sqrt
fundef sgr
fundef spw
fundef fix

- 154 -
Annex 3 : Embedded software
CurrentDrive Page 18, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

01155 05AD undef flt
01157 05AF undef int
01158 05B1 undef frac
01159 05B3 undef out
01160 05B5 undef crlf
01161 05B7 undef spec
01162 0589 0726 fdb fnop E
01163 0588 0726 fdb fnop F

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

01166

01168 05BD 8F 6A fintp stx SMODC+1 high part of source address
01169 05BF 87 6B sta SMODC+2 low part of source address
01170 05C1 CD 070F fintpl jsr gnb get next byte to interpret
01171 05C4 87 6B sta FCODE save code for later
01172 05C6 3F 67 clr FLAGS
01173 05C8 44 lsra
01174 05C9 44 lsra
01175 05CA 44 lsra
01176 05CB 44 lsra
01177 05CC CD 06ED jsr fdec execute using foptab
01178 05CF brst flnoex,FLAGS,fintp2
01179 05D3 86 6B lda FCODE
01180 05D5 A4 0F and #$0F get low nibble
01181 05D7 3F 68 clr FCODE default sub-code to zero
01182 05D9 CD 06ED jsr fdec execute
01183 05DC fintp2 brcl flexit,FLAGS,fintp1
01184 05ED 81 rts return

01186 * set up table offset to access branch group
01187 05E1 12 67 fpbr bset flnoex,FLAGS
01188 05E3 AE 20 ldx #brtab-foptab form offset
01189 05E5 20 08 06EF bra fdec1

01191 * set up table offset to access function group
01192 05E7 12 67 fpfun bset flnoex,FLAGS
01193 05E9 AE 40 ldx #funtab-foptab form offset
01194 05EB 20 02 06EF bra fdec1

01196 * main decoder using look-up tables
01197 05ED AE 00 fdec ldx #foptab-foptab
01198 05EF 6F 5E fdec1 stx FPTO store offset
01199 05F1 4B lsla form word offset
01200 05F2 8B 5E add FPTO point to right table
01201 05F4 97 tax
01202 05F5 D6 0650 lda foptab,x get jump address
01203 05F8 B7 6E sta SMODC+5 into self modifying code
01204 05FA D6 065E lda foptab+1,x
01205 05FD 87 6F sta SMODC+6
01206 05FF 86 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 rolb
01216 0708 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 60 gnb jsr SMODC a=code byte
01222 0711 3C 6B sminc inc SMODC+2 increment address
01223 0713 26 02 0717 bne smincz
CurrentDrive Page 19, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

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

01227 01228 01229 01230 01231 01232 01233 01234 01235
* add accumulator to address with sign extension
smadd tsta sign extend first
0719 40 0718 80 071B 8A 071D 86 071F 87 0721 24 0723 3C 6A
dec SMOC+1 reduce high byte of address
add SMOC+2 add to low byte
dec SMOC+2 sta SMOC+2
smaddz
inc SMOC+1 correct high byte
smaddz rts

01239 01240 01242
* no operation
fpnop rts return with no other operation

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

01248 01249 01250 01251 01252 01253 01254 01255 01256 01257
* load FP1 using code in low nibble of A
fpsld bset finoex,FLAGS indicate no execution of low-nibble code
0727 12 67 jsr cstol convert short to long code
01258
01259
01260
* load FP1 from source determined by x
fpsld jsr gnb get next byte
072E CD 070F fpldla jsr compx compute value of x
01261 01262 01263 01264 01265 01266 01267 01268 01269
01270
01271 01272 01273 01274 01275 01276 01277 01278 01279 01280
* load FP1 from constants in ROM using x as offset i.e. FP1=(RAM)
fpslra lda fpram,x
01281
01282 01283 01284 01285 01286 01287 01288 01289 01290 01291
* load FP1 from constants in EEPROM using a as offset/2 i.e. FP1=(EEPROM)
0728 6B 37
01292 01293 01294 01295 01296 01297 01298 01299 01300 01301
01302 01303 01304 01305 01306 01307 01308 01309 01310 01311
01312

- 156 - Annex 3 : Embedded software
sta 1,x
lda CMDBUF+2
sta 2,x
lda CMDBUF+3
sta 3,x
rts

* not yet implemented

* action is to copy 4 bytes of EEPROM to CMDBUF
fpx asla move msb of offset to carry
clrx
rolx and then into x

* now copy from assigned location plus offset
rts

* store FP1 using code in low nibble of A
fpsst1 bset finoex,FLAGS indicate no execution of low-nibble code
jsr cstol convert short to long code
bra fpsstl

* store FP1 to destination determined by x
fpsst1 jsr gnb get next byte
fpsstl jsr compx compute value of x
bcs fpsstl2 external not yet defined
fpsstl2 0795 0796 bne fpsstl2 store to ram
fpsstl2 0796 0795 bne fpsstl2 store to rom not defined
fpsstl2 0795 0798 rts load FP1

* load FP2 using code in low nibble of A
fpsld2 bset finoex,FLAGS indicate no execution of low-nibble code
jsr cstol convert short to long code
bra fpsld2a

* load FP2 from source determined by x
fpsld2 jsr gnb get next byte
fpsld2a jsr compx compute value of x
bcs fpsld2 load from external device = EEPROM
beq fpsld2 load from ram
bne fpsld2 load from rom

* load FP2 from page zero using x as offset i.e. (RAM)=FP1
fpsld2a lda FP1EX
sta fpram,x
lda fpram+l,x
sta FP2EX
lda fpram+2,x
sta FP2HB
lda fpram+3,x
rts

* load FP2 from constants in ROM using x as offset i.e. FP2=(RAM)
fpsld2ra lda fpram,x
sta FP2EX
lda fpram+1,x
sta FP2HB
lda fpram+2,x
sta FP2MB
lda fpram+3,x
rts

* load FP2 from page zero using x as offset i.e. FP1=(RAM)
fpsld2ra lda fpram,x
sta FP2EX
lda fpram+1,x
sta FP2HB
lda fpram+2,x
sta FP2MB
lda fpram+3,x
rts

* load FP2 from constants in ROM using x as offset i.e. FP2=(ROM)
fpsld2ra lda fpram,x
sta FP2EX
lda fpram+1,x
sta FP2HB
lda fpram+2,x

01360 0708 B7 3A     sta FP2MB
01361 07DA 06 0CC1   lda fprom+3,x
01362 070D B7 3B     sta FP2LB
01363 070F 81        rts

01365 * support routines used by above load/store operations
01366 * input 0Bxxxxxx = RAM  output x=xxxxxxxx0
01367 * 0Lxxxxxx = ROM    a=yyyyyyyy0
01368 * lyyyyyyyy = External
01369 07E0 97        cmpx  tax   copy of byte code number
01370 07E1 58        aslx     x=???????0
01371 07E2 58        aslx
01372 07E3 48        asla
01373 07E4 A5 80     bit  #$80  CY=1 if external, CY=0 if internal
01374 07E5 61        rts  if CY=0 then Z=1 if RAM, Z=0 if ROM

01376 * converts 0000?xxxx to 00000xxxx; ?:0->RAM, ?:1->ROM, xxxx=FPx or FKx
01377 * 0 to 7 => RAM FP0 to FP7
01378 * 8 to 15 => ROM FK0 to FK7
01379 07E7 A8 08     csto1 add #$08  find out if bit #3 is set
01380 07EF 29 04 O7EF bhcs csto1l
01381 07E8 A4 07     and #$07  remove bit #3
01382 07ED 20 02 O7F1 bra csto1z
01383 07EF AB 30     csto1l add #$30  move to bit #6
01384 07F1 81        csto1z 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 O7FF 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 O805 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 O871 beq fpaddz
01401 0805 CD 0960    fpadd0 jsr fp1tst test FP1
01402 0808 27 13 O810 beq fpadd2 result is FP2
01403 080A B6 34     lda FP1EX find difference between exponents
01404 080C 80 35     sub FP2EX
01405 080E 97        tax keep count
01406 080F 26 05 O816 bne fpadd1
01407 0811 CD 0BEC    jsr fppre prepare mantissas
01408 0814 20 30 O846 bra fpadd7 continue immediately - no shifts necessary
01409 0816 24 1E O836 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 O824 bcs fpadd3 do shift
01413 081D AE 0E     fpadd2 ldx #FP2-fpre
01414 081F CD 073A    jsr fp1lra FP1=FP2
01415 0822 20 4D O871 bra fpaddz
01416 0824 B6 38     fpadd3 lda FP2EX set up exponent of result
01417 0826 B7 34     sta FP1EX
01418 0828 CD 0BEC    jsr fppre prepare
01419 082B 34 35     fpadd4 lsr FP1HB
01420 082D 36 38     ror FP1MB
01421 082F 36 37     ror FP1LB
01422 0831 SC        incx
01423 0832 26 F7 O828 bne fpadd4
01424 0834 20 10 O846 bra fpadd7
01425 0836 A1 18     fpadd5 cmp #24 test for worthwhile shift
01426 0838 24 37 O871 bcc fpaddz FPZ is too small therefore result is FP1
01427 083A CD 0BEC    jsr fppre prepare

- 158 -

Annex 3 : Embedded software
CurrentDrive Page 22, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

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 7F  083D  bne fpadd6
01433 0846  fpadd7 bste 7,FPRSGN,fpadd8
01434 084A CD 09A5  jsr   uadd  addition may cause carry
01435 084D CD 0925  jsr   norm
01436 0850 86 64  lda FP2SGN
01437 0852 87 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
* FP1 FP2 abs(FP1)-abs(FP2) required sign use
* FP1 FP2 +4 -2 4-2 -> + -> CY=0 + sign(FP1)
* FP1 FP2 +4 -5 4-5 -> - -> CY=1 - sign(FP2)
* FP1 FP2 -4 +2 4-2 -> + -> CY=0 + sign(FP1)
* FP1 FP2 -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 86 64  lda FP2SGN
01449 0860 87 65  sta FPRSGN  make sign of result = sign of original FP2
01450 0862 20 04 0868 bra fpadda
01451 0864 86 63  fpadd9 lda FPRSGN
01452 0866 87 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 80 37  sub FPILB
01462 0875 87 37  sta FPILB
01463 0877 4F  clra
01464 0878 82 36  sbc FPIMB
01465 087A 87 36  sta FPIMB
01466 087C 4F  clra
01467 087D 82 35  sbc FPIMB
01468 087F 87 35  sta FPIMB
01469 0881 81  rts

01471  * floating point multiply
01472 0882 CD 0960  fpmul jsr fplstst if FP1=0 then
01473 0885 27 26 08AD  beq fpmulz exit
01474 0887 CD 0964 jsr fp2tst if FP2=0 then
01475 088A 26 05 0891  bne fpmul0 then
01476 088C CD 082F jsr fpl1clr FP1=0
01477 088F 20 1C 08AD  bra fpmulz exit
01478 0891 CD 08EC fpmul0 jsr fppre prepare
01479 0894 CD 0889 jsr umul multiply mantissas and return with cy=0
01480 0895 CD 0925 jsr norm normalize
01481 089A 86 34  lda FP1EX  add exponents with byte extension
01482 089C 88 3B  add FP2EX
01483 089E 24 02 08A2 bcc fpmul1
01484 08A0 3C 60  inc FP1EX  correct extension byte
01485 08A2 80 80  fpmul sub #80 remove one bias
01486 08A4 87 34  sta FP1EX
01487 08A6 24 02 08AA bcc fpmulz2
dec FP1EX correct extension byte
01489 08AA CD 0905 fpmul2 jsr fpemld see if exponent is too big/small
01490 08A0 CD 096A fpmulz rts

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

- 159 -  Annex 3 : Embedded software
01496 08B6 26 0B 08C3 bne fpidiv0 then
01497 08BB 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 fplmax FP1=max
01501 08C1 20 EA 08AD bra fmulz exit
01502 08C3 CD 08EC fpidiv0 jsr fppre prepare
01499 08BE CD 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 fplmax FP1=max
01501 08C1 20 EA 08AD bra fmulz exit
01502 08C3 CD 08EC fpidiv0 jsr fppre prepare
01503 08C6 3F 5C 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 3F 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 fpdivv1 correct extension byte
01511 08D7 3A 60 dec FPEXE correct extension byte
01512 08D9 A8 80 fpidiv1 add #$80 restore bias
01513 08DB B7 34 sta FP1EX prepare to do fp operation - computesign of result and set msb's
01520 08EB 81 fpendz rts

01523 * support routines used by fpsub, fpadd, fpmul and fpidiv
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 B6 63 eor FP1SGN
01534 08FE B7 65 sta FP3SGN sign FP1 exor sign FP2
01535 0900 1E 35 bset 7,FP1HB set mb'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 tat FPEXE FPEXE Condition
01541 0907 27 0E 0917 beq fpendz =0 no error
01542 0909 28 07 0912 bmi fpendl <0 underflow
01543 090B 12 66 bset over,FPERR >0 overflow
01544 090D CD 0976 jsr fplmax set to max value using FPRSGN
01545 0910 20 12 0924 bra fpendz
01546 0912 CD 082F fpendl jsr fpclr
01547 0915 20 00 0924 bra fpendz
01548 0917 CD 0960 fpend2 jsr fpstalt 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 mb=1
01557 0925 24 0B 0932 norm bcc norm0
01558 0927 36 35 ror FP1HB increment exponent
01559 0929 36 36 ror FP1HB
01560 092B 36 37 ror FP1LB
01561 092D CD 0952 jsr incexp increment exponent
01562 0930 20 1F 0951 bra normz exit
01563 0932 B6 35 norm0 lda FP1HB nothing to do if mb=1
01564 0934 28 18 0951  bmi normz so exit
01565 0935 8A 36 o ra FP1MB test for zero
01566 0935 8A 37 o ra 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,FP1MB,normz and exit
01571 0944 38 5F fnor2 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 deces
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 correct extension byte
01581 0956 3C 60 inc FP1EX incexz rts
01584 0959 3A 34 decex dec FP1EX increment exponent with byte extension
01585 095B 26 02 095F bne decexz correct extension byte
01586 095D 3A 60 dec FP1EX decexz rts
01587 095F 81 decexz rts

01589 0960 A6 34 fp1tst lda #FP1
01590 0962 20 02 0966 bra fpst
01591 0964 3E 1E fp2tst lda #FP2
01592 0964 3E 1E fp2tst lda #FP2

01596 0966 A6 01 fpstst lda ,x get sign bit
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 fpstsz
01602 096F E6 01 lda 1,x
01603 0971 A4 80 and #$30
01604 0973 AA 01 ora #$01 force to be non-zero
01605 0975 81 fpstsz rts

01607 0976 A6 FF fp1max lda #$FF
01609 0978 B7 34 sta FP1EX
01610 0978 B7 35 sta FP1LB
01611 097A B7 36 sta FP1MB
01612 097C B7 37 sta FP1HB
01613 097E B7 37 sta FP1MB
01614 0980 brst 7,FPRSGN,fp1max
01615 0984 1F 35 bclr 7,FP1HB
01616 0986 81 fp1max rts

01618 0987 98 * unsigned operations
01619 0988 3F 5F usub c1c
01620 0988 3F 5F usub0 clr FP1GB
01621 098A 39 5F rol FP1GB get 1sb of Guard Byte=Carry
01622 098C B6 37 lda FP1LB byte by byte
01623 098E B0 38 sub FP2LB
01624 0990 B7 37 sta FP1LB
01625 0992 B6 36 lda FP1HB
01626 0994 B2 3A sbc FP2HB
01627 0996 B7 36 sta FP1MB
01628 0998 B6 35 lda FP1HB
01629 099A B2 39 sbc FP2HB
01630 099C B7 35 sta FP1HB

- 161 -
CurrentDrive Page 25, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

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 3E 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 09BB 8E 3E umul lda FP1LB
01653 09BB B8 3E ldx FP2LB
01654 09BD 49 mul
01655 09BE B7 5E sta FP1O store lowest part
01656 09CD B0 5D stx FP1T clear higher parts
01657 09C2 3F 5C clr FP2T
01658 09C4 3F 5B clr FP3T
01659 09C6 3F 5A clr FP4T
01660 09C8 3F 59 clr FP5T

01661 * now do other partial products
01662 09CA pmul FP1LB,FP2MB,FP1O
01663 09DC pmul FP1MB,FP2LB,FP1T
01664 09EE pmul FP1MB,FP2MB,FP2T
01665 09F0 pmul FP1MB,FP2MB,FP3T
01666 09A1 pmul FP1MB,FP2MB,FP4T
01667 09A4 pmul FP1MB,FP2MB,FP5T
01668 09A6 pmul FP1HB,FP2MB,FP1T
01669 09A8 pmul FP1HB,FP2MB,FP2T
01670 09A5 A6 59 lda FP1O copy result to FP1
01671 09A5 B7 35 sta FP1HB
01672 09A5 B6 5A lda FP1T
01673 09A6 B7 36 sta FP1MB
01674 09A6 B6 5B lda FP1T
01675 09A6 B7 37 sta FP1LB
01676 09A6 B6 5C lda FP1T keep bottom bits in Guard Byte
01677 09A8 B7 5F sta FP1GB
01678 09A6 81 rts

01680 * Unsigned divide
01681 09A6 AE 19 udiv ldx #25 set up counter
01682 09A6 9B clc
01683 09A6 CD 0988 udiv1 jsr usub0 try subtract FP1=FP1-FP2; cy=1 => won't go
01684 09A7 24 06 OA79 bcc udiv2
01685 09A7 CD 09A5 jsr uadd restore FP1=(FP1-FP2)+FP2=FP1
01686 09A7 9B clc indicate subtraction failed
01687 09A7 20 01 OA7A bra udiv3
01688 09A7 99 udiv2 sec indicate subtraction was ok
01689 09A7 39 5E udiv3 rol FP0 rotate result bits
01690 09A7 C9 5D rol FP1
01691 09A7 39 5C rol FP2 high bit may produce carry
01692 09A8 39 37 rol FP1LB
01693 09A8 39 36 rol FP1MB
01694 09A8 39 35 rol FP1HB
01695 09A8 5A decx
01696 09A7 26 E5 OA6E bne udiv1
01697 09A8 36 37 ror FP1LB recover msb to CY
01698 09A8 B6 5E lda FP0
01699 09A8 B7 37 sta FP1LB

- 162 -  Annex 3 : Embedded software
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 B1  udivz rts  result in CY:FP1HB:FP1MB:FP1LB

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

01710  * branch always
01711  0A9B CD 070F  fpbra  jsr  gnb  get byte
01712  0A9B CD 071B  jsr  smadd  and add it in
01713  0A9E B1  rts

01715  0A9F CD 070F  fpbran  jsr  gnb  no branch => throw away offset byte
01716  0AA2 B1  rts

01718  * branch if FP1=0
01719  0AA3 CD 0960  fpbeq  jsr  fplst
01720  0AA6 27 F0  OA98  beq  fpbra
01721  0AA8 20 F5  OA9F  bra  fpbran

01723  * branch if FP1<>0
01724  0AAA CD 0960  fpbne  jsr  fplst
01725  0AAD 26 E9  OA98  bne  fpbra
01726  0AAF 20 EE  OA9F  bra  fpbran

01728  * branch if FP1 sign bit = 0
01729  0AB1 CD 0960  fpbpl  jsr  fplst
01730  0AB4 2A E2  OA98  bpl  fpbra
01731  0AB6 20 E7  OA9F  bra  fpbran

01733  * branch if FP1 sign bit = 1
01734  0AB8 CD 0960  fpbmi  jsr  fplst
01735  0ABB 2B DB  OA98  bmi  fpbra
01736  0ABD 20 E0  OA9F  bra  fpbran

01738  0A8F B1  fpcall  rts  call subroutine
01740  0AC0 B1  fpret  rts  return from subroutine

01742  * exit interpreter
01743  0AC1 10 67  fpexit  bset  flexit,FLAGS
01744  0AC3 B1  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  QACC  bcc  fpwtz
01749  0AC9 CD 0F9B  jsr  zgetw  now wait for a character
01750  0ACC B1  fpwtz  rts

01752  * synchronise with CHANnel counter; wait for channel 3 to become 0
01753  0ACD  fpsynt  brcl  QUIET,MODE,fpsynt
01754  0A01 A6 FF  lda  #$FF
01755  0A03 B7 AD  sta  SAMCNT
01756  0A05 B6 AD  fpsynt  lda  SAMCNT  wait for 8 entries
01757  0A07 A1 08  cmp  #0
01758  0A09 26 FA  OA05  bne  fpsynt
01759  0A0B CD 0ADF  jsr  adccp  copy ADC block
01760  0A0E B1  rts

01762  0A0F  adccp  dely  1  protect environment for copy
01763  0AE7 9B  sei
01764  0AE8 5F  clr
01765  0AEB E6 70  adccpl  lda  ADCR,x  copy block from ADCReadings to
01766  0AEB E7 88  sta  ADCC,x  ADCCopy of readings
01767  0AED 5C  incx

- 163 - Annex 3 : Embedded software
01768 OAE A3 18  
cpx #ADCLEN
01769 OAFE 26 F7  OAE9  
bne edcclpl
01770 OAFE 3F AD  
clr SAMCNT flag that all readings have been copied
01771 OAFE 4A  
cli enable again
01772 OAF5 81  
rts

01774  
* usage 'fcfbr,fpbc,mask,port,offset'
* branch if bits selected by mask are all zero
01776 OAFE CD 070F  
fpbc jsr gnb get mask
01777 OAFE B7 BB  
sta TEMPA
01778 OAFE CD 070F  
jsr gnb form an index
01780 OAFF F6  
lda ,x get data
01781 OBOO B4 BB  
fpbc1 and TEMPA apply mask
01782 OBO2 26 08  
OBOC bne fpbc2 add offset
01783 OBO4 CD 070F  
jsr gnb add offset
01784 OBO7 CD 0718  
jsr smadd to address
01785 OBOA 20 03  
OBOF bra fpbcz
01786 OBOC CD 070F  
fpbc2 jsr gnb throw offset away
01787 OBOF 81  
fpbcz rts

01790  
* usage 'fcfbr,fpbs,mask,port,offset'
* branch if bits selected by mask are all one
01792 OBO1 CD 070F  
fpbs jsr gnb get mask
01793 OBO3 B7 BB  
sta TEMPA
01794 OBO5 CD 070F  
jsr gnb form an index
01795 OBO6 97  
tax get data
01796 OBO8 F6  
lda ,x get data
01797 OBOA 43  
com
01798 OBOB 20 03 0BOO bra fpbc1

01800  
* delay for multiple of 0.1 seconds; 25 seconds max
01801 OBO1 CD 070F  
fpdly jsr gnb delay for n * 0.1 sec
01802 OBO2 B7 BB  
sta TEMPA
01803 OBO5  
fpdly1 delay 40
01804 OBOA 3A BB  
dec TEMPA
01805 OBOC 26 F4  
OBO2 bne fpdly1
01806 OBOE 81  
rts

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

01812  
* clear floating point accumulator 1, ie set to zero
01813 0BO2F  
fpclr equ *
01814 OBO2F 3F 34  
fpclr clr FP1EX FP1=0
01815 OBO31 3F 35  
clr FP1HB
01816 OBO33 3F 36  
clr FP1MB
01817 OBO35 3F 37  
clr FP1LB
01818 OBO37 81  
rts

01820  
* absolute value
01821 OBO3 1F 35  
fpabs bclr 7,FP1HB
01822 OBO3A 81  
rts

01824  
* negate value
01825 OBO3 0860  
fpneg jsr fpst if FP1<>0 then toggle sign bit
01826 OBO3E 27 06  
OBO4 beq fpnegz
01827 OBO40 B6 35  
lda FP1HB
01828 OBO42 A8 80  
eor #$80
01829 OBO44 B7 35  
sta FP1HB
01830 OBO46 81  
fpnegz rts

01832  
* FP1=-1,0,+1 matching sign of FP1
01833 OBO7 0D 0960  
fpst jsr fpst
01834 OBOA 27 10  
OBOC beq fpst if zero
01835 OBOC 97  
tax save sign

- 164 - Annex 3 : Embedded software
01836 084D A6 81 lda #$81 set up +1
01837 084F B7 34 sta FP1EX
01838 0851 3F 35 clr FP1HB
01839 0853 3F 36 clr FP1MB
01840 0855 3F 37 clr FP1LB
01841 0857 50 tsto
01842 0858 2A 02 BPL fpmsgnz leave as plus 1
01843 085A 1E 35 bset 7,FP1HB make negative
01844 085C 81 fpmsgnz rts

01846 * makes use of FP0,FP1,FP2 and FP3
01847 085D 0D 0960 fpmsg jsr fpmsgst
01848 0860 27 6D 0BCF beq fpmsgzM nothing to do if zero
01849 0862 2A 05 0B69 bpl fpmsgzt ok if positive
01850 0864 CD 0B2F jsr fpmsgC clear if negative
01851 0867 20 66 0BCF bra fpmsgE exit
01852 0869 86 34 fpmsgt0 lda FP1EX reduce range
01853 086B AE 80 sub #$80 remove bias
01854 086D 47 asra divide by 2 saving sign
01855 086E B7 61 sta FP2OF exponent offset - 2's complement
01856 0870 A6 40 lda #$40
01857 0872 49 rola $80 or $81
01858 0873 B7 34 sta FP1EX new exponent
01859 0875 AE 0C ldx #FP3-fprom store FP1 to FP3 - call it y
01860 0877 CD 0796 jsr fp1Ra
01861 087A 0F 0F PLA 0,FP1EX,fpsqt1
01862 087E AE 14 ldx #fpsqt-b-fprom load fp2 with 0.5901621 - constant b
01863 0880 CD 07CB jsr fp21ro
01864 0883 CD 0882 jsr fpmul multiply FP1=FP1*FP2
01865 0886 AE 10 ldx #fpsqt-b-fprom load fp2 with 0.4173076 - constant a
01866 0888 CD 07CB jsr fp21ro
01867 088A CD 0860 jsr fpadd add FP1=FP1+FP2
01868 088E 20 10 0BA0 bra fpsqt2
01869 0890 AE 10 fpsqt1 ldx #fpsqt-a-fprom load fp2 with 0.4173076 - constant a
01870 0892 CD 07CB jsr fp21ro
01871 0895 CD 0862 jsr fpmul multiply FP1=FP1*FP2
01872 0898 AE 14 ldx #fpsqt-b-fprom load fp2 with 0.5901621 - constant b
01873 089A CD 07CB jsr fp21ro
01874 089C CD 0800 jsr fpadd add FP1=FP1+FP2
01875 08A0 AE 0D fpsqt2 ldx #FP0-fprom
01876 08A2 CD 0796 jsr fp1Ra store FP1 to FP0 - store estimate xi
01877 08A5 AE 02 lda #2
01878 08A7 B7 62 sta FP2CNt set up counter
01879 08A9 AE 0C fpsqt3 ldx #FP3-fprom load FP1 from FP3 - recover y
01880 08AB CD 073A jsr fp1Ra
01881 08A9 AE 00 ldx #FP0-fprom load FP2 from FP0 - recover x1
01882 08A8 CD 078A jsr fp1Ra
01883 08B3 CD 08AE jsr fpdiv divide FP1=FP1/FP2
01884 08B4 AE 00 ldx #FP0-fprom load FP2 from FP0 - recover x1
01885 08B6 CD 078A jsr fp21ra
01886 08B8 CD 0800 jsr fpadd add FP1=FP1+FP2
01887 08BE 3A 34 dec FP1EX divide result by 2
01888 08C0 AE 00 ldx #FP0-fprom store FP1 to FP0 - new estimate
01889 08C2 CD 0784 jsr fp1Ra
01890 08C5 3A 62 dec FP2CNt loop counter
01891 08C7 26 E0 0BA9 bne fpmsgt3
01892 08CB B6 34 lda FP1EX correct exponent
01893 08CB BB 61 add FP2OF
01894 08CD B7 34 sta FP1EX
01895 08CF 81 fpsqtz rts

01897 * square FP1
01898 08D0 AE 04 fpsqr ldx #FP1-fprom
01899 08D2 CD 078A jsr fp21ra FP2=FP1
01900 08D5 CD 0882 jmp fpmul

01902 * swap FP1 with FP2
01903 08D8 AE 03 fpswp ldx #3 set up counter
* convert floating point number in FP1 into fixed format

* Exponent=0; sign+7+8+8 data bits in FP1HB:FP1MB:FP1lB

* set FP1 to maximum signed integer - 007FFFFF

* float integer in FP1 in 3 byte signed format

* find integer part of FP1. (FP2 corrupted)

* find fractional part of FP1. (FP2 corrupted)

* generate mask in FP2 mantissa corresponding to the position

* of the binary point; 1's before and 0's after

* generate mask

* set FP1 to maximum signed integer

* set FP1 to maximum signed integer - 007FFFFF
01972 OC4E B7 3A sta FP2MB
01973 OC50 B7 3B sta FP2LB
01974 OC52 B6 34 lda FP1EX
01975 OC54 A0 80 sub $#80 test too small
01976 OC56 23 0C OC64 bls fgmksz
01977 OC58 99 fgmksl sec introduce i's
01978 OC59 36 39 ror FP2HB
01979 OC5B 36 3A ror FP2MB
01980 OC5D 36 38 ror FP2LB
01981 OC5F 25 03 OC64 bcs fgmksz full up
01982 OC61 4A deca
01983 OC62 26 F4 OC58 bne fgmks1
01984 OC64 81 fgmkz rts

01986 * apply mask in FP2 to FP1
01987 OC65 B6 37 fmsk lda FP1LB apply mask byte by byte
01988 OC67 B4 3B and FP2LB
01989 OC69 B7 37 sta FP1LB
01990 OC6B B6 36 lda FP1MB
01991 OC6D B4 3A and FP2MB
01992 OC6F B7 36 sta FP1MB
01993 OC71 B6 35 lda FP1HB
01994 OC73 B4 39 and FP2HB
01995 OC75 B7 35 sta FP1HB
01996 OC77 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 OC7B 12 67 fpcnt bset finoex,FLAGS indicate no execution of low-nibble code
02004 OC7A A1 08 cmp #$08 test for >7
02005 OC7C A4 07 and #$07 allow only counters 0 to 7
02006 OC7E 97 tax save index
02007 OC7F 25 0C OC80 bcs fpct1 just load counter
02008 OC81 0D 0F jsr gnb get displacement if needed
02009 OC84 6A 50 dec FCNTO,x decrement and branch if non zero
02010 OC86 27 0A OC92 beq fpctz nothing to do if zero
02011 OC88 CD 0718 jsr smadd add a to self modifying code with sign extension
02012 OC8B 20 05 OC92 bra fpctnz
02013 OC8D CD 070F fpctnz lsr gnb
02014 OC90 E7 50 sta FCNTO,x load counter
02015 OC92 81 fpctnz rts

02017 * convert ADC channel given in low nibble - pass to FP1 as 0 to 2.0(ish)
02018 * Number format from CSS505 is 0000 to FFFF for -2.5v to +2.5v
02019 * Subtract $80 from high byte to change to 2's complement

02020 *******************************************************

02021 OC93 12 67 fpadc bset finoex,FLAGS indicate no execution of low-nibble code
02022 OC95 A1 08 cmp #8 test for MAX132 range 0 to 7
02023 OC97 25 19 OC82 blo fpadc1
02024 OC99 A4 07 and #7
02025 OC9B 97 tax pick up start address of 3-byte number
02026 OC9C DE 0CD6 ldx fpadct,x #8 -> PADCR, >= #9 ->PADCA
02027 OC9F 9B sei protect transfer
02028 OCAC E6 02 lda 2,x
02029 OCAD B7 37 sta FP1LB
02030 OCA4 E6 01 lda 1,x
02031 OCA6 B7 36 sta FP1MB
02032 OCAF F6 lda ,x
02033 OCA9 9A cli
02034 OCAA 0A 40 sub $#40 convert to 2's complement
02035 OCBE A6 35 sta FP1HB
02036 OCAE A6 83 lda #83 prepare exponent
02037 OCB0 20 12 OCC4 bra fpadc2
02038 OCB2 AE 03 fpadc1 ldx #3 form offset
02039 OCB4 mul
02040 OCBS 97       tax            into ADCC table
02041 OCBS 66 8A    lda ADCC+2,x get low part
02042 OCBS 87 37    sta FPILB
02043 OCBS A6 89    lda ADCC+1,x get middle part
02044 OCBS B8 3E    sta FPIMB
02045 OCBS E8 88    lda ADCC,x get high part
02046 OCBS D7 35    sta FPIH
02047 OCBS C2 A8 65 lda #085
02048 OCBS C4 B7 34 fpadc2 sta FP1EX anticipate bias
02049 OCBS C6 3F 60 clrr FPEXE clear extension
02050 OCBS C8 3F 65 clrr FPRSGN anticipate positive
02051 OCA7 1E 65    bset 7,FPRSGN indicate negative
02052 OCO0 00 872   jsr Ineg integer negate
02053 OCO3 CC 0668   fpadc3 jmp fpadda normalise and return

* table to convert channel numbers above 7 to start address

02057 OCD6 AE      fpadct fcb PADCR, PADCA, PADCA, PADCA, PADCA, PADCA, PADCA
* at most 64 constants. First 8 accessible by short load operations.

02061 OCDE fprom equ *
02062 OCDE 00      fko fcb 0,0,0,0, zero
02063 OCDE 81      fke0 fcb $81,$00,$00,$00 + 1
02064 OCDE 84      fke1 fcb $84,$20,$00,$00 + 10
02065 OCDE 94      fke6 fcb $84,$74,$24,$00 + 1,000,000
02066 OCDE 7F      fksqta fcb $7F,$55,$A9,$57 + 0.4173076
02067 OCDE BO      fksqtb fcb $80,$17,$14,$00 + 0.5901621
02068 OCFA 00      fcb 0,0,0,0

* General configuration and support routines

* Note references to SMOO must be present for interpreter to work

02070 OCFA 9B      config sel must be protected
02071 OCFA 5F      clr offset into table
02072 OCFA 00      config STx SPIx keep x
02073 OCFA 00      lda configb+1,x get data
02074 OCFA 00      ldx configb,x get address
02075 OCFA 00      cpx #$FF
02076 OCFA 00      beq config2 end of table
02077 OCFA 00      sta ,x write to port
02078 OCFA 00      ldx SPIx recover x
02079 OCFA 00      ldx SPIx move to next entry
02080 OCFA 00      bra config1 and continue

02081 OCFA 00      config2 lda #X11000010 extra ram at $20 and $100
02082 OCFA 00      sta OPT1 68HC705C8
02083 OCFA 00      sta OPT2 68HC705C9 (used in EVNO5 emulator)
02084 OCFA 00      lda #X00100010 no start,50Hz, awake,vin,BA=01 => status
02085 OCFA 00      jsr adcc configure MAX132 ADC
02086 OCFA 00      jsr adccb clear buffer for precision ADC readings
02087 OCFA 00      rts

* configuration table

02088 OCFA 00      configtb fcb PA,$00 data bus as outputs
02089 OCFA 00      fcb PB,%001111111 controls inactive
02080 OCFA 00      fcb PC,%11111111 all selects on stand-by
02081 OCFA 00      fcb DORA,$FF default state of 'bus' is output
02082 OCFA 00      fcb DORB,$FF
02083 OCFA 00      fcb DORC,$FF CONV,CAL,CS's, SCLas outputs
02084 OCFA 00      fcb DORD,$00110000 needed for SPI with EVNO5 using MC68HC705C9
02085 OCFA 00      jsp SPCR,%01010001
02086 OCFA 00      jsp SCSBR,%000110000 9600 baud
02087 OCFA 00      jsp SCCR1,%00001000
02088 OCFA 00      jsp SCCR2,%00001100

- 168 -

Annex 3: Embedded software
CurrentOrivePage32, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

02108 003A 12  fcb  TCR,$01100000  OCIE,TOIE on, OLVL=1 => LED off
02109 003C 1C  fcb  PR,$00000000 Program EEPROM
02110 003E A9  fcb  TOUT,0  counters to zero
02111 0040 AA  fcb  TOVF,0
02112 0042 AB  fcb  STATE,40 force time-out
02113 0044 AC  fcb  CHAN,0
02114 0046 B0  fcb  SAMCNT,$F8 allow ($100-$F8+8) test/warm-up cycles
02115 0048 B9  fcb  MODE,0 allow sampling during RS232 reception

02116 * following are needed for floating point routines
02117 004A 69  fcb  SMODOC,$C6 lda extended
02118 004C 6C  fcb  SMODOC+3,$81  rts
02119 004E 6D  fcb  SMODOC+4,$CC  jmp extended
02120 0050 C0  fcb  BLSTAT,5 Blink State => 5 changes
02121 0052 C1  fcb  BLCNT,0 Blink interval counter is cleared
02122 0054 C2  fcb  BLIVL,4 Blink Interval 4 * 0.131s = 0.5
02123 0056 FF  fcb  $$FF Stopper code

02126 * spi (interface) output and input a byte
02127 0057 B7 A8  spioi  sta  SPIT  keep last byte sent out
02128 0059 B7 OC  sta  SPDAT output a byte to get a byte
02129 005B 00  brcl SPIF,SPSR,spioi
02130 005F B6 OC  lda  SPDAT get data
02131 0061 81  rts

02134 * Timer interrupt service routine
02135 04E2  TINV  lalf  2500*4/8 (interval in us)* (clock in MHz.) / 8
02136 0062  tims  brcl  TOF,TSR,tims1 (interval in us)* (clock in MHz.) / 8
02137 0068 B6 19  lda  CLR  remove overflow flag
02138 006A 00 CO  lds  BLSTAT if BLSTAT<>0 then
02139 006C 27 20  beq  tims1
02140 006C 27 01  006C  beq  tims0
02141 006E 27 04  0074  beq  tims0
02142 0070 3A C1  dec  BLCNT dec(BLCNT)
02143 0072 20 12  006E  bra  tims0a
02144 0074 B6 C2  tims0 lda  BLIVL else begin
02145 0076 B7 C1  sta  BLCNT BLCNT+=BLIVL
02146 0078 B6 12  lda  TCR TCR:=TCR xor $01 for OLVL
02147 007A A8 01  eor  #$1<OLVL
02148 007C B7 12  sta  TCR
02149 007E B6 C0  lda  BLSTAT
02150 0080 A1 FF  cmp  #$FF if BLSTAT<>$$FF then dec(BLSTAT)
02151 0082 27 02  0086  beq  tims0a
02152 0084 3A C0  dec  BLSTAT end
02153 0086 30 AA  tims0a  tst  TOVF
02154 0088 27 02  008C  beq  tims1
02155 008A 3A AA  dec  TOVF
02156 008B 00  tims1 brcl  OCF,TSR,tims2 end
02157 0090 B6 17  lda  OCLR add offset for timing interval
02158 0092 A8 E2  add  #TINVLSX56 low byte
02159 0094 97  tax  OCLR must be written last!
02160 0095 B6 16  lda  OCHR
02161 0097 A9 04  adc  #TINV/256 high byte
02162 0099 B7 16  sta  OCHR
02163 009B BF 17  stx  OCLR re-enable by write to low byte
02164 009D CD 00AA  jsr  padcs service precision adc
02165 00A0 CD 0E77  jsr  adcs service adc
02166 00A3 3D A9  tst  TOUT
02167 00A5 27 02  00A9  beq  tims2
02168 00A7 3A A9  dec  TOUT
02169 00A9 80  tims2 rti may be extended to allow for ICF

02170 * service precision adc
02172 00A4 CD 00F  padcs jsr  getst get status byte
02173 00A4 A5 81  bit  #81 leave if warming up or busy
02174 00AF 26 00  00BE  bne  padcag
02175 00B1 B7 B1  sta  PADCST save in status byte

- 169 -

Annex 3 : Embedded software
**get status of AD1175K by reading register 0**

```assembly
getsr GETR A0, PB set to register 0
bcrl A1, PB
clr DDRA change port A to inputs
bcrl N1175, PA select AD1175K
bcrl NRD, PB enable reading
lda PA get data
bset NRD, PB disable reading
bset N1175, PB disable AD1175K
com DDRA restore to outputs
rts
```

**get readings from precision ADC**

```assembly
getr clr DDRA change port A to inputs
ldx #2 form index to PADCR table
getrl inc PB move from status byte to next byte(s)
bclrl PA, B0 00
sta PADCR, x store in table
bclrl NR, PB enable reading
bclrl N1175, PB disable writing
bset NR, PB disable writing
bset N1175, PB disable AD1175K
decx move to higher byte
bpl getr
com DDRA restore to outputs
rts
```

**put command to precision ADC**

```assembly
putcm sta PA
bcrl A0, PB set to register 00
bcrl A1, PB
bcrl N1175, PB select AD1175K
bcrl NWR, PB enable writing
nop
bset NWR, PB disable writing
bset N1175, PB disable AD1175K
rts
```

**sum: = sum - buf(index); buf(index) = new entry = ADCH, M, L; sum: = sum + buf(index);**

```assembly
sum: = sum - buf(index)
sum: = sum + buf(index)
```

**wrapup/round 1;**

```assembly
```

**Routine timed at 120us**

```assembly
```

**samplib lda ADCBP
```

**Account for borrow from highest byte**

```assembly
```
02244 0E28 B6 A2  lda ADCS+2
02245 0E20 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  sampb2  txs get start point
02252 0E30 9F  inc ADCS account for carry to highest byte
02253 0E3E A8 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 3B 86  ror PADCA
02268 0E59 3A 86  ror PADCA+1
02269 0E5A 3B 86  ror PADCA+2
02270 0E5B 5A  decx
02271 0E5F 26 0F 0E57  bne sampb4
02272 0E61 B1  rts PADCA now is sum/2^n (n=ADCBD)

02274 0E62 5F  adccb clrx clear ADC buffer, pointer and sum
02275 0E63 4F  clra
02276 0E64 B7 A4  sta ADCBP
02277 0E66 B7 A0  sta ADCS
02278 0E68 B7 A1  sta ADCS+1
02279 0E6A B7 A2  sta ADCS+2
02280 0E6C B7 A3  sta ADCS+3
02281 0E6D 07 0128  adccb1 sta ADCB,x
02282 0E71 5C  inca
02283 0E72 3A 30  cpux #ADCBL
02284 0E74 26 0F 0E6E  bne adccb1
02285 0E76 81  rts

02288 0E77 3C A8  adcs inc STATE
02289 0E79 B6 A8  lda STATE
02290 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

- 171 - Annex 3 : Embedded software
02312 OEF4 20 62 OEF4  bra adcsz now end
02313 OEF4 A2 28 adcs1 cmp #40
02314 OEF4 24 57 OEEP bhs adcs3 greater than 100ms
02315 OEF4 66 42 lda #%X10000010 no start,50Hz,awake,vin,BA=01 = status
02316 OEF4 CD OEF5 jsr adccom
02317 OEF4 A5 40 bit #%X10000000 wait for EOC to become high
02318 OEF4 27 55 OEF4 beq adcsz nothing to do
02319 OEF4 B7 A7 sta ADCL store status and lsb's
02320 OEF4 A6 44 lda #%X10000010 no start,50Hz,awake,vin,BA=10 => B18..B11
02321 OEF4 CD OEF5 jsr adccom
02322 OEF4 A6 40 lda #%X10000000 no start,50Hz,awake,vin,BA=00 => B10..B3
02323 OEF4 CD OEF5 jsr adccom
02324 OEF4 B7 A5 lda ADCH
02325 OEF4 A6 C2 lda #%X11000010 start,50Hz,awake,vin,BA=01 => status
02326 OEF4 CD OEF5 jsr adccom
02327 OEF4 B6 A7 sta ADCM
02328 OEF4 B6 A7 lda ADCL extract polarity and data bits
02329 OEF4 48 asla
02330 OEF4 48 asla
02331 OEF4 48 asla
02332 OEF4 48 asla
02333 OEF4 48 asla
02334 OEF4 B6 A5 ror ADCH move polarity to msb
02335 OEF4 B6 A6 ror ADCM
02336 OEF4 46 rora
02337 OEF4 AE 04 ldx #4
02338 OEF4 AE 07 adcs2 asr ADCH now move to right justify
02339 OEF4 AE 03 ror ADCM
02340 OEF4 CE 46 rora
02341 OEF4 5A decx
02342 OEF4 C6 F8 OEC2 bne adcs2
02343 OEF4 B7 A7 sta ADCL
02344 OEF4 B6 AC lda CHAN
02345 OEF4 4A deca restore previous channel
02346 OEF4 AF 07 and #7
02347 OEF4 AE 03 ldx #3
02348 OEF4 AD3 mul
02349 OEF4 97 tax form an index
02350 OEF4 B6 A5 lda ADCH
02351 OEF4 E7 70 lda ADCM sta ADCR,x copy to destination
02352 OEF4 B6 A6 lda ADCM
02353 OEF4 E7 71 lda ADCL sta ADCR+1,x
02354 OEF4 D6 A7 lda ADCL
02355 OEF4 E7 72 lda ADCR+2,x
02356 OEF4 3F AB clrf STATE
02357 OEF4 B6 AD lda SAMCNT increase sample count
02358 OEF4 A1 08 cmp #8
02359 OEF4 27 08 OEF4 beq adcsz limit at 8
02360 OEF4 3C AD lda SAMCNT
02361 OEF4 20 07 OEF4 inc SAMCNT
02362 OEF4 B6 C2 adcs3 lda #%X11000010 start,50Hz,awake,vin,BA=01 => status
02363 OEF4 CD OEF5 jsr adccom
02364 OEF4 3F AB clrf STATE
02365 OEF4 81 adcsz rts
02367 * MAXIM MAX132 ADC communication via SPI
02368 OEF5 1D 02 adccom bclr N132,PC
02369 OEF7 CD 0D57 jsr spioi
02370 OEF4 1C 02 bset N132,PC
02371 OEF4 81 rts
02372 * output integer part of FP1
02373 OEF4 AE OC fpout ldx #%Fk1e6-fprom get constant of 1e6
02374 OEF4 CD 07CB jsr fp2lro load FP2 from rom
02375 OEF4 CD 08B2 jsr fpmul find product
02376 OEF4 CD 08EA jsr fpfix
02377 OEF4 86 A0 lda #5P output a space
02378 OEF4 CD 0FA7 jsr zputw

- 172 - Appendix 3: Embedded software
lda #'*' anticipate plus sign
brcl 7,FP1HB,fpoutl
bclr 7,FP1HB clear sign bit
lda #'*-' minus sign
fpoutl jsr zputw output sign
jsr btod binary to decimal conversion
jsr drnd round up
jsr dprnt in fixed point format x.xxxxx
rts

* output a carriage return and line feed
fpcrlf jmp crif

* output a space
fpspac lda #$20 code for a space
jmp zputw output and return

* decimal round up if least significant digit is >=5
drnd ldx #5 get least significant digit
da DMAL+1,x digit #6, le 7th. digit
cmp #$05 test for range
blo drndz nothing to do as digit is 0..4
sec force increment
drnd1 lda DMAL,x BCD add
adc $0 decimaladjust
bcs drnd2
sec
drnd3 sta DMAL,x
decx
bpl drnd1
drndz rts

* print decimal string without leading zero suppression
* decimal point after first digit
dprnt clrx
dpnlnd1 lda DMAL,x get number
add #$30 convert to ascii
jsr zputw
lncx
cpx #1 decimal point after first digitnbne dprnt2
lda #'. output a decimal point
drnd2 clc
drnd3 sta DMAL,x
decx
drndz rts

* binary to decimal conversion
btod ldx #6 clear result registers
drnd1 clr DMAL,x
decx
drnlbtod1 bpl btod1
ldc #24 24 loops
sta FPCTN12
btod2 rol FP1LB shift binary pattern
rol FP1MB
rol FP1HB
rol
bcs btod4
add #6 set up loop counter
btod3 lda DMAL,x decimal double
rola
cmp #10 decimal adjust
bne dprnt1
bne btod4
add #6

- 173 - Annex 3 : Embedded software
CurrentDrive Page 37, 10:21 27 Jan 1994 Version 1.12 incorporating FP2.11

02448 0F78 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 ret

02459 * output CR and LF
02460 0F88 A6 0D crlf lda #CR
02461 0F8D AD 18 OFA7 bar zputw
02462 0F8F A6 0A lda #LF
02463 0F91 20 14 OFA7 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 ret

02471 * wait until there is a received character
02472 0F9B AD F6 OF93 zgetw bar zget try to get a character
02473 0F9D 24 FC OF98 bcc zgetw wait until successful
02474 0F9F 81 ret

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 ret carry set => data sent

02481 * wait until character has been output to buffer
02482 0FA7 AD F7 0FA0 zputw bar zput try to send
02483 0FA9 24 FC OFA7 bcc zputw wait until successful
02484 OFA8 81 ret
02485 OFAC end

Annex 3: Embedded software

- 174 -
### SYMBOl TABlE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Symbol 1</td>
</tr>
<tr>
<td>0002</td>
<td>Symbol 2</td>
</tr>
<tr>
<td>0003</td>
<td>Symbol 3</td>
</tr>
<tr>
<td>0004</td>
<td>Symbol 4</td>
</tr>
<tr>
<td>0005</td>
<td>Symbol 5</td>
</tr>
<tr>
<td>0006</td>
<td>Symbol 6</td>
</tr>
<tr>
<td>0007</td>
<td>Symbol 7</td>
</tr>
<tr>
<td>0008</td>
<td>Symbol 8</td>
</tr>
<tr>
<td>0009</td>
<td>Symbol 9</td>
</tr>
<tr>
<td>0010</td>
<td>Symbol 10</td>
</tr>
<tr>
<td>0011</td>
<td>Symbol 11</td>
</tr>
<tr>
<td>0012</td>
<td>Symbol 12</td>
</tr>
<tr>
<td>0013</td>
<td>Symbol 13</td>
</tr>
<tr>
<td>0014</td>
<td>Symbol 14</td>
</tr>
<tr>
<td>0015</td>
<td>Symbol 15</td>
</tr>
<tr>
<td>0016</td>
<td>Symbol 16</td>
</tr>
<tr>
<td>0017</td>
<td>Symbol 17</td>
</tr>
<tr>
<td>0018</td>
<td>Symbol 18</td>
</tr>
<tr>
<td>0019</td>
<td>Symbol 19</td>
</tr>
<tr>
<td>0020</td>
<td>Symbol 20</td>
</tr>
<tr>
<td>0021</td>
<td>Symbol 21</td>
</tr>
<tr>
<td>0022</td>
<td>Symbol 22</td>
</tr>
<tr>
<td>0023</td>
<td>Symbol 23</td>
</tr>
<tr>
<td>0024</td>
<td>Symbol 24</td>
</tr>
<tr>
<td>0025</td>
<td>Symbol 25</td>
</tr>
<tr>
<td>0026</td>
<td>Symbol 26</td>
</tr>
<tr>
<td>0027</td>
<td>Symbol 27</td>
</tr>
<tr>
<td>0028</td>
<td>Symbol 28</td>
</tr>
<tr>
<td>0029</td>
<td>Symbol 29</td>
</tr>
<tr>
<td>0030</td>
<td>Symbol 30</td>
</tr>
<tr>
<td>0031</td>
<td>Symbol 31</td>
</tr>
<tr>
<td>0032</td>
<td>Symbol 32</td>
</tr>
<tr>
<td>0033</td>
<td>Symbol 33</td>
</tr>
<tr>
<td>0034</td>
<td>Symbol 34</td>
</tr>
<tr>
<td>0035</td>
<td>Symbol 35</td>
</tr>
<tr>
<td>0036</td>
<td>Symbol 36</td>
</tr>
<tr>
<td>0037</td>
<td>Symbol 37</td>
</tr>
<tr>
<td>0038</td>
<td>Symbol 38</td>
</tr>
<tr>
<td>0039</td>
<td>Symbol 39</td>
</tr>
<tr>
<td>0040</td>
<td>Symbol 40</td>
</tr>
<tr>
<td>0041</td>
<td>Symbol 41</td>
</tr>
<tr>
<td>0042</td>
<td>Symbol 42</td>
</tr>
<tr>
<td>0043</td>
<td>Symbol 43</td>
</tr>
<tr>
<td>0044</td>
<td>Symbol 44</td>
</tr>
<tr>
<td>0045</td>
<td>Symbol 45</td>
</tr>
<tr>
<td>0046</td>
<td>Symbol 46</td>
</tr>
<tr>
<td>0047</td>
<td>Symbol 47</td>
</tr>
<tr>
<td>0048</td>
<td>Symbol 48</td>
</tr>
<tr>
<td>0049</td>
<td>Symbol 49</td>
</tr>
<tr>
<td>0050</td>
<td>Symbol 50</td>
</tr>
<tr>
<td>0051</td>
<td>Symbol 51</td>
</tr>
<tr>
<td>0052</td>
<td>Symbol 52</td>
</tr>
<tr>
<td>0053</td>
<td>Symbol 53</td>
</tr>
<tr>
<td>0054</td>
<td>Symbol 54</td>
</tr>
<tr>
<td>0055</td>
<td>Symbol 55</td>
</tr>
<tr>
<td>0056</td>
<td>Symbol 56</td>
</tr>
<tr>
<td>0057</td>
<td>Symbol 57</td>
</tr>
<tr>
<td>0058</td>
<td>Symbol 58</td>
</tr>
<tr>
<td>0059</td>
<td>Symbol 59</td>
</tr>
<tr>
<td>0060</td>
<td>Symbol 60</td>
</tr>
<tr>
<td>0061</td>
<td>Symbol 61</td>
</tr>
<tr>
<td>0062</td>
<td>Symbol 62</td>
</tr>
<tr>
<td>0063</td>
<td>Symbol 63</td>
</tr>
<tr>
<td>0064</td>
<td>Symbol 64</td>
</tr>
<tr>
<td>0065</td>
<td>Symbol 65</td>
</tr>
<tr>
<td>0066</td>
<td>Symbol 66</td>
</tr>
<tr>
<td>0067</td>
<td>Symbol 67</td>
</tr>
<tr>
<td>0068</td>
<td>Symbol 68</td>
</tr>
<tr>
<td>0069</td>
<td>Symbol 69</td>
</tr>
<tr>
<td>0070</td>
<td>Symbol 70</td>
</tr>
<tr>
<td>0071</td>
<td>Symbol 71</td>
</tr>
<tr>
<td>0072</td>
<td>Symbol 72</td>
</tr>
<tr>
<td>0073</td>
<td>Symbol 73</td>
</tr>
<tr>
<td>0074</td>
<td>Symbol 74</td>
</tr>
<tr>
<td>0075</td>
<td>Symbol 75</td>
</tr>
<tr>
<td>0076</td>
<td>Symbol 76</td>
</tr>
<tr>
<td>0077</td>
<td>Symbol 77</td>
</tr>
<tr>
<td>0078</td>
<td>Symbol 78</td>
</tr>
<tr>
<td>0079</td>
<td>Symbol 79</td>
</tr>
<tr>
<td>0080</td>
<td>Symbol 80</td>
</tr>
<tr>
<td>0081</td>
<td>Symbol 81</td>
</tr>
<tr>
<td>0082</td>
<td>Symbol 82</td>
</tr>
<tr>
<td>0083</td>
<td>Symbol 83</td>
</tr>
<tr>
<td>0084</td>
<td>Symbol 84</td>
</tr>
<tr>
<td>0085</td>
<td>Symbol 85</td>
</tr>
<tr>
<td>0086</td>
<td>Symbol 86</td>
</tr>
<tr>
<td>0087</td>
<td>Symbol 87</td>
</tr>
<tr>
<td>0088</td>
<td>Symbol 88</td>
</tr>
<tr>
<td>0089</td>
<td>Symbol 89</td>
</tr>
<tr>
<td>0090</td>
<td>Symbol 90</td>
</tr>
<tr>
<td>0091</td>
<td>Symbol 91</td>
</tr>
<tr>
<td>0092</td>
<td>Symbol 92</td>
</tr>
<tr>
<td>0093</td>
<td>Symbol 93</td>
</tr>
<tr>
<td>0094</td>
<td>Symbol 94</td>
</tr>
<tr>
<td>0095</td>
<td>Symbol 95</td>
</tr>
<tr>
<td>0096</td>
<td>Symbol 96</td>
</tr>
<tr>
<td>0097</td>
<td>Symbol 97</td>
</tr>
<tr>
<td>0098</td>
<td>Symbol 98</td>
</tr>
<tr>
<td>0099</td>
<td>Symbol 99</td>
</tr>
</tbody>
</table>

---

**Annex 3 : Embedded software**

- 175 -
Annex 3 : Embedded software:
program cclvl5fp;
{ Program to control current drives with messages sent from host or keyboard }  
{ same as cclvl5f.pas but with additional comments and revised layout }  
uses  
dos, crt, ccsup, unit_a0, unit_a1, unit_a2, unit_dd;  
const  
file_text = 'cclvl5f.dt'; {file for ASCII copy of calibration parameters}  
file_binary = 'cclvl5f.db'; {file for binary copy of calibration parameters}  
{ Development History  
1v1 Data types and variables for one drive on COM1  
1v2 Data Types for 1..3 drives but using just 1  
1v2a minor but essential corrections to 1v2  
1v3 includes routines to convert longint to hex  
1v4 major changes to allow 2 drives with scanning  
1v4a loop_state now initialised  
1v4b wt initialised  
1v5 auxiliary ADC channels read once every 2 seconds  
1v6 Windows and screen layout established  
1v6a some variables displayed  
1v7 Refined presentation  
1v7a Amplifier 1 shown to 4dp for testing  
1v8 Warning Box with Blink  
1v8a Difference between set point and precision adc is also averaged  
1v9 Host communications established  
1v10 Keyboard Entry primitives  
1v10b Main program revised into smaller procedures  
1v10d Keyboard entry almost complete  
1v10e Mean loop time and Loop monitor displayed. Ctrl Left Arrow available  
1v11 AIM Automatic determination of gains and offset for precision drive.  
1v11a Moving average filters rationalised  
1v11b AIM calibrate via internal source  
1v11c revised position of initial set_source(host);  
1v11d 1vlc has strange compile-time error  
1v11e may calibrate p_dac  
1v11f corrected incorrect scope of if in cal_p  
1v11g uses fp_p_dac in calibration  
1v11h tests levels with p_dac=X10000000...0 to X01111110..0  
1v12 AIM to create a file of calibration points.  
1v12a regression fits now volts per 1sb  
1v12b regression data stored in stats.x[] and y[]  
1v12c Revised Keywords. Reads calibration file.  
1v12d does open loop control of dacs using calibration information  
1v12d1 corrections to calculation of p_code in main control loop  
1v13 AIM trimming both manual and automatic. revised filenames  
1v13d length of filter for p_adc is altered depending on activity  
1v13e for calibration and after keyboard entry  
1v13f length=precision_adc_filter_length otherwise it is shorter  
1v13g heater is controlled and power estimated. files name.lpt  
1v13h contains resistance of coil then heater in ohms.  
1v13i find_nearest_index applies to stats_type generally  
1v13j keeps a copy of the fp_last_setpoint to decide if lock_up  
1v13k is needed;  
1v14 AIM revision of structure for parameters and data files  
1v14a Reports on time of last calibration  
1v14b reports current time  
1v14c white text now included in set_source  
1v15 AIM new procedure for displaying options  
1v15a stats_type now includes temperature in  
1v15b temperature during calibration stored in parameter block  
1v15c estimate of finish time for calibration  
1v15d drive 1 showed out of range value for fp_s_dac after calibration  
1v15e cal finished placed in set_dacs, revised keyboard routines  
1v15f major revision of keyboard entry  
1v15g beginnings of parameter form  
1v15h fully operational  
1v15f revised format of source code }
const
max_filter_length = 50;
loop_time_filter_length = 25;
stats_data_max = 200;
max_drive = 2;
number_of_parameters = 2;
space_for_parameters = 10;

t_ref = 70.0;  { Reference resistance assumed same for coil and heater }
v_max = 5;    { Maximum voltage from dac is 5 volts }
differential_range = 2 ; {During keyboard entry limit difference in drives}
max_error = 0.0000005;
ticks_Y_max = 18 * 5 ; { 18.2 ticks per second * number of seconds }
ticks_time_out_max = 18 * 3 ; { 18.2 ticks per second * number of seconds }
ticks_loop_max = 18;
ticks_clock_max = 18;  { up_date visible clock once per second }
wait = 0;
data_block_NULL : data_block_type = (v:0.0; vs:'0.0 '; vv:false);

type (Main)
float = real;
com_state_type = (idle, command_sent, ack_received, nack_received,
    valid_reply, time_out, command_received, reply_sent,
    valid_command);
com_type = record
    number, address, IRQ : word;
    last_command, last_reply : string;
    state : com_state_type;
    error : integer;
    ticks_timeout : longint;
    response : boolean;
end;

file_type = record
    inp, out : text;
end;
loop_state_type = (loop_idle,
    sent_X, received_X, sent_Y, received_Y, sent_Z, received_Z);

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

loop_time_type = record
    current_tick, last_tick : longint;
    filter : filter_type;
end;

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

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

calibrate_type = record
    state : cal_state_type;
    p_stats, s_stats : stats_type;
end;

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

param_type = record
    valid : boolean;
    p_stats, s_stats : stats_type;
    case integer of
        0 : (r coil, r_heater : data_block_type);
        1 : (data : param_array_type);
end;
drive_type = record
  number : integer;
  com : com_type;
  f : file_type;
  longint_precision_adc : longint;
  p_dac, s_dac, h_dac : string[4];
  p_code, s_code, h_code : longint;
  s_trim : integer;
  vref,
  fp_setpoint, fp_precision_adc,
  fp_p_dac, fp_s_dac, fp_h_dac,
  fp_agnd, fp_d1, fp_d2,
  fp_temperature_in, fp_temperature_out,
  fp_last_setpoint : float;
  nearest_index : integer;
  precision_adc, agnd, d1, d2, sensor_in, sensor_out : string;
  loop_state : loop_state_type;
  ticks_Y : longint;
  ticks_loop : longint;
  padc_filter : filter_type;
  fp_mean_error : float;
  loop_time : loop_time_type;
  monitor_count : integer;
  cal : calibrate_type;
  p : param_type;
  fp_power : float;
end;

host_type = record
  com : com_type;
  f : file_type;
  fp_setpoint : float;
  drive_number : integer;
  drive_id : char;
  fault : boolean;
end;

source_type = (host_pc, keyboard, calibrate);
warning_type = (exit, abort, calib);
keyboard_type = record
  data : data_block_type;
  x, y, f1, f2 : integer;
  index : integer;
  first : boolean;
  drive_data : array[1..max_drive] of data_block_type;
  kb_drive_number : integer;
end;

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

var {Main}
  drive : array[1..max_drive] of drive_type;
  host : host_type;
  kb : keyboard_type;
  done, flag : boolean;
  source : source_type;
  ticks_clock, tries, loop_count, reply_count : longint;
  ExitSave : pointer;
  j, precision_adc_filter_length : integer;
  ft : text;
  fp : file of param_type;

function sign(x : float) : integer;

- 179 -
Annex 4 : Slave PC software
begin
  if x>0 then
    sign := 1
  else
    if x<0 then
      sign := -1
    else
      sign := 0;
end;

function right(s : string; n:integer) : string;
var
  start : integer;
begin
  start := length(s)-n+1;
  if start<1 then
    start := 1;
  right := copy(s,start,n);
end;

function file_exists(name: string) : boolean;
var
  f : file;
begin
  assign(f,name);
  reset(f);
  close(f);
  file_exists := (IOResult.O) and (name<>"");
end;

procedure white_text;
begin
  TextAttr := white;
end;

procedure white_text_with_underline;
begin
  TextAttr := 9;
end;

procedure grey_text;
begin
  TextAttr := lightgray;
end;

procedure blinking_white_text;
begin
  TextAttr := $80+white;
end;

function ticks : longint;
begin
  ticks := longint(Ptr($40,$6C)) ; (Specific location holds real-time tick)
end;

procedure message(s : string; delay_in_milli_seconds : integer);
var
  xmin,xmax,width,depth : byte;
begin (message)
  width := length(s);
  if delay_in_milli_seconds<>0 then
    depth := 4
  else
    depth := 5;
  xmin := 40-width div 2 - 2;
  xmax := 40+width div 2 + 2;
  openwindow(xmin,10,xmax,10+depth," message ",white,$7A);
  put_string(2,2,s);
  if delay_in_milli_seconds=0 then
    begin
      put_string(2,3,"Press any key to continue");
      inkey
    end;
267     end
268   else
269     delay(delay_in_milli_seconds*1000);
270     closewindow;
271   end; {message}
272
273   function longint_to_hex(number: longint; count: integer): string;
274     var
275       r: string;
276       i: integer;
277     const
278       hex_table: array[0..15] of char = ('0', '1', '2', '3', '4', '5', '6', '7',
279         '8', '9', 'A', 'B', 'C', 'D', 'E', 'F');
280     begin {longint_to_hex}
281       r := '';
282       for i := count downto 1 do
283         begin
284           r := hex_table[number and 15] + r;
285           number := number shr 4;
286         end;
287       endint_to_hex := r;
288     end; {longint_to_hex}
289
290     function fptohex(number, scale: float; count: integer): string;
291     begin
292       fptohex := longint_to_hex(round(number * scale), count);
293     end;
294
295     function hex2char(hex_string: string): byte;
296     begin
297       case ch of
298         '0'..'9': hex2char := ord(ch) - ord('0');
299         'A'..'F': hex2char := ord(ch) - ord('A') + 10;
300         else
301           hex2char := 0;
302         end;
303     end;
304
305     function hex_to_longint(hex_string: string): longint;
306     var
307       t: longint;
308       i: integer;
309     begin
310       if length(hex_string) = 0 then
311         hex_to_longint := 0
312       else
313         begin
314           t := 0;
315           for i := 1 to length(hex_string) do
316             t := (t shl 4) + hex2char(hex_string[i]);
317           hex_to_longint := t
318         end;
319     end;
320
321     function padhex_to_fp(hex_string: string): float;
322     begin
323       padhex_to_fp := 5.0*(hex_to_longint(hex_string) - $400000)/$200000;
324     end;
325
326     function max132_hex_to_fp(hex_string: string; v: float): float;
327     var
328       t: string;
329     begin
330       if hex_string[1] = '0' then
331         t := '00'
332       else
333         t := 'FF';
334       max132_hex_to_fp := $v*(hex_to_longint(t + hex_string))/(655*512);
335     end;
336
337     procedure put_highlighted_string(x, y: integer; s: string; i: integer);
338     begin
339       grey_text;
340     end
- 181 - Annex 4: Slave PC software
334  put_string(x,y,s);
335  change_attribute(x+i-1,y,9);
336  white_text;
337  end;
338
339  procedure edit_keyboard_entry;
340  procedure delete_char( var d: data_block_type; i: integer);
341  begin
342    with d do vs := copy(vs,1,i-1)+right(vs,length(vs)-i)+'
343    end;
344
345  procedure insert_char( var d: data_block_type; var i:integer; code:integer);
346  begin
347    with d do
348    begin
349      vs :=copy(vs,1,i-1)+chr(code)+copy(vs,i,length(vs)-i);
350      if i<length(vs) then inc(i);
351    end;
352  end;
353
354  procedure validate_data( var d: data_block_type; f1,f2 : integer);
355  var
356    error_code: integer;
357  begin
358    with d do
359    begin
360      val(stripspaces(vs),v,error_code);
361      if error_code=0 then update(d,d.v,f1,f2);
362    end;
363  end;
364
365  begin{edit_keyboard_entry}
366    with kb do
367    begin
368      case last_key_code of
369        CHome : begin
370          data.vs := spaces(9);
371          index := 1;
372        end;
373        HOME : index := 1;
374        Endkey : index := length(data.vs);
375        Delkey : delete_char(data,index);
376        RightArrow : wrap_up(index,length(data.vs));
377        LeftArrow : wrap_down(index,length(data.vs));
378        BS : begin
379          if index>1 then dec(index);
380          delete_char(data,index);
381        end;
382        ord('-'),ord('.'),ord('('),ord('0').ord('9')
383          : begin
384            if first then
385            begin
386              data.vs := spaces(length(data.vs));
387              case last_key_code of
388                ord('-') : index := 1;
389                ord('.') : begin
390                  data.vs := spaces(f1-f2-2)+'0'+spaces(f2+1);
391                  index := f1-f2;
392                end;
393              else
394                index := 2;
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          end;
405        end;
- 182 -
procedure start_keyboard_entry(value: float; x0, y0, f1, f2: integer);
begin
  with kb do
  begin
    f1 := f1; f2 := f2; x := x0; y := y0;
    update(data.value, f1, f2);
    index := length(data.vs);
    first := true;
    put_highlighted_string(x, y, data.vs, index);
    data.vv := false;
  end;
end;

procedure start_drive_entry(n: integer);
begin
  with kb do
  begin
    kb.drive_number := n;
    start_keyboard_entry(drive[n].fp_setpoint, screen_offset_fp[n], 8, 9, 6);
  end;
end;

procedure set_filter(var f: filter_type;
  new_value: float;
  new_length, new_guard: integer);
var
  i: integer;
begin (set_filter)
  with f do
  begin
    length := new_length;
    guard := new_guard;
    for i := 1 to length do history[i] := new_value;
    index := 1;
    count := 0;
    mean := new_value;
    valid := false;
  end;
end (set_filter);

procedure apply_filter(var f: filter_type; new_value: float);
var
  i: integer;
  sum: float;
begin (apply_filter)
  with f do
  begin
    history[index] := new_value;
    inc(count);
    valid := valid or (count>=length+guard);
    wrap_up(index, length);
    sum := 0;
    for i := 1 to length do sum := sum+history[i];
    mean := sum/length;
  end;
end (apply_filter);

procedure set_stats(var s: stats_type);
var
  i: integer;
begin (set_stats)
  with s do
  begin
    n := 0; sx := 0; sx2 := 0; sy := 0; sy2 := 0; sxy := 0;
    for i := 1 to stats_data_max do
      begin
        n := n+1;
        sx := sx + i*sx2;
        sx2 := sx2 + i*i;
        sy := sy + i*sy2;
        sy2 := sy2 + i*i;
        sxy := sxy + i*sx2;
      end;
  end;
end;
procedure enter_stats(var s: stats_type; xd, yd: float);
begin
with s do
begin
inc(n);
sx := sx + xd;
sx2 := sx2 + sqr(xd);
sy := sy + yd;
sy2 := sy2 + sqr(yd);
sxy := sxy + xd * yd;
x[n] := xd;
y[n] := yd;
end;
end;

procedure apply_stats(var s: stats_type);
var
i: integer;
begin
with s do
if n > 1 then
begin
xb := sx/n; yb := sy/n;
var x := (sx2 - sqr(sx)/n)/n;
var y := (sy2 - sqr(sy)/n)/n;
b := (sxy - (sx * sy)/n) / (sx2 - sqr(sx)/n);
a := 1/n * (sy - bx * sx);
cov := 0;
for i := 1 to n do
error[i] := y[i] - (a + b * x[i]);
cov := cov + (x[i] - bx) * (y[i] - yb);
end;
cov := cov/n;
r := cov / sqrt(var * var_y);
end
else
begin
a := 0; b := y[1]; error[1] := 0; r := 1;
end;
end;

procedure report_time_of_last_calibration;
var
time: longint;
DT: DateTime; // from dos unit
begin
GetFTime(fp, time);
UnpackTime(time, DT);
grey_text;
with DT do put_string(42, 21, 'Last calibration ' +
copy(time_string(Hour, Min, Sec), 1, 5) +
' on ' + date_string(Year, Month, Day));
put_string((42, 23, spaces(37));
white_text;
end;

procedure report_current_time;
var
Year, Month, Day, DayOfWeek, Hour, Min, Sec, Sec100: word;
begin
if abs(ticks - ticks_clock) > ticks_clock_max then
begin
ticks_clock := ticks;
GetDate(Year, Month, Day, DayOfWeek);
GetTime(Hour, Min, Sec, Sec100);
grey_text;
put_string(50, 22, 'Currently ' + copy(time_string(Hour, Min, Sec), 1, 5) +
'on ' + date_string(Year, Month, Day));
procedure write_all_parameters;
procedure write_stats(var s : stats_type);
var
i : integer;
begin
with s do
begin
  writeln(ft,'Temperature (in) = ',temperature_in:7:4);
  writeln(ft,'Number= ',n);
  for i := 1 to n do
    writeln(ft,'x[i]:11:1, y[i]:12:8, error[i]:12:8);
  writeln(ft,'mean x = ',xb:11:1, mean y := ',yb:12:8);
  writeln(ft,'a = ',a,' b = ',b,' r = ',r);
end;
end;

var
i : integer;
begin {writeallparameters}
message('Saving parameters and calibration data',3);
assign(ft,file_text);
rewrite(ft);
assign(fp,file_binary);
rewrite(fp);
for i := 1 to max_drive do
  with drive[i] do
    begin
      writeln(ft,'Drive = ',i);
      p.p_stats := cal.p_stats;
      p.s_stats := cal.s_stats;
      writeln(ft,'Coi1 Resistance = ',r.coi1.vs);
      writeln(ft,'Heater Resistance = ',r.heater.vs);
      writeln(ft,'Primary OAC statistics');
      writestats(p.p_stats);
      writeln(ft,'Secondary OAC statistics');
      writestats(p.s_stats);
      write(fp,p);
      report_time_of_last_calibration;
    end;
  close(ft);
  close(fp);
end; {write_all_parameters}

procedure restore_all_parameters;
var
i : integer;
begin {restoreallparameters}
for i := 1 to max_drive do
  begin
    drive[i].p.valid := false;
    for j := 1 to space_for_parameters do
      drive[i].p.data[j] := data_block_null;
    white_text;
    if file_exists(file_binary) then
      begin
        assign(fp,file_binary);
        reset(fp);
        message('Restoring parameters and calibration data',3);
        for i := 1 to max_drive do
          with drive[i] do
            begin
              read(fp,p);
              cal.p.stats := p.p.stats;
              cal.s.stats := p.s.stats;
            end;
        report_time_of_last_calibration;
        close(fp);
      end;
    else
      begin
        white_text;
      end;
  end;
end; {restore_all_parameters}
for i := 1 to max_drive do drive[i].p.valid := false;
put_string(42,21,'No Previous Calibration');
end;
end; (restore_all_parameters)

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

procedure prepare_com_file( var f : text; com : com_type;
direction : direction_type);
var
error : word;
options : _OptionRecord;
begin (prepare_com_file)
with _StdOpen do
begin
InQueue Size := 1024;
Out Queue Size := 1024;
Port Address := com.address;
Int Level := com.IRQ;
end;
assignDD(f,com.number);
if direction = input then
  reset(f)
else
  rewrite(f);
error := RetOpOO(f,options);
if (error<>0)then
  writeln('Unable to read port settings - error=',error);
with options do
begin
  BaudRate :=9600;
  Parity := 0;
  DataBits := 8;
  StopBits := 1;
  RequireCTS := false;
end;
error := SetOpOO(f,options);
if (error<>0)then
  writeln('Unable to set port settings - error=',error);
end; (prepare_com_files)
begin
prepare_com_file(ft.inp,com,input);
prepare_com_file(ft.out,com,output);
end; (prepare_com_files)

procedure initialise_com( var com : com_type; channel : integer);
begin
with com do
begin
  number := com_ref[channel].number;
  address := com_ref[channel].address;
  IRQ := com_ref[channel].IRQ;
  last_command := "";
  last_reply := ";
  state := idle;
  response := true;
end;

procedure initialise_loop_time(var d : drive_type);
begin
with d.loop_time do
begin
  current_tick := ticks;
  set_filter(filter,0,loop_time_filter_length,2);
end;
end;

procedure initialise_drive(var dr : drive_type; channel : integer);
var
  i : integer;
begin
with dr do

begin

number := channel;
initialise_com(com,channel);
prepare_com_files(f,com);
case channel of
1 : drive[1].vref := 0.61772; {voltage across R44 of Drive 1}
2 : drive[2].vref := 0.61700; {voltage across R44 of Drive 2}
end;
loop_state := loop_idle;
longint_precision_adc := 0;
s_trim := 0;
f_prime := 0;
f_prime_last_setpoint := 1.0e10; {impossibly large to force lookup}
f_prime_precision_adc := 0;
f_prime_p_dac := 0.0;
f_prime_s_dac := 0.0;
f_prime_h_dac := 0.0;
f_prime_agnd := 0.0;
f_prime_d1 := 0.0;
f_prime_d2 := 0.0;
f_prime_temperature_in := 0.0;
f_prime_temperature_out := 0.0;
precision_adc := ";";
agnd := ";
d1 := ";
d2 := ";
sensor_in := ";
sensor_out := ";
loop_state := loop_idle;
ticks_y := ticks-ticks_y_max;
ticks_loop := ticks-ticks_loop_max;
set_filter(padc_filter,f_prime_setpoint,precision_adc_filter_length,2);
initialise_loop_time(dr);
cal.state := cal_idle;
monitor_count := 1;
end;
end; {initialise_drive}

procedure initialise_host(var hs : host_type; channel : integer);
begin
with hs do
begin
initialise_com(com,channel);
prepare_com_files(f,com);
f_prime_setpoint := 0;
fault := false;
end;
end;
procedure send_drive_command(var dr : drive_type; message : string);
begin
with dr do
begin
{$I-}
write(f.out,message+chr(13);
{$I+}
with com do
begin
last_command := message;
last_reply := ";";
error := IO Result;
state := command_sent;
ticks_timeout := ticks;
end;
end;
end; {send_drive_command}
procedure send_host_reply(var hs : host_type; message : string);
begin
with hs do
begin
{$I-}
write(f.out,'K'+message+chr(04));
{$I+}
end;
end;

Annex 4 : Slave PC software
procedure get_drive_reply(var dr: drive_type);
    const
        ack_string = 'Y'+chr(13)+chr(10);
        nack_string = 'N'+chr(13)+chr(10);
        var
            ch: char;
            len: integer;
            ts: string;
    begin
        with dr.com do begin
            repeat
                read(dr.f.inp,ch);
                if ord(ch)<>26 then
                    begin
                        last_reply := last_reply+ch;
                        if right(last_reply,length(ack_string))=ack_string then
                            state := ack_received;
                        if right(last_reply,length(nack_string))=nack_string then
                            state := nack_received;
                        end;
                    until ord(ch)=26;
                    if state=ack_received then
                        begin
                            if pos(last_command,last_reply)=1 then
                                begin
                                    last_reply := copy(last_reply,length(last_command)+3,
                                           length(last_reply)-length(last_command)-5);
                                    state := valid_reply;
                                    inc(reply_count);
                                    if not response then
                                        begin
                                            response := true;
                                            put_string(screen_offset_hex[dr.number],2,spaces(12));
                                        end;
                                end;
                        end;
                        if abs(ticks-ticks_timeout)>ticks_timeout_max then
                            begin
                                send_drive_command(dr,dr.com.last_command);
                                response := false;
                                gotoxy(screen_offset_hex[dr.number],2);
                                blinking_white_text;
                                Write('No Response');
                                white_text;
                            end;
                        end;
                    end;
                end; {get_drive_reply}

procedure get_host_command(var hs: host_type);
    const
        end_string = chr(4);
        var
            ch: char;
            len: integer;
            ts: string;
    begin
        with hs.com do begin
            repeat
                read(hs.f.inp,ch);
                if ord(ch)<>26 then
                    begin
                        if (last_command=';') then begin
                            last_command := ''; 
                            last_reply := message;
                        end;
                        if state = reply_sent then
                            state := reply_waiting; 
                            last_reply := message;
                            error := IOResult;
                        end;
                        if state = reply_waiting then begin
                            if (last_reply=';') then begin
                                last_reply := message;
                                error := IOResult;
                            end;
                            if (last_reply=';') then begin
                                last_reply := message;
                                error := IOResult;
                            end;
                        end;
                        if state = reply_sent then begin
                            if (last_reply=';') then begin
                                last_reply := message;
                                error := IOResult;
                            end;
                            if (last_reply=';') then begin
                                last_reply := message;
                                error := IOResult;
                            end;
                        end;
                    end;
                end;
            end; {send_host_reply}

procedure get_drive_reply(var dr: drive_type);
    const
        ack_string = 'Y'+chr(13)+chr(10);
        nack_string = 'N'+chr(13)+chr(10);
        var
            ch: char;
            len: integer;
            ts: string;
    begin
        with dr.com do begin
            repeat
                read(dr.f.inp,ch);
                if ord(ch)<>26 then
                    begin
                        last_reply := last_reply+ch;
                        if right(last_reply,length(ack_string))=ack_string then
                            state := ack_received;
                        if right(last_reply,length(nack_string))=nack_string then
                            state := nack_received;
                        end;
                    until ord(ch)=26;
                    if state=ack_received then
                        begin
                            if pos(last_command,last_reply)=1 then
                                begin
                                    last_reply := copy(last_reply,length(last_command)+3,
                                           length(last_reply)-length(last_command)-5);
                                    state := valid_reply;
                                    inc(reply_count);
                                    if not response then
                                        begin
                                            response := true;
                                            put_string(screen_offset_hex[dr.number],2,spaces(12));
                                        end;
                                end;
                        end;
                        if abs(ticks-ticks_timeout)>ticks_timeout_max then
                            begin
                                send_drive_command(dr,dr.com.last_command);
                                response := false;
                                gotoxy(screen_offset_hex[dr.number],2);
                                blinking_white_text;
                                Write('No Response');
                                white_text;
                            end;
                        end;
                    end;
                end;
            end; {get_drive_reply}
last_command := last_command+ch;
if right(last_command,length(end_string))=end_string then
  state := command_received;
end;
until ord(ch)=26;
if (state=command_received) and
  (length(last_command)=13) and
  (copy(last_command,4,1)="K") and
  (copy(last_command,4,1)="V") then
  with hs do
    begin
      drive_id := hs.com.last_command[3];
      if ord(drive_id)>128 then
        drive_number := ord(drive_id)-$80
      else
        drive_number := ord(drive_id)-ord('0');
      fp_setpoint := 5.0/8388608*hex_to_longint(copy(hs.com.last_command,5,8));
      if (drive_number in[1..2]) and
        (fp_setpoint<5.0) and
        (fp_setpoint>-5.0) then
        state := valid_command;
    end;
end;{get_host_command}

procedure put_shaded_number(x,y:integer; number:float;
  f1,f2:integer; flag:boolean);
begin
  if flag then
    whitetext
  else
    greytext;
  put_number(x,y,number,f1,f2);
  whitetext;
end;{put_shaded_number}

function find_nearest_index(s:stats_type; yd:float) : integer;
var
  i,j:integer;
  min_difference,difference : float;
begin{find_nearest_index}
  min_difference := $e5;
  with s do
    begin
      for i := 1 to n do
        begin
          difference := abs(s.y[i]-yd);
          if difference<min_difference then
            begin
              j := i;
              min_difference := difference;
            end;
        end;
    end;
  find_nearest_index := j;
end;{find_nearest_index}

procedure control_loop;
var
  drive_number :integer;
begin{calc_and_disp_aux_adc}
  with drive[drive_number] do
    begin{calc_and_disp_aux_adc}
      grey_text;
      put_string(screen_offset_hex[drive_number],15,d1);
      put_string(screen_offset_hex[drive_number],16,d2);
      put_string(screen_offset_hex[drive_number],17,sensor_in);
      put_string(screen_offset_hex[drive_number],18,sensor_out);
      white_text;
      fp_agnd := max132_hex_to_fp(agnd.vref);
      fp_d1 := (max132_hex_to_fp(d1.vref)-fp_agnd)*24.2;
      fp_d2 := (max132_hex_to_fp(d2.vref)-fp_agnd)*24.2;
end;
fp_temperature_in := (max132_hex_to_fp(sensor_in,vref)-fp_agnd)*100.0;
fp_temperature_out := (max132_hex_to_fp(sensor_out,vref)-fp_agnd)*100.0;
put_number(screen_offset_fp[drive_number]-1,15,fp_d1,8,4);
pnput_number(screen_offset_fp[drive_number]-1,16,fp_d2,8,4);
pnput_number(screen_offset_fp[drive_number],17,fp_temperature_in,5,2);
pnput_number(screen_offset_fp[drive_number],18,fp_temperature_out,5,2);
end;

end; (calc_and_disp_aux_adc)

procedure calc_and_disp_precision_adc;
var
i : integer;
sum : float;
begin {calc_and_dispprecisionadc}
with drive[drive_number] do
begin
grey_text;
pnput_string(screen_offset_hex[drive_number],6,precision_adc);
white_text;
pnput_number(screen_offset_fp[drive_number],5,fp_setpoint,9,6);
apply_filter(padc_filter,fp_precision_adc);
fp_mean_error := padc_filter.mean-fp_setpoint;
pnput_shaded_number(screen_offset_fp[drive_number],7,fp_mean_error,9,6,padc_filter.valid);
wrapup(monitor_count,4);
pnput_string(screen_offset_fp[drive_number]+10,6,monitor_char[monitor_count]);
end;
end; {calc_and_disp_precision_adc}

procedure display_settings;
var
sum : longint;
i : integer;
begin {displaysettings}
with drive[drive_number] do
begin
grey_text;
pnput_string(screen_offset_hex[drive_number],10,p dac);
pnput_string(screen_offset_hex[drive_number],11,s dac);
pnput_string(screen_offset_hex[drive_number]+5,11,

longint_to_hex(s_trim,2));
white_text;
pnput_number(screen_offset_fp[drive_number],10,fp_p dac,8,5);
pnput_number(screen_offset_fp[drive_number],11,fp_s dac,8,5);
pnput_number(screen_offset_fp[drive_number],12,fp_h dac,8,5);
pnput_number(screen_offset_fp[drive_number],13,fp_power,5,2);
with loop_time do
begin
last_tick := current_tick;
current_tick := ticks;
apply_filter(filter.abs(current_tick-last_tick));
pnput_shaded_number(screen_offset_fp[drive_number],19,

filter.mean/18.2,5,2,filter.valid);
end;
end;
end; {display_settings}

procedure auto_trim(var d : drive_type);
var
difference : float;
begin
with d do
begin
if padc_filter.valid then
begin

difference := (fp_setpoint-padc_filter.mean);
if abs(difference)>max_error then
begin
s_trim := s_trim+sign(difference);
set_filter(padc_filter,fp_setpoint,4,0);
end;
end;

Annex 4 : Slave PC software
procedure maintain_constant_power(var d: drive_type);

begin
  max_power := sqrt(5/r_ref)*p.r_coil.v;
  fp_h_dac := r_ref*sqrt(max_power-sqr(fp_setpoint/r_ref)*p.r_coil.v)/p.rheater.v;
  fp_power := sqrt(fp_h_dac/r_ref)*p.r.heater.v+
              sqr(fp_setpoint/r_ref)*p.r_coil.v;
end;

procedure set_dacs(var d: drive_type);

begin
  with d do
  begin
    case cal.state of
    cal_idle, cal_finished :
      begin
        if fp_setpoint<>fp_last_setpoint then
          nearest_index := find_nearest_index(cal.p_stats,fp_setpoint);
        fp_last_setpoint := fp_setpoint;
        p_code := round(cal.p_stats.x[nearest_index]);
        s_code := round((fp_setpoint-cal.p_stats.y[nearest_index])/cal.s_stats.b)+strim;
        fppdac := p_code*5.0732768;
        fp_s_dac := s_code*S.0/32768;
        maintain_constant_power(d);
        h_code := round(fphdac/5.0*32768);
        autotrim(d); --
      end;
    cal_p :
      begin
        fppdac := fp_setpoint;
        fp_s_dac := 0;
        maintain_constant_power(d);
        h_code := round(fphdac/5.0*32768);
        p_code := 0;
        s_code := round(fppdac/S.0*32768);
      end;
    cal_s :
      begin
        fppdac := 0;
        fp_s_dac := fp_setpoint*64;
        maintain_constant_power(d);
        h_code := round(fphdac/5.0*32768);
        p_code := 0;
        s_code := round(fpsdac/S.0*32768);
      end;
    end; --
  end;
end;

begin (control_loop)
  for drive_number := 1 to 2 do
    with drive[drive_number] do
      begin
        case drive[drive_number].loop_state of
        loop_idle :
          begin
            if abs(ticks-ticks_loop) > ticks_loop_max then
              begin
                send_drive_command(drive[drive_number],'x');
                loop_state := sent_X;
                ticks_loop := ticks;
          end;
    end; --
end; --

Annex 4: Slave PC software
begin
get_drive_reply(drive[drive_number]);
if com.state=valid_reply then
begin
precision_adc := com.last_reply;
fp_precision_adc := p_adc.hex_to_fp(precision_adc);
calc and disp precision adc;
loop_state := received_x;
end;
end;
end;
end;
sent_x :
begin
get_drive_reply(drive[drive_number]);
if com.state=valid_reply then
begin
precision_adc := com.last_reply;
fp_precision_adc := p_adc.hex_to_fp(precision_adc);
calc and disp precision adc;
loop_state := received_x;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;
end;{control_loop}
procedure display_options(x,y:integer; s:string);
var
  i : integer;
begin (display_options)
  grey_text;
  put_string(x,y,'Options');
gotoxy(x+y+2);
for i := 1 to length(s) do
begin
  if s[i] in ['A'..'Z'] then
    white_text_with_underline
else
grey_text;
write(s[i]);
end;
end; {display_options}

procedure draw_main_form;
begin
  clrscr;
  horizontal_line(1,78,20,1,TextAttr);
  vertical_line(40,21,23,1,TextAttr);
  put_string2(31,2,'DRIVE 1',59,2,'DRIVE 2');
  put_string2(26,3,'Floating',54,3,'Floating');
  put_string2(3,5,'Set Point',20,5,'V');
  put_string2(3,6,'Precision ADC',20,6,'V');
  put_string2(3,7,'Mean Difference',20,7,'V');
  put_string2(3,8,'Keyboard Entry',20,8,'V');
  put_string2(3,10,'Primary DAC',20,10,'V');
  put_string2(3,11,'Secondary DAC',20,11,'V');
  put_string2(3,12,'Heater_DAC',20,12,'V');
  put_string2(3,13,'Power',20,13,'W');
  put_string2(3,15,'Amplifier',20,15,'V');
  put_string2(3,16,'Amplifier 2',20,16,'V');
  put_string2(3,17,'Temperature in',19,17,'C');
  put_string2(3,18,'Temperature out',19,18,'C');
  put_string2(3,19,'Mean Loop Time',20,19,'s');
  display_options(3,21,'Abort Calibrate Exit Host Keyboard');
  grey_text;
  put_string(3,2,'Source=');
  put_string2(39,3,'HEX',67,3,'HEX');
  whitetext;
end; {draw_main_form}

procedure cursor_control(a,b : byte);
var
  Regs : Registers;
begin
  Regs.AH := 1;
  Regs.CH := a;
  Regs.CL := b;
  Intr(16,Regs);
end;

procedure cursor_on;
begin
  cursor_control(11,12);
end;

procedure cursor_off;
begin
  cursor_control($20,12);
end;

{$F+}
procedure MyExit;
begin
  ExitProc := ExitSave;
  close(drive[1].f.inp);
  close(drive[1].f.out);
  close(drive[2].f.inp);
  close(drive[2].f.out);
  cursor_on;
  closewindow;
  clrscr;
end; {MyExit}
{$F-}

function warning(typ : warning_type) : boolean;
const
  message : array[0..2] of string[25] =
  ('OK to Exit? Y/N',
   'OK to Abort? Y/N',
   'OK to Calibrate Y/N');
var
  ...
begin {warning}
width := length(message[ord(typ)]);
depth := 4;
xmin := 40 - width div 2 - 2;
xmax := 40 + width div 2 + 2;
openwindow(xmin,10,xmax,10+depth,'warning',white,$FA);
put_string(2,2,message[ord(typ)]);
repeat
  inkey;
  case last_key_code of
    ord('y'),ord('Y') : begin
      reply := true;
      done := true;
    end;
    ord('n'),ord('N') : begin
      reply := false;
      done := true;
    end;
  end;
untildone;
warning := reply;
closewindow;
end; {warning}

procedure flush_host(var hs : host_type);
var
  ch : char;
begin {flush_host}
  repeat
    read(hs.f.inp,ch);
  until ord(ch)=26;
end; {flush_host}

function test_all_parameters:boolean;
var
  i,j : integer;
  flag:boolean;
begin {testallparameters}
  flag := true;
  for i := 1 to max_drive do
    with drive[i].p do
      valid := true;
      for j := 1 to number_of_parameters do
        valid := valid and data[j].vv;
      flag := flag and valid;
    end;
  test_all_parameters := flag;
end; {test_all_parameters}

procedure edit_parameters;
const
  screen_offset_param : array[1..max_drive] of integer = (29,45);
var
  i,j : integer;
  local_done:boolean;
begin {drawform}
clearscr;
horizontal_line(1,57,7,1,TextAttr);
put_string2(30,2,'DRIVE 1',46,2,'DRIVE 2');
put_string2(3,4,'Coil Resistance',23,4,chr(234));
put_string2(3,5,'Heater Resistance',23,5,chr(234));
display_options(3,8,'Continue');
for i := 1 to max_drive do
  for j := 1 to number_of_parameters do
    with drive[i].p.data[j] do
      if vv then
        put_string(screen_offset_param[i],3+j,vs);
end; {drawform}
procedure next_entry;
begin
with drive[i].p do
begin
    if not data[j].vv then
        update(data[j],0,7,3);
        start_keyboard_entry(data[j],v,screen_offset_param[i],j+3,7,3);
    end;
end; {next_entry}

procedure cycle_pointers;
begin
    case last_key_code of
        CR : begin
            wrap_up(j,number_of_parameters);
            if j=1 then wrap_up(1,max_drive);
            end;
        TAB,STAB : wrap_up(i,max_drive);
        UpArrow : if j>1 then dec(j);
        DownArrow : if j<number_of_parameters then inc(j);
        end;
    next_entry;
end; {Cycle_pointers}

begin {edit_parameters}
    openwindow(11,5,69,16,'Parameters',white,white);
    draw form;
    local done := false;
    i := 1;
    j := 1;
    next_entry;
    repeat
        get_key;
        case last_key_code of
        ord('c'),ord('C') : begin
            local done := test_all_parameters;
            if not local done then
                message('All parameters must be defined',2);
            end;
        else
            edit_keyboard_entry;
        end;
        caselast_key_code of
        CR,TAB,STAB,DownArrow,UpArrow :
        if kb.data.vv then
            begin
                drive[i].p.data[j] := kb.data;
                cycle_pointers
            end
        else
            message('Invalid Entry',1);
        end;
    until local done;
    closewindow;
end; {edit_parameters}

procedure execute_host_command(var hs : host_type);
begin
    case hs.com.state of
    valid_command :
        begin
            drive[hs.drive_number].fp_setpoint := hs.fp_setpoint;
            { possibly change length of filter depending on the difference
              between hs.fp_setpoint and drive[?].fp_setpoint }
            set_filter(drive[hs.drive_number].padc_filter,
                        drive[hs.drive_number].fp_setpoint,
                        precision_adc_filter_length,2);
            send_host_reply(hs,hs.drive_id);
        end; 
    command_received :
        begin
            send_host_reply(hs,'E');
        end;
end; {execute_host_command}

- 195 - Annex 4 : Slave PC software
procedure set_source(src : source_type);

i : integer;

begin (set_source)
  source := src;
  for i := 1 to max_drive do
    put_string(screen_offset_fp[i],8,spaces(12));
  white_text;
  case source of
    host_pc : begin
      put_string(10,2,'HOST PC ');
      flush_host(host);
      for i := 1 to max_drive do
        with drive[i] do
          begin
            cal.state := cal_idle;
            fp_setpoint := 0;
            set_filter(padc_filter,fp_setpoint,
                        precision_adc_filter_length,3);
            initialise_loop_time(drive[i]);
          end;
      end;
    keyboard : put_string(10,2,'KEYBOARD ');
    calibrate : begin
      put_string(10,2,'CALIBRATION');
      edit_parameters;
      for i := 1 to max_drive do
        drive[i].cal.state := cal_start;
      end;
    end;
  end; {set_source}

procedure keyboard_activity;

i : integer;

s : string;

begin (keyboard_activity)
  get_key;
  case source of
    keyboard :
      case last_key_code of
        ESC : set_source(host_pc);
        ord('c'),ord('C') : if Warning(callb) then
          set_source(calibrate);
        ord('e'),ord('E') : done := warning(exit);
        ord('h'),ord('H') : set_source(host_pc);
        0 : { no key => no action };
      else
        with kb do
          begin
            edit_keyboard_entry;
            if data.vv then
              begin
                case kb_drive_number of
                  1 : begin
                    drive_data[1] := data;
                    start_drive_entry(2);
                  end;
                  2 : begin
                    if last_key_code=TAB then
                      start_drive_entry(1)
                    else
                      if abs(drive_data[1].v-data.v)<=
                        differential_range then
                        begin
                          drive[1].fp_setpoint := drive_data[1].v;
                          set_filter(drive[1].padc_filter,
                        drive[1].fp_setpoint,
                        precision_adc_filter_length,3);
                          set_filter(drive[2].padc_filter,
                        drive[2].fp_setpoint,
                        precision_adc_filter_length,3);
                        end;
                    end;
                  end;
                end;
              end;
            end;
          end;
      end;
    end;
end;
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 host\_pc:
1408 case last\_key\_code of
1409 ord('c'),ord('C') : if warning(calib) then
1410 set\_source(calibrate);
1411 ord('e'),ord('E') : done := warning(exit);
1412 ord('k'),ord('K') :
1413 begin
1414 set\_source(keyboard);
1415 start\_drive\_entry(1);
1416 end;
1417 end;
1418 calibrate:
1419 case last\_key\_code of
1420 ord('a'),ord('A') : if warning(abort) then
1421 begin
1422 if file\_exists(file\_binary) then
1423 restore\_all\_parameters;
1424 set\_source(host\_pc);
1425 end
1426 else
1427 message('Cannot abort first calibration',wait);
1428 end;
1429 ord('e'),ord('E') : done := warning(exit);
1430 end;
1431 end;
1432 end;
1433 end; (keyboard\_activity)
1434
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
1443 procedure define\_setpoint(var d : drive\_type; k : integer);
1444 function fun(bits,limit,count : integer) : float;
1445 const
1446 one: longint = 1; { force long integer calculations }
1447 begin{fun}
1448 fun := (one shl (16-bits))*5/32768*(count-(limit div 2));
1449 end;{fun}
1450 begin{define\_setpoint}
1451 with d do
1452 begin
1453 case cal\_state of
1454 cal\_p : fp\_setpoint := fun(p\_bits,p\_limit,cal\_p\_stats.n);
1455 cal\_s : fp\_setpoint := fun(s\_bits,s\_limit,cal\_s\_stats.n)/64;
1456 end;
1457 set\_filter(padc\_filter,fp\_setpoint,precision\_adc\_filter\_length*k,guard);
1458 end;
1459 end;{define\_setpoint}
1460
1461 procedure display\_finish\_time;
1462 var
1463 number\_of\_samples,ts,tm,th,fth,ftm : longint;
1464 Hour,Min,Sec,Sec100 : word;
1465 begin{display\_finish\_time}
1466 number\_of\_samples := 4*precision\_adc\_filter\_length+2*guard +
1467 (precision\_adc\_filter\_length+guard)*(p\_limit+s\_limit);
if drive[1].loop_time.filter.valid then
    ts := round(drive[1].loop_time.filter.mean/18.2*number_of_samples)
else
    ts := round(number_of_samples*ticks_loop_max/18);
tm := ts div 60;
if (ts mod 60)>30 then
    inc(tm);
th := tm div 60;
GetTime(Hour, Min, Sec, Sec100);
ftm := (Min+tm) mod 60;
fh := (Hour+fh+(Min+tm) div 60) mod 24;

put_string(42, 23, 'Calibration until '+copy(time_string(fth, ftm, 0), 1, 5)+
            ' approx.');

var
i : integer;
begin {calibrate_cycle}
    for i := 1 to max_driver do
        with drive[i].cal do
            case state of
                cal_idle :
                    set_stats(p_stats);
                    state := cal_p;
                begin_set_point(drive[i], 2);
                drive[i].strim := 0;
                if i=1 then
                    display_finish_time;
                end;
                cal_p :
                    begin
                        if drive[i].padc_filter.valid then
                            begin
                                enter_stats(p_stats, drive[i].p_code,
                                            drive[i].padc_filter.mean);
                                if p_stats.n=p_limit/2 then { record in middle of run }
                                    p_stats.temperature_in := drive[i].fp_temperature_in;
                                if p_stats.n<=p_limit then
                                    define_set_point(drive[i], 1)
                                else
                                    begin
                                        apply_stats(p_stats);
                                        set_stats(s_stats);
                                        state := cal_s;
                                        define_set_point(drive[i], 2);
                                    end;
                            end;
                        end;
                cal_s :
                    begin
                        if drive[i].padc_filter.valid then
                            begin
                                enter_stats(s_stats, drive[i].s_code,
                                            drive[i].padc_filter.mean);
                                if s_stats.n=s_limit/2 then { record in middle of run }
                                    s_stats.temperature_in := drive[i].fp_temperature_in;
                                if s_stats.n<=s_limit then
                                    define_set_point(drive[i], 1)
                                else
                                    begin
                                        apply_stats(s_stats);
                                        state := cal_finished;
                                        with drive[i] do
                                            begin
                                                drive[i].fp_setpoint := 0;
                                                set_filter(padc_filter, fp_setpoint, 
                                                        precision_adc_filter_length, 3);
                                            end;
                                    end;
                        end;
        end;
end; {display_finish_time}
procedure general_initialise;
begin
ExitSave := ExitProc;
ExitProc := @MyExit;
cursor off;
openwindow(1,1,80,25," COIL CONTROLLER ",white,white);
draw_main_form;
loop_count := 0;
reply_count := 0;
tries := 0;
done := false;
precision_adc_filter_length := 10;
ticks_clock := ticks;
end; {general_initialise}

begin {main}
generalinitialise;
initialise_drive(drive[1],1);
initialise_drive(drive[2],2);
initialise_host(host,3);
restore_all_parameters;
if drive[1].p.valid and drive[2].p.valid then
  set_source(host_pc)
else
  set_source(calibrate);
repeat
  keyboard_activity;
  case source of
    host_pc : begin
      get_host_command(host);
      execute_host_command(host);
    end;
    calibrate : begin
      if (drive[1].cal.state=cal_finished) and
          (drive[2].cal.state=cal_finished) then
        begin
          drive[1].p.valid := true;
          drive[2].p.valid := true;
          write_all_parameters;
          set_source(host_pc);
        end
      else
        calibrate_cycle
    end;
  end;
  control_loop;
  inc(tries);
  { gotoxy(10,21); write(loop_count:6,reply_count:6,tries:8); }
  report_current_time;
  until done;
clrscr;
halt; {close files/coms via MyExit}
end. {main}
type
  float = real;
  char_set = set of char;
  data_block_type = record
    v : float;
    vs : string[12];
    vv : boolean
  end;
  line_string = string[80];
  TitleStrPtr = TitleStr;
  WinRecPtr = WinRec;
  WinRec = record
    Next:WinRecPtr;
    State:WinState;
    Title:TitleStrPtr;
    TitleAttr,FrameAttr: Buffer: Pointer;
  end;

const
  BELL = 7;
  BS = 8;
  LF = 10;
  CR = 13;
  TAB = 9;
  ESC = 27;
  F1 = 256+50; F2 = 256+60; F3 = 256+61; F4 = 256+62; F5 = 256+63;
  F6 = 256+64; F7 = 256+65; F8 = 256+66; F9 = 256+67; F10 = 256+68;
  F11 = 256+133; F12 = 256+134;
  SF1 = 256+84; SF2 = 256+85; SF3 = 256+86; SF4 = 256+87; SF5 = 256+88;
  SF6 = 256+89; SF7 = 256+90; SF8 = 256+91; SF9 = 256+92; SF10 = 256+93;
  SF11 = 256+135; SF12 = 256+136;
  CF1 = 256+94; CF2 = 256+95; CF3 = 256+96; CF4 = 256+97; CF5 = 256+98;
  CF6 = 256+99; CF7 = 256+100; CF8 = 256+101; CF9 = 256+102; CF10 = 256+103;
  CF11 = 256+137; CF12 = 256+138;
  AF1 = 256+104; AF2 = 256+105; AF3 = 256+106; AF4 = 256+107; AF5 = 256+107;
  AF6 = 256+108; AF7 = 256+109; AF8 = 256+110; AF9 = 256+111; AF10 = 256+113;
  AF11 = 256+137; AF12 = 256+138;
  ALT1 = 256+120; ALT2 = 256+121; ALT3 = 256+122; ALT4 = 256+123; ALT5 = 256+124;
  ALT6 = 256+125; ALT7 = 256+126; ALT8 = 256+127; ALT9 = 256+128; ALT10 = 256+129;
  ALT11 = 256+130; ALT12 = 256+131; ALT13 = 256+132; ALT14 = 256+133; ALT15 = 256+134;
  ALT16 = 256+135; ALT17 = 256+136; ALT18 = 256+137; ALT19 = 256+138; ALT20 = 256+139;
  ALT21 = 256+140; ALT22 = 256+141; ALT23 = 256+142; ALT24 = 256+143; ALT25 = 256+144;
  ALT26 = 256+145; ALT27 = 256+146; ALT28 = 256+147; ALT29 = 256+148; ALT30 = 256+149;
  ALT31 = 256+150; ALT32 = 256+151; ALT33 = 256+152; ALT34 = 256+153; ALT35 = 256+154;
  ALT36 = 256+155; ALT37 = 256+156; ALT38 = 256+157; ALT39 = 256+158; ALT40 = 256+159;
  ALT41 = 256+160; ALT42 = 256+161; ALT43 = 256+162; ALT44 = 256+163; ALT45 = 256+164;
  ALT46 = 256+165;
  Home = 256+71; UpArrow = 256+72; PgUp = 256+73; 
  LeftArrow = 256+75; RightArrow = 256+77; Endkey = 256+79;
  DownArrow = 256+80; PgDn = 256+81; Insertkey = 256+82; 
  Delkey = 256+83; CPrtSc = 256+114; CLeftArrow = 256+115; 
  CRightArrow = 256+118; CEndkey = 256+117; CPgDn = 256+118; 
  Home = 256+119; CPgUp = 256+132;

  number_set : char_set = ['0'..'9','(',')','~','!','$','@','#','^','&','*','(',')','%','\',']','\','|']
  filename_set : char_set = ['A'..'Z','a'..'z','0'..'9','\','\','\','\','\','\','\','\','\','\','\','\']
  all_set : char_set = [chr(32)..chr(126)];

var
  page_number : byte;
  relative_mode : boolean;
  last_key_code : integer;
  TopWindow : WinRecPtr;
  WindowCount : integer;
  Regs : Registers;
procedure beep;
function extended_read_key : integer;
procedure get_key;
procedure inkey;
function test_key(key_code : integer) : boolean;
function spaces(number : byte) : string;
function strip_spaces(s : line_string) : string;
function date_string(year, month, day : word) : string;
function time_string(hour, minute, second : word) : string;
function htof(t : char) : byte;
function decimal(hx : string) : longint;
procedure expand_window;
procedure contract_window;
procedure wrap_up(var n : integer; limit : integer);
procedure wrap_down(var n : integer; limit : integer);
procedure set_video_mode(mode : byte);
procedure blink_enable(flag : boolean);
procedure set_page(n : byte);
procedure set_cursor(x, y : byte);
procedure set_char(x, y : byte; var character, attribute : byte);
procedure put_char(character, attribute : byte);
procedure put_string(x, y : byte; s : string);
procedure put_string2(x1, y1 : byte; s1 : string; x2, y2 : byte; s2 : string);
procedure put_number(x, y : byte; n : float; f1, f2 : byte);
procedure put_string_and_number(x1, y1 : byte; s1 : string; x2, y2 : byte; n : float; f1, f2 : byte);
procedure change_attribute(x, y, new_attribute : byte);
procedure highlight_line(xmin, xmax, y, new_attribute : byte);
procedure write_boarder_char(x, y, character_index, attribute : byte);
procedure horizontal_line(xa, xb, y : integer; width, attribute : byte);
procedure vertical_line(x, ya, yb : integer; width, attribute : byte);
procedure left_line(x, ya, yb, width, attribute : byte);
procedure right_line(x, ya, yb, width, attribute : byte);
procedure top_line(xa, xb, y, width, attribute : byte);
procedure bottom_line(xa, xb, y, width, attribute : byte);
procedure draw_box(xmin, ymin, xmax, ymax, width, attribute : byte);
procedure update(var n : datablock_type; v : float; f1, f2 : byte);
procedure edit_line(xa, xb, y, next_attribute : byte; vars : linestring; allowedset : charset);
procedure get_full_string(xa, xb, y, next_attribute : byte; vars : line_string; allowedset : char_set);
procedure edit_number(xa, xb, y, editcolour, f1, f2 : byte; var d : datablock_type);
procedure get_number(xa, xb, y, editcolour : byte; f1, f2 : byte; var d : datablock_type);
procedure full_window;
procedure clear_screen_area(xa, ya, xb, yb : byte);
procedure ActiveWindow(Active : Boolean);
procedure OpenWindow(X1, Y1, X2, Y2 : Byte; T : TitleStr; TAttr, FAttr : Byte);
procedure CloseWindow;

implementation

var
  character, attribute : byte;
  begin
  Sound(500); Delay(25); NoSound;
end;

function extended_read_key : integer;
{ Returns 0 if no Key is pressed
  n if an ASCII key is pressed
  256+n for extended keys like F1 or Alt-1 etc.
  See Pages 135 and 478 of The IBM PC and PS/2 by Peter Norton }
var
  Ch : char;
  Code : integer;
begin
  Code := 0;
  Ch := #0;
  if KeyPressed then
    begin
      Ch := ReadKey;
    end;
if Ch = #0 then
    begin
      Code := 256;
      Ch := ReadKey;
      end;
    end;

  extended_read_key := Code + ord(Ch);
end;

procedure get_key;
( assigns value to last_key_code )
begin
  Ch := ReadKey;
end;

procedure inkey;
( waits for key and returns extended code )
begin
  repeat
    last_key_code := extended_read_key;
    until last_key_code<>0;
end;

function test_key(key_code: integer): boolean;
( compares last_key_code with parameter )
begin
  get_key;
  if last_key_code=key_code then
    test_key:=true
  else
    test_key:=false;
end;

function spaces(number: byte) string;
( returns a string of spaces )
var
  i : integer;
  t : line_string;
begin
  for i := 1 to number do t[i] := ' ';
  spaces := t;
end;

function strip_spaces(s: line_string) string;
( remove leading and trailing spaces )
var
  ts, t : line_string;
begin
  ts := s;
  while copy(ts,1,1)=' ' do
    ts := copy(ts,2,length(ts)-1);
  while copy(ts,length(ts),1)=' ' do
    ts := copy(ts,1,length(ts)-1);
  strip_spaces := ts;
end;

function date_string(year,month,day: word): string;
const
  month_names: array[1..12] of string[3] = (
    'Jan','Feb','Mar','Apr','May','Jun',
    'Jul','Aug','Sep','Oct','Nov','Dec');
var
  ts,t : line_string;
begin
  str(Day:2,ts);
  ts := ts+ '<month_names[month]>';
  str(Year,t); ts := ts+copy(t,3,2);
  date_string := copy(ts+spaces(8),1,9);
end;

function time_string(hour,minute,second: word): string;
var
  ts,t : line_string;
begin
  str(Day:2,ts);
  ts := ts+ '<month_names[month]>';
  str(Year,t); ts := ts+copy(t,3,2);
  date_string := copy(ts+spaces(8),1,9);
end;
begin
  ts := '';
  str(Hour,t); if length(t)=1 then t := '0'+t; ts := ts+t+':';
  str(Minute,t); if length(t)=1 then t := '0'+t; ts := ts+t+':';
  str(Second,t); if length(t)=1 then t := '0'+t; ts := ts+t;
  time_string := ts;
end;

function htod(t : char) : byte;
{ convert a hexadecimal character to byte value 0..15 }
begin
  case t of
    '0'..'9': htd := ord(t)-ord('0');
    'A'..'F': htd := ord(t)-ord('A')+10;
    else
      begin
        write(chr(7);
        htd := 0;
      end;
  end{ case}
end;

function decimal(hx : string) : longint;
{ convert hexadecimal string into decimal }
var
  i : integer;
  temp : longint;
begin
  temp := 0;
  for i := 1 to length(hx) do
    temp := temp*16+htod(hx[i]);
  decimal := temp;
end;

procedure expand_window;
begin
  dec(windmin,$0101);
  inc(windmax,$0101);
end;

procedure contract_window;
begin
  inc(windmin,$0101);
  dec(windmax,$0101);
end;

procedure wrap_up(var n integer; limit integer);
begin
  if n<limit then
    inc(n)
  else
    n := 1;
end;

procedure wrap_down(var n integer; limit integer);
begin
  if n>limit then
    dec(n)
  else
    n := limit;
end;

procedure set_video_mode(mode byte);
begin
  with Regs do
    begin
      AH := 0; { Set mode }
      AL := mode;
      Intr(16,Regs);
      end;
end;

procedure blink_enable(flag boolean);
begin

end;

Annex 4 : Slave PC software
with Regs do
begin
    AH := 16; \{Colour palette interface\}
    AL := 3; \{Blink option\}
    if flag then
        BL := 1 \{Blink on\}
    else
        BL := 0; \{Blink off\}
    Intr(16,Regs);
end;
end;

procedure set_page(n: byte);
{ selects a page and makes it active }
begin
    with Regs do
    begin
        AH := 5; \{ Set active page number \}
        pagenumber := n;
        AL := n;
        Intr(16,Regs);
    end;
end;

procedure set_cursor(x, y: byte);
begin
    with Regs do
    begin
        AH := 2; \{ Set cursor position \}
        if relative_mode then
            begin
                DH := y+hi(WindMin)-1;
                DL := x+lo(WindMin)-1;
            end
        else
            begin
                DH := y;
                DL := x;
            end;
        BH := pagenumber;
        Intr(16,Regs);
    end;
end;

procedure get_char(x, y: byte; var character, attribute: byte);
begin
    set_cursor(x, y);
    with Regs do
    begin
        AH := 8; \{ read character and attribute \}
        BH := page_number;
        Intr(16,Regs);
        character := AL;
        attribute := AH;
    end;
end;

procedure put_char(character, attribute: byte);
begin
    with Regs do
    begin
        if attribute=0 then
            AH := 10 \{ write character \}
        else
            AH := 9; \{ write character and attribute \}
        AL := character;
        BH := page_number;
        BL := attribute;
        CX := 1;
        Intr(16,Regs);
    end;
end;

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   putstring(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 := wherery;
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 200,190,194,195,197,194,195,194,197,198),
1989 199,200,201,202,203,204,209,206,216,208),
1990 208,210,211,196,213,214,215,216,197,218),
1991 (180,180,182,185,183,184,185,182,187,188),
1992 189,181,191,197,194,197,194,197,198,199),
1993 215,200,201,202,203,204,209,206,216,208),
1994 209,210,208,198,213,214,215,216,180,194),
1995 (180,180,181,182,183,181,185,182,187,188),
1996 189,190,180,193,193,197,193,197,198,199),
1997 215,200,201,202,203,204,209,206,216,208),
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),
- 205 -
begin
  get_char(x,y,current_character,current_attribute);
  case current_character of
    179..218 :
      next_character := merge_table(character_index,current_character-178);
    else
      next_character := character_table(character_index);
  end;
  put_char(next_character,attribute);
end;

procedure horizontal_line(xa,xb,y: integer;width,attribute: byte);
var
  i : integer;
begin
  if xa<xb then
    for i := xa to xb do write_boarder_char(1,y,5+6*(width-1),attribute);
  end;
end;

procedure vertical_line(x,ya,yb: integer;width,attribute: byte);
var
  i : integer;
begin
  if ya<=yb then
    for i := ya to yb do write_boarder_char(x,1,6+6*(width-1),attribute);
  end;
end;

procedure left_line(x,ya,yb,width,attribute: byte);
var
  offset : shortint;
begin
  offset := 6*(width-1);
  write_boarder_char(x,ya,1+offset,attribute);
  write_boarder_char(x,yb,4+offset,attribute);
  vertical_line(x,ya+1,yb-1,width,attribute);
end;

procedure right_line(x,ya,yb,width,attribute: byte);
var
  offset : shortint;
begin
  offset := 6*(width-1);
  write_boarder_char(x,ya,1+offset,attribute);
  write_boarder_char(x,yb,4+offset,attribute);
  vertical_line(x,ya+1,yb-1,width,attribute);
end;

procedure horizontal_line(xa,xb,y: integer;width,attribute: byte);
var
  i : integer;
begin
  if xa<xb then
    for i := xa to xb do write_boarder_char(1,y,5+6*(width-1),attribute);
  end;
end;

procedure vertical_line(x,ya,yb: integer;width,attribute: byte);
var
  i : integer;
begin
  if ya<=yb then
    for i := ya to yb do write_boarder_char(x,1,6+6*(width-1),attribute);
  end;
end;

procedure left_line(x,ya,yb,width,attribute: byte);
var
  offset : shortint;
begin
  offset := 6*(width-1);
  write_boarder_char(x,ya,1+offset,attribute);
  write_boarder_char(x,yb,4+offset,attribute);
  vertical_line(x,ya+1,yb-1,width,attribute);
end;

procedure right_line(x,ya,yb,width,attribute: byte);
var
  offset : shortint;
begin
  offset := 6*(width-1);
write_boarder_char(x,ya,2+offset,attribute);
write_boarder_char(x,yb,3+offset,attribute);
vertical_line(x,ya+1,yb-1,width,attribute);
end;

procedure top_line(xa,xb,y,width,attribute : byte);
var
offset : shortint;
begin
offset := 6*(width-1);
write_boarder_char(xa,y,1+offset,attribute);
write_boarder_char(xb,y,2+offset,attribute);
horizontal_line(xa+1,xb-1,y,width,attribute);
end;

procedure bottom_line(xa,xb,y,width,attribute : byte);
var
offset : shortint;
begin
offset := 6*(width-1);
write_boarder_char(xa,y,4+offset,attribute);
write_boarder_char(xb,y,3+offset,attribute);
horizontal_line(xa+1,xb-1,y,width,attribute);
end;

procedure update(var n : data_black_type; v : float; f1,f2 byte);
begin
n.v := v;
str(v:f1:f2,n.vs);
n.vv := true;
end;

procedure draw_box(xmin,ymin,xmax,ymax,width,attribute : byte);
begin
 top_line(xmin,xmax,ymin,width,attribute);
 bottom_line(xmin,xmax,ymax,width,attribute);
 left_line(xmin,ymin,ymax,width,attribute);
 right_line(xmin,ymin,ymax,width,attribute);
end;

procedure edit_line(xa,xb,y,nextattribute byte; vars line_string;
 allowed_set : char_set);
var
i : integer;
ts,ta : line_string;
x,character,attributebyte;
done,first : boolean;
procedure write_string;
var
i : integer;
begin
for i := x to xb do
begin
 set_cursor(i,y);
 put_char(ord(ts[i-xa+1]),0);
end;
 set_cursor(x,y)
end;

procedure move_right;
begin
if (x<xb)then x := x+1;
end;

procedure move_left;
begin
if x>xa then x := x-1;
end;

procedure move_end;
begin
x := xb;
while(copy(ts,x-xa+1,l)='') and (x>xa) do x := x-1;
end;
procedure delete_char;
begin
  ts := copy(ts,1,x-xa)+copy(ts,x-xa+2,length(ts)-(x-xa+l)+1);''
  write_string;
end;

procedure restore_attributes;
var
  i : integer;
begin
  for i := xa to xb do
  begin
    set_cursor(i,y);
    put_char(ord(ts[i-xa+l]),ord(ta[i-xa+l]));
  end;
  set_cursor(xa,y);
end;

begin
  ts := '';
  ta := '';
  for i := xa to xb do
  begin
    get_char(i,y,character,attribute);
    ts := ts+chr(character);
    ta := ta+chr(attribute);
    put_char(character,next_attribute);
  end;
  x := xa;
done := false;
end;

repeat
  set_cursor(x,y);
inkey;
casetlase_keycodeof
  Home : x := xa; { home }
  LeftArrow : move_left; { left }
  RightArrow : move_right; { right }
  Endkey : move_end; { right edge of string }
  Delkey : delete_char; { delete }
  BS : begin { delete left }
    move_left;
    delete_char;
  end;
  127 : begin { move to far left and clear }
    x := xa;
    ts := spaces(xb-xa+1);
    write_string;
  end;
  32..255 : if chr(last_key_code) in allowed_set then
    begin { allowed character }
      if first and (ts<>spaces(xb-xa+1)) then
      begin
        insert(chr(last_key_code),ts,x-xa+1);
        ts := copy(ts,1,length(ts)-1);
        write_string;
        move_right;
      end;
      CR,ESC,256..512 : done := true;
    end;
    first := false;
  until done;
s := ts;
s := strip_spaces(ts);
put_string(xa,y,copy(s,1,xb-xa+1));
restore_attributes;
put_string(xa,y,copy(s,1,xb-xa+1));
set_cursor(xa,y);
end;

procedure get_full_string(xa,xb,y,next_attribute : byte; var s : line_string;
allowed_set : char_set);

begin
  repeat
    edit_line(xa,xb,y,next_attribute,s,allowed_set);
  until pos(' ',s)=0;
end;

procedure edit_number(xa,xb,y,edit_colour,f1,f2 : byte; var d : data_block_type);
  var
    i,code : integer;
    done : boolean;
    ts : line_string;
  begin
    done := false;
    repeat
      edit_line(xa,xb,y,edit_colour,ts,number_set);
      ts := strip_spaces(ts);
      if ts=' ' then
        done := true;
        with d do
          begin
            vv := false; v := 0; vs := ''
          end;
    until done;
    end;
end;

procedure get_number(xa,xb,y,edit_colour : byte; fl,f2 byte; var d : data_block_type);
begin
  repeat
    edit_number(xa,xb,y,edit_colour,fl,f2,d);
    until d.vv or (length(d.vs)=0);
end;

procedure full_window;
begin
  window(1,1,80,25);
end;

procedure clear_screen_area(xa,ya,xb,yb byte);
var
  wmin, wmax : word;
begin
  wmin := window(xa,ya,xb,yb);
  wmax := window(lo(wmin),hi(wmin),lo(wmax),hi(wmax));
end;

procedure ActiveWindow(Active: Boolean);
begin
  if TopWindow <> nil then
    begin
      UnFrameWin;
      with TopWindow do
        if Active then
          FrameWin(Title$, DoubleFrame, TitleAttr, FrameAttr)
    end;
end;
else
  FrameWin(Title^, SingleFrame, FrameAttr, FrameAttr);
end;
end;

procedure OpenWindow(X1, Y1, X2, Y2: Byte; T: TitleStr;
  TA: Attr, FA: Attr: Byte);
var
  W: WinRecPtr;
begin
  ActiveWindow(False);
  New(W);
  with WAdo do
    begin
      TopWindow := W;
      Inc(WindowCount);
      TextAttr := TA;
      ClrScr;
    end;

  TopWindow := W;
  Inc(WindowCount);
  TextAttr := TA;
  ClrScr;
end;

procedure CloseWindow;
var
  W: WinRecPtr;
begin
  if TopWindow <> nil then
    begin
      W := TopWindow;
      with W do
        begin
          UnFrameWin;
          WriteWin(Buffer^);
          FreeMem(Buffer, WinSize);
          FreeMem(Title, Length(Title^) + 1);
          RestoreWin(State);
          TopWindow := Next;
        end;
      Dispose(W);
      ActiveWindow(True);
      Dec(WindowCount);
    end;

  ( initialisation )
  begin
    set_page(0);
    relative_mode := true;
    last_key_code := 0;
    TopWindow := nil;
    WindowCount := 0;
  end.
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

- 212 - Annex 5 : PSpICE simulations
vtoi2

* Output Stage of Precision Current Drive based on DRIDRV1.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 SMEG
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
Annex 5: PSPICE simulations
vtoi3

* Output Stage of Precision Current Drive based on DRIDRV1.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
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

Annex 5: PSpice simulations
Date/Time run: 03/24/94 09:15:44

Temperature: 27.0

Output of E1

Output Voltage

Current in Coil

Gain in dB

100mH 1.0h 10h 100h 1.0Kh 10Kh

vdB(6) vdB(14) 20*\log_{10}(i(L2)/v(15)/70)
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
REF 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
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.

**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.

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.
SPECIFICATIONS
(typical @ +25°C, Vg = ±15 V, Vb = +5 V unless otherwise specified)

Model
AD1175K

RESOLUTION
22 Bits +10% Overrange
(4,000,000 Counts) max

DYNAMIC RANGE
133 dB

ACCURACY
Integral Nonlinearity
±0.1 ppm FSR, 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 FS, max
Differential Nonlinearity TC
±0.0025 ppm FS, max

POWER SUPPLY REJECTION RATIO (±15 V)
±5 ppm FSR/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%1
Maximum Input (Operating Only)
+9.6 V
Reference Output Voltage
+6.95 ± 2%
Output Resistance
250 Ω
Temperature Coefficient
−0.04 °C/°C (±0.08 °C/°C, max)
Drift with Time
+1 ppm/Day
1st 15 Days Operating
±25 ppm/1000 hrs., max
After 15 Days Operation
±25 ppm/1000 hrs., max
Noise, 0.00 Hz to 10 Hz (95% Confidence)
1 ppm/p-p, max
User Reference Output Gain (Refereed to Reference In)
1.000 to 1.0122
Current
±2 mA, max
Stability Temperature Coefficient
±1 µV/°C, max

THROUGHPUT RATE3
@ 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 Range4
±5 V Bipolar
Max Vp-P (at Input Hi, Without Damage)
±12 V
Max Vp-P (at Input Lo, Without Damage)
±3 V
Max Vp-P,LR (Input Lo, for Rated Performance)
±100 mV
Input Resistance (Input Hi, or Input Lo)
1000 Mohms
Input Bias Current, Input Hi or Input Lo
(>10°C to +50°C)
±10 nA, typ., ±40 nA max
Input Bandwidth4
2 MHz
Large Signal
150 kHz
CARR at dc to 60 Hz
80 dB, min

ADJUSTMENTS
Offset (Programmable)
Range
±75 mV
Resolution
1 LSB Steps
Gain-Centre (Programmable)4
Range
<4.7 V p-p >5.6 V
Resolution
0.009 Steps
Gain-Fine Range4
±0.006% FS
Gain-Balance (± Full Scale) Range4
±0.005% FS

DIGITAL LEVELS
Input
0.8 V max
Low
2.0 V min
High
2.0 V min
Output
Low (≥ 4 mA)
0.45 V max
High (≥ 100 mA)
2.4 V min

POWER REQUIREMENTS
Supply Voltages (for Rated Accuracy)
±Vg
±15 V (+0.3 V each)
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
3.9" x 2.2" x 0.53" max
Shielding
Electromagnetic, 5 Sides
Weight
170 grams

NOTE1
Integral nonlinearity is specified over the entire input range (MINIMUM, FULL-SCALE ±10% Overrange).
It is a specified using the "Full Range" definition, where the error is measured after removing the offset error and
the gain errors as plus and minus full scale.
2FSR means Full Scale Range which = 10 bits.
3Gain, input, ground referenced.
4Voltage rated line.
5Adjustment performed with use according to the manufactures procedures.
6Adjustment Time is to allow 1/30 sec for 60 Hz operation, or 1/25 sec for 50 Hz operation.
7The Nominal Analog Input Voltage Range is ±5 V, but the AD1175 may be calibrated for input voltages from
+2.7 V to +5.6 V and maximum specified accuracy over range is maintained. Therefore, input voltages of ±15 V will
accurately convert when calibrated for ±5.6 V Nominal input.
8Current section is defined as the current, the input has, at a step of 0.009% from <4.7 V to +5.6 V FS.
9User accurate 10% for non-reference output is also provided for the GAIN center (±0.006% range).
10All units are factory calibrated for ±5 V Nominal Full Scale to within ±3 mV.
11Input Bandwidth specifications are for true unipolar without clipping.

OUTLINE DIMENSIONS
Dimensions shown in inches and (mm).

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
CAUTION: This module is not an embedded assembly and

PIN DESCRIPTIONS

Appendix A : AD1175K ADC
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:

\[
\text{CHARGE IN} = \int_{0}^{T} V \, dt = \int_{0}^{T} \frac{1}{R} \, \text{V} \, dt
\]

OR...

\[
\frac{1}{V_{REF}} \int_{0}^{T} V_{REF} \, dt = \int_{0}^{T} V_{REF} \, dt
\]

OR...

\[
\int_{0}^{T} V_{REF} \, dt = V_{REF} \times T_{REF} \quad \text{(SINCE V_{REF} = CONSTANT)}
\]

OR...

\[
\int_{0}^{T} V \, dt = \text{AVG.} \left[ V_{REF} \right] = V_{REF} \times T_{REF}
\]

HENCE...

\[
\text{AVG.} \left[ V_{REF} \right] = \frac{V_{REF}}{T_{REF}} \quad \text{WHERE T_{REF} IS MEASURED AND}
\]

\[
V_{REF} \quad \text{AND} \quad T_{REF} \text{ARE CONSTANT}
\]

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 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.

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) at the converter. "ANA IN LO" should reference to GROUND (SIGNAL RTN) at the signal source, via a minimum of resistance.

2. "DIO GND" AND "*V REF AND *V 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 "*V REF AND *V GND" PIN WHICH MUST BE CONNECTED VIA HEAVY COPPER TO THE "GND" PIN. "SIGNA RTN" PIN 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 specifi-
cations for initial absolute accuracy. REF OUT is a high imped-
ance 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 exci-
tation 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 toler-
ant 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 sys-
tem'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
correlated to a simple RC circuit which will automatically gen-
erate a reset sequence upon power-up. See Figure 2 for the rec-
commended 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/50 Hz (Pin 6; Input)
Pin 6 of the module selects either 33-1/3 milliseconds or 40 mil-
liiseconds 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 nor-
mal 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 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 val-
ues (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).

BUSY (Pin 24; Output)
When a COMMAND byte is written to the microprocessor com-
patible 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 associ-
ated control lines: Chip Select (CS), READ (RD), WRITE
(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.

Write Cycle Timing Requirements

Read Cycle Timing Requirements

Read Cycle Timing Requirements

Figure 3. Interface Timing Requirements

Figure 4. Bus Control Functions

- 224 -

Appendix A: AD1175K ADC
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 x 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. 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)
  - INCr ease 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*.

- **DECGAN** (06)
  - DECRease GAIN by N x 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 (<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.

- **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.

- **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.

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.

**INCROS** (03)
- INCr ease 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*.
- DECGAN (06)
- DECRease GAIN by N x 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 (<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.
- **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.

**Figure 6. Synopsis of Commands**

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>FUNCTIONAL DESCRIPTION</th>
<th>EXECUTION TIME (APPROXIMATE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFCON</td>
<td>Initiates a Conversion Using the Power-Up Default Offset and Gain</td>
<td>50ms</td>
</tr>
<tr>
<td>MODCON</td>
<td>Initiates a Conversion Using the Modified Offset and Gain Values</td>
<td>50ms</td>
</tr>
<tr>
<td>NEWOS</td>
<td>Subtract System Offset (Last Conv. Result) from All MODCON Conversions</td>
<td>120µs</td>
</tr>
<tr>
<td>INCROS</td>
<td>Increase the Offset Used by MODCON Conversions</td>
<td>110µs</td>
</tr>
<tr>
<td>DECROS</td>
<td>Decrease the Offset Used by MODCON Conversions</td>
<td>110µs</td>
</tr>
<tr>
<td>INCGAN</td>
<td>Increase the Gain Used by MODCON Conversions</td>
<td>120µs</td>
</tr>
<tr>
<td>DECGAN</td>
<td>Decrease the Gain Used by MODCON Conversions</td>
<td>120µs</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Write Most Recent Modified Offset &amp; Gain Values to Nonvolatile Memory</td>
<td>48ms</td>
</tr>
</tbody>
</table>

- 225 - Appendix A : AD1175K ADC
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 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.

<table>
<thead>
<tr>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAR</td>
<td>M</td>
<td>U</td>
<td>P</td>
<td>E</td>
<td>R</td>
<td>O</td>
<td>L</td>
</tr>
</tbody>
</table>

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 DEGAN 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 both at the High and the Low states in order to operate properly.

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.
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) and model 732A dc secondary voltage standard (accurate to within ±1.5 ppm of the international volt†).

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.

*FLUKE is a registered trademark of John Fluke Manufacturing Company, Inc.
**IBM PC/XT/AT is a trademark of International Business Machines Corp.
†Traceable to the NATIONAL BUREAU OF STANDARDS.
INTERFACI NG TO AN 8051 MICRO CONTROLLER

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.

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.

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 P1.2, #OFFH ;SET DATA BUS TO ALL ONES

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

SETB 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

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 BASICA 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 the AD1175K 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.
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 mounted directly on a metal chassis for maximum heat-transfer capability. All-welded construction ensures maximum reliability.

### VISHAY MODELS VHP-3, VHP-4

<table>
<thead>
<tr>
<th>Res. Range</th>
<th>VHP-3*</th>
<th>VHP-4*</th>
</tr>
</thead>
<tbody>
<tr>
<td>± ±0.01%</td>
<td>± ±0.02%</td>
<td>± ±0.02%</td>
</tr>
<tr>
<td>± ±0.02%</td>
<td>± ±0.05%</td>
<td>± ±0.05%</td>
</tr>
<tr>
<td>± ±0.1%</td>
<td>± ±0.1%</td>
<td>± ±0.1%</td>
</tr>
<tr>
<td>± ±0.5%</td>
<td>± ±0.5%</td>
<td>± ±0.5%</td>
</tr>
<tr>
<td>± ±1.0%</td>
<td>± ±1.0%</td>
<td>± ±1.0%</td>
</tr>
<tr>
<td>± ±2.0%</td>
<td>± ±2.0%</td>
<td>± ±2.0%</td>
</tr>
<tr>
<td>± ±5.0%</td>
<td>± ±5.0%</td>
<td>± ±5.0%</td>
</tr>
</tbody>
</table>

**FEATURES**

- **Temperature Coefficient of Resistance — Nominal TCR:**
  - +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:** ±1.5ppm/°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:** ±2.5ppm/°C (0°C to +25°C and +25°C to +60°C)
  - **Selected TCR Tracking:** ±0.5ppm/°C

- **Load-Life Stability:**
  - 0.01% Maximum Δ R, 3 watts on heat sink at 25°C.
  - 0.05% Maximum Δ R, 10 watts on heat sink at 25°C.

- **Shelf life stability 5pmp/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:** 0.050 to 80KO

- **Current Noise:** <0.025 μV (RMS)/volt of Applied Voltage

- **Thermal EMF:**
  - 0.5 V/°C (lead effect), 4 μV/watt (power effect).

- **Non-Inductive Design: Standard**

---

**STANDARD IMPRINTING AND DIMENSIONS**

**VHP-3**

**STANDARD MARKING ARRANGEMENT**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>21.5K</td>
<td>±0.01%</td>
<td>±0.01%</td>
<td>8530</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Case not active; insulators not required; mounting hardware not supplied.
POWER RESISTOR ENVIRONMENTAL PERFORMANCE COMPARISON/TABLE III

<table>
<thead>
<tr>
<th>Test</th>
<th>Method PAR14</th>
<th>MIL-R-39009B Limits</th>
<th>VHP-3 Typical15</th>
<th>Test Data16</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEST GROUP I Conditioning</td>
<td>4.7.2</td>
<td>±0.2% + 0.05Ω</td>
<td>0.03%</td>
<td></td>
</tr>
<tr>
<td>TEST GROUP II Resistance</td>
<td>4.7.4</td>
<td>≤ 0.1Ω</td>
<td>See Graph VI</td>
<td></td>
</tr>
<tr>
<td>Characteristic</td>
<td></td>
<td>(55°C to -125°C)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature14</td>
<td>4.7.5</td>
<td>±0.5% + 0.05Ω</td>
<td>0.01%</td>
<td></td>
</tr>
<tr>
<td>(2 hours @ -175°C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low-temperature Storage</td>
<td>4.7.17</td>
<td>≤ 0.3% + 0.05Ω</td>
<td>0.005%</td>
<td></td>
</tr>
<tr>
<td>DWV (750V @ atmos. press.)</td>
<td>4.7.6</td>
<td>≤ 0.2% + 0.05Ω</td>
<td>0.003%</td>
<td></td>
</tr>
<tr>
<td>Insulation Resistance</td>
<td>4.7.7</td>
<td>≤ 10Ω</td>
<td>&gt;10Ω/W</td>
<td></td>
</tr>
<tr>
<td>Low-temperature Operation</td>
<td>4.7.8</td>
<td>≤ 0.3% + 0.05Ω</td>
<td>0.01%</td>
<td></td>
</tr>
<tr>
<td>Momentary overload</td>
<td>4.7.9</td>
<td>≤ 0.3% + 0.05Ω</td>
<td>0.003%</td>
<td></td>
</tr>
<tr>
<td>(5 sec. @ 750V max.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moisture Resistance</td>
<td>4.7.10</td>
<td>≤ 0.5% + 0.05Ω</td>
<td>0.02%</td>
<td></td>
</tr>
<tr>
<td>Terminal Strength</td>
<td>4.7.11</td>
<td>≤ 0.2% + 0.05Ω</td>
<td>0.005%</td>
<td></td>
</tr>
<tr>
<td>TEST GROUP III Shock</td>
<td>4.7.12</td>
<td>≤ 0.2% + 0.05Ω</td>
<td>0.005%</td>
<td></td>
</tr>
<tr>
<td>Medium Impact</td>
<td></td>
<td>(Post-test DWV @ 750V)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vibration - High-Frequency</td>
<td>4.7.13</td>
<td>≤ 0.2% + 0.05Ω</td>
<td>0.005%</td>
<td></td>
</tr>
<tr>
<td>(Post-test DWV @ 750V)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEST GROUP IV Life</td>
<td>4.7.14</td>
<td>≤ 1.0% + 0.05Ω</td>
<td>0.01%</td>
<td></td>
</tr>
<tr>
<td>10-1000h @ 25°C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(85% power for 2,000 hours @ 70°C)</td>
<td>—</td>
<td></td>
<td>0.04%</td>
<td></td>
</tr>
<tr>
<td>TEST GROUP V High-</td>
<td>4.7.15</td>
<td>≤ 1.0% + 0.05Ω</td>
<td>0.05%</td>
<td></td>
</tr>
<tr>
<td>Temperature Exposure</td>
<td></td>
<td>(2,000 hours @ 125°C)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.O.L. standard for all specifications except TCR. (For TCR information, see Section 3.)
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 Normal MIL-TCR apply to all resistance values including low-value resistors.
4. Vishay Standard TCR Spread is defined as a designer reference which represents that 95% of the units supplied, over a long period of time, will be at least the figure shown or better.
5. 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 Normal MIL-TCR apply to all resistance values including low-value resistors.
6. 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 Normal MIL-TCR apply to all resistance values including low-value resistors.
7. Heat sink chassis dimensions and requirements per MIL-R-39099/18: 1.000" = 25.4 mm; L: 6.00 to 152.4; W: 4.00 to 101.6; H: 2.00 to 50.8; T: 0.84 to 1.0.
8. Inductance (L) due mainly to the leads.
9. Corrosion resistance of all leads.
10. Resistance values are noted within parentheses.
11. Vishay Standard TCR Spread 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 Normal MIL-TCR apply to all resistance values including low-value resistors.
12. Vishay 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.
14. Vishay 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.
15. Vishay 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.
16. Vishay 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.

PART NUMBER SYSTEM

Please specify Vishay VHP-3 resistors as follows:

Example:

<table>
<thead>
<tr>
<th>VHP-3</th>
<th>21.5K</th>
<th>0.01%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model No.</td>
<td>Resistance Value</td>
<td>Tolerance</td>
</tr>
</tbody>
</table>

Resistance Value Designation is not encoded on this product.