"MICROCOMPUTER CONTROL OF A BLAST FURNACE STOVE MODEL"

PETER BUDIMIR, B.Sc., B.E.(Hon)

BEING A THESIS SUBMITTED
AS PARTIAL FULFILMENT FOR THE
DEGREE OF MASTER OF ENGINEERING SCIENCE
IN THE
DEPARTMENT OF ELECTRICAL ENGINEERING
THE UNIVERSITY OF ADELAIDE

APRIL 1983

This thesis embodies the results of supervised project work making up 2/3 of the work for the degree.

DECLARATION

This thesis contains no material which has been accepted for the award of any other degree or diploma in any University, and to the best of my knowledge it contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

P. BUDIMIR

ACKNOWLEDGEMENT

Many have been an encouragement and help in the work relating to this thesis. In particular I am grateful to my project supervisors, Dr. D.A. Pucknell and Dr. C.P. Jeffreson for assistance of both technical and non-technical nature.

Thanks go to Telecom Australia for the leave granted to enable me to complete the thesis writing.

There are also a number of other individuals who were of particular help or encouragement; Dr. C.W. Nettle, Mr. N.M. Martin, and Mr. R.W. Korbel who produced the DAC/ADC board and developed stage 2 of the SDK-86 software.

Finally, but most importantly, I would thank my wife, Alison, for invaluable support throughout the project.

SUMMARY

Because of the large fuel requirements of blast furnace stoves, methods of increasing their operating efficiency are of real practical concern.

This has motivated research into the feasibility of implementing a microprocessor system to control these stoves so as to achieve maximum thermal efficiency. The first phase of this study is the subject matter of this thesis. It involves the initial development of a system to be used for verifying/developing control strategies on an experimental stove model in the Department of Chemical Engineering, University of Adelaide.

The thesis describes,

(1) basic stove structure,

(2) operation and control requirements,

(3) system specifications,

(4) followed by a description of the microcomputer control system: consisting of an 'upper level' PDP-11/03 microcomputer (DEC) and a 'lower level' SDK-86 microcomputer kit (INTEL).

Hardware design, construction and testing has been completed. A pre-written 'package' has been chosen for the 'upper level' software and 'lower level' software has been developed in two stages. The first stage only involved a single feedback loop for initial hardware tests whereas stage two incorporates the multi-loop system as specified.

CONTENTS

CONTENTS

CONTENTS

CONTENTS

1. INTRODUCTION

Operation of blast furnaces for iron production requires large 'blasts' of hot air for extended periods of time. Typically, air blast flowrates of 80 to 100 kg/s are necessary, at temperatures of about 1200 ℃.

This 'blast' is produced by so called Cowper stoves. Because of the huge load requirements of the furnace, it is clear that large quantities of fuel are needed to satisfy the energy demand. It is, therefore, desirable to operate the stove systems as efficiently as possible. Even small increases in stove efficiency can result in large savings in fuel costs.

This has motivated studies into means of improving the thermal efficiency of Cowper stove operation [eg. 5, 6]. However, this problem is a complex one and until the advent of cheap microprocessing and computing elements, no cost effective means existed to implement algorithms to achieve this maximum efficiency.

This project was initiated as a joint effort between the Chemical and Electrical Engineering Departments of the University of Adelaide, to develop a control system for an experimental stove system in the Chemical Engineering Department. This computer based system was required to;

(1) provide a tool for verifying and/or developing control strategies already proposed [eg. 1, 2, 6] and enable testing of the validity of assumptions made in modelling the stove operation and configurations,

(2) possibly form a base for development of an industrial controller. Since the natural response times in the industrial installation are much longer than those of the experimental rig, any scheme that can meet the speed requirements in the laboratory will be adequate for industry.

Note that, as a research tool, it is required that the system be as flexible as possible. It is, therefore, particularly important that the hardware (processors and interface) does not place restrictions on the types of control strategies to be examined. This has led to the choice of a 16 bit low level microprocessor with a parallel interface to the 'high level function' processor.

This thesis is primarily concerned with the 'by-pass main' stove configuration but the control scheme proposed, being computer based, is flexible enough to be later used for studies of 'staggered parallel' operation.

An overview of this work can be found in the author's joint paper presented at the IE Aust Conference on Microprocessors 1979 [3].

2. BLAST FURNACE STOVE OPERATION

2.1 INTRODUCTION

There were two considerations in designing the control system;

(1) firstly, and most importantly, a useful tool was required by the Chemical Engineering Department to enable research into control mechanisms to be carried out using experimental stove models,

and (2) of lesser importance, a scheme that could be adapted to actual blast furnace stove control was desirable.

To understand both the nature and complexity of the required system it is necessary to examine stove structure and operation in some detail. This chapter describes both the industrial and experimental stoves and gives a basic mathematical description of the control problem.

2.2 BLAST FURNACE STOVES

2.2.1 Introduction

The first regenerative blast furnace stove was developed in 1860 by E. A. Cowper of Scotland. Although much has been done since to improve the effectiveness of these Cowper stoves (as they are commonly known), the essential operating scheme and stove structure have remained unchanged since they were first introduced.

Stove description and operation is well covered in the literature [eg. 4, 5].

2.2.2 Stove Description

Today's Cowper stoves typically produce air blast flow rates of 80 to 100 kg/s at temperatures of about 1200 °C. A cross-sectional diagram of a stove is shown in figure 2.1. From this the stove can be seen to consist essentially of:

(1) a chequerwork - This is a large mass of solid material containing many flues through which air can be forced. It performs the heat storage function of the stove and is usually a type of refractory brick. Typically the chequerwork is about 20 m to 30 m high and about 8 m in diameter, weighing of the order of 1 million kg. The efficiency and thermal capacity of the stove are governed by the surface area of the brick chequers and their mass respectively.

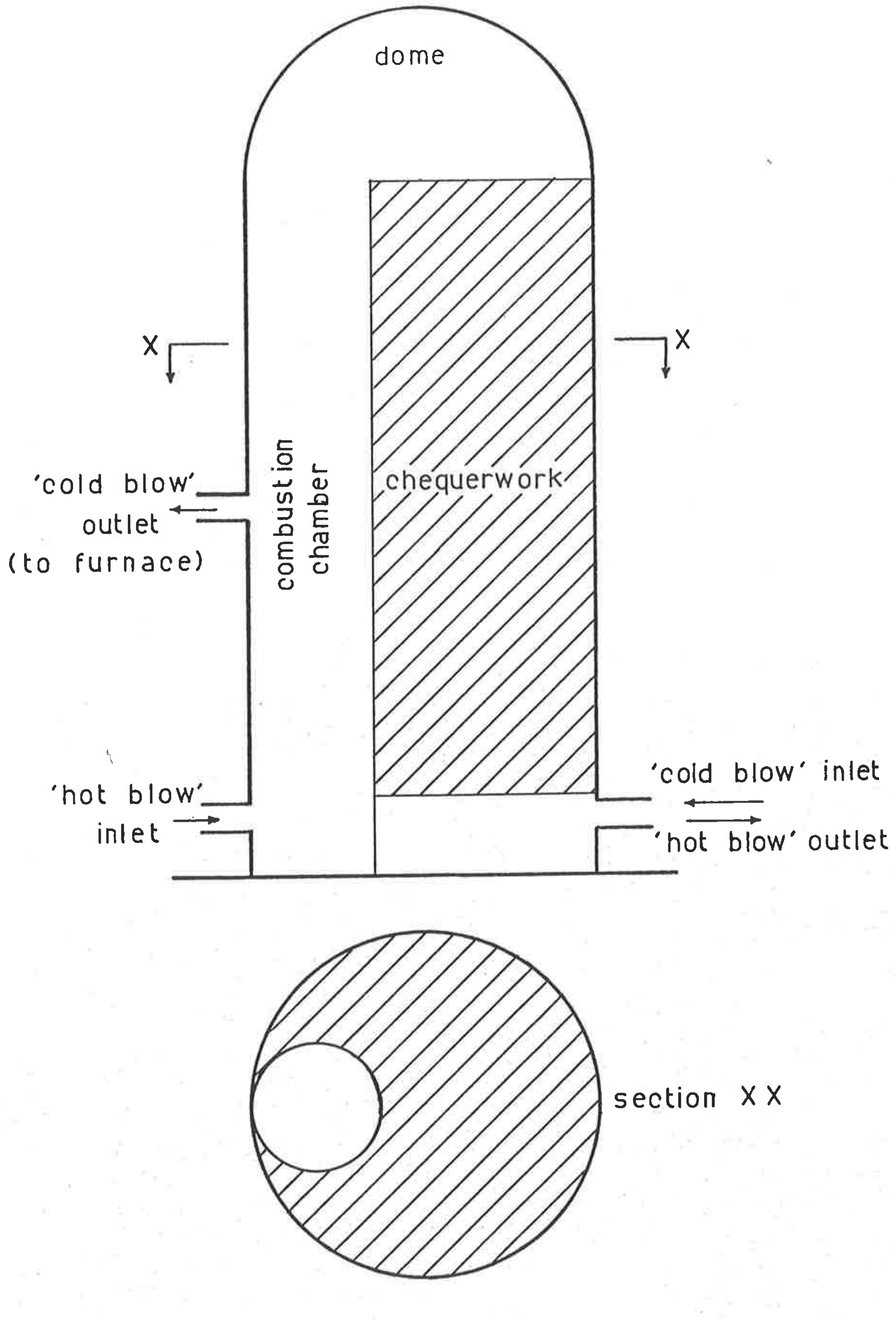
(2) a combustion chamber - Blast furnace gas (fuel) is burnt with air at the base of this chamber. The hot gas can then be forced up the chamber and down through chequerwork.

2.2.3 Stove Operation

The operation of a Cowper stove involves two distinct phases:

(1) the heating period - During this phase hot gas is forced up through the combustion chamber and down through the chequerwork. Heat is thus transferred from the gas to the chequerwork. This process is often referred to as the 'on gas' phase of the cycle or, in academic circles, the 'hot blow' period.

(2) the cooling period - This is the phase in which the



Stove Cross Section

FIGURE 2.1

stove actually supplies the hot air blast to the furnace (i.e. the 'on blast' part of the cycle). Cold air is forced up through the chequerwork and is thus heated and exits the stove as a hot blast. This is often referred to as the 'cold blow' part of the cycle.

2.2.4 Stove Configuration

It is clear from the above description that one stove is not sufficient if a continuous hot blast is required, since the chequerwork will cool during the 'cold blow' cycle. Eventually the outlet blast temperature will no longer be sufficient to supply the blast furnace requirement.

Thus two or more stoves are necessary. While one (or more) is 'on blast' the others are in their heating phases. By 'switching' the stoves at appropriate times the required continuous blast is maintained.

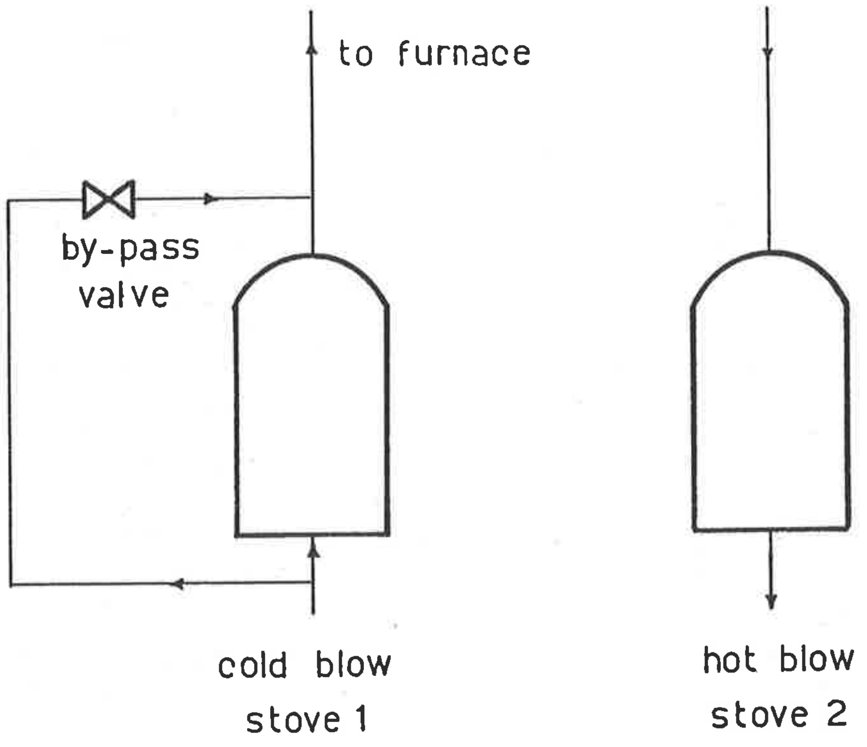
Since the outlet temperature of the 'cold blow' stove will drop in time, some mechanism must exist to provide a fixed blast temperature at a given flow rate. At present two main stove configurations exist to meet this requirement:

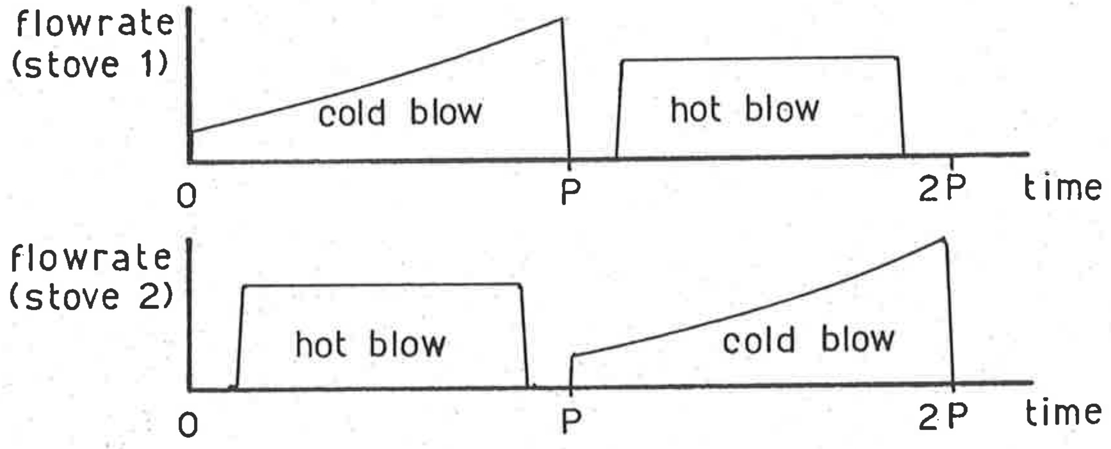
(1) the 'staggered parallel' system - This is the more efficient [5] but typically requires four stoves (a minimum of three). Operation is described in [5] and basically involves two stoves 'on blast' at any one time. These two stoves are 'staggered' in that one is further into its cycle than the other. Thus the outlet temperatures will be different. By controlling the percentage of air passing

through each of the two stoves the outlet temperature can be held at the required value. Eventually the 'coldest' of the 'on blast' stoves is switched with the 'hottest' of the 'hot blow' stoves. This process continues cyclically.

(2) the 'by-pass main' system - This is the older of the two configurations and for economic reasons is still preferred by the majority of users, in Australia at least. Typically three stoves are used, with two being a minimum. Also being the simpler system, initial control studies will deal with this configuration. Hence it is described in more detail here than the 'staggered parallel' system (unless otherwise stated, further references to stove configuration will assume a 'by-pass main' system).

The 'by-pass main' system is well described in [6] and this description is outlined here. Consider first the two stove system of figure 2.2. From the diagram it is seen that one stove is 'on blast' at any one time. The required blast flow rate is determined by the flow rate of the cool air input. The temperature is controlled by allowing some of the air to by-pass the stove and then mix with the heated air at the stove outlet. At the start of the 'cold blow' cycle, stove outlet temperature and hence the amount air by-passing the stove will be at a maximum. As the chequerwork cools, the exit temperature will drop and a greater portion of the air will pass through the stove. The limit of this cycle occurs when the exit temperature equals the required blast temperature. At this stage all the air





By-Pass Main Stove Configuration for 2 Stoves

FIGURE 2.2

flow would pass through the stove. In practice stove switching must occur well before this stage is reached.

During this process the 'hot blow' stove is being heated in preparation for an 'on blast' cycle. Because of finite switching times this stove must be taken off its current cycle in preparation for 'cold blow' before the 'on blast' stove ends its cycle (see figure 2.2, 2.3).

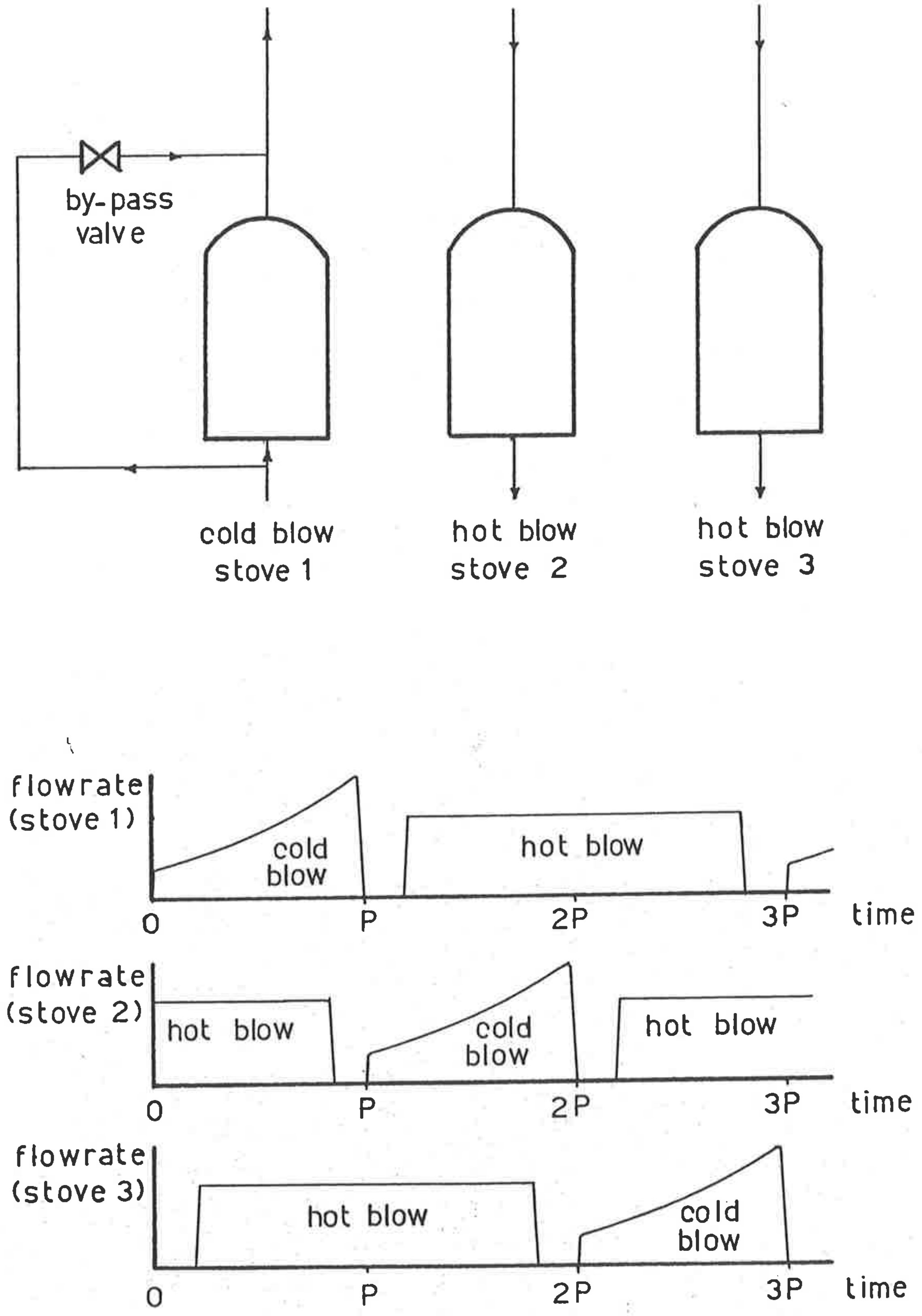
The process is essentially the same for a three stove 'by-pass main' system as can be seen from figure 2.3. Although further details of the 'by-pass main' configuration will be discussed later, one concept that bears mentioning at this stage is that of 'cyclic equilibrium'. This refers to a dynamic equilibrium of stove operation. Once reached, any one stove will have the same temperature profile (of the chequerwork) at equivalent times in each complete cycle. Note that the 'state' of the stove, at any time, is defined by this temperature profile, providing the thermal capacitance of the gas is zero.

One key factor in the control problem is determining under which conditions 'cyclic equilibrium' can be established.

2.3 EXPERIMENTAL STOVE MODEL

2.3.1 Introduction

Because of the massive size of the actual Cowper stoves it is not possible to have these available to test control procedures and study stove operation. Although computer



By-Pass Main Stove Configuration for 3 Stoves

FIGURE 2.3

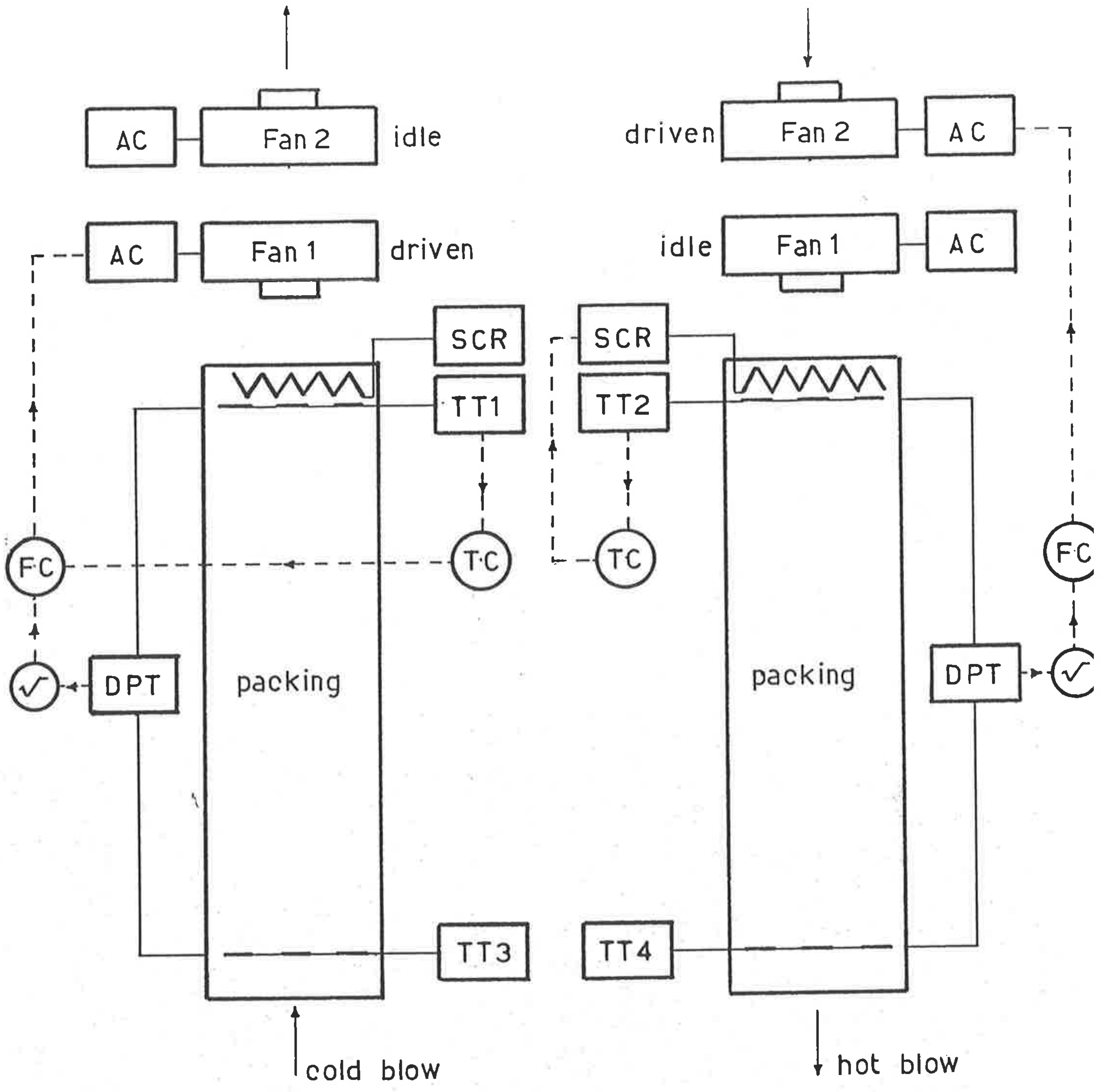
simulations have yielded some useful results [5, 6] a laboratory test bed is necessary for further investigation. The Chemical Engineering Department has developed such test columns.

Using these, control strategies can be developed and tested. Because the time constants relating to the models are much shorter than those of actual Cowper stoves, any digital control system which can handle the sampling rates required for the model will certainly have adequate capacity for controlling an industrial system.

2.3.2 Description

Figure 2.4 shows, diagramatically, the experimental stove system, set up for 'by-pass main' operation.

The left hand stove is heating cold air drawn through the packing by fan 1. As has been seen before, the exit air temperature (measured by the temperature transmitter TT1) is falling with time. Thus to obtain a constant 'blast' temperature and flow rate the air leaving stove 1 needs to be 'mixed' with a cooler 'by-pass' stream (as described in sec 2.2.4). This mixing procedure is not carried out in the experimental set up, but the 'by-pass' effect on the stove can be reproduced using a local flow control loop. This is to be achieved using a differential pressure transmitter, software square root extraction, a Zener Electrics armature current controller and variable speed DC motor. The set point of this loop can be adjusted by the temperature



KEY

TT :	Resistance Bulb Temperature Transmitter
DPT :	Differential Pressure Transmitter
TC :	Temperature Control Algorithm
FC :	Flow Control Algorithm
AC :	Armature Current Speed Controller
– → – – :	Control Flow

Experimental Stove System

FIGURE 2.4

control algorithm.

Experiments performed in the Chemical Engineering Department indicate sample rates of the order of 50 Hz will be required if significant degradation in the performance of the flow loops is to be avoided.

The second stove of figure 2.4 is on 'hot blow' using the second blower to force air over a nichrome wire heater with a phase-angle controlled SCR regulator. Experiments on a similar loop indicate that a sample rate of 10 Hz should be satisfactory for the associated temperature control loop.

Flow through the 'hot blow' stove will be initially constant, using a similar fast 'local' flow loop to the 'cold blow' stove.

This system differs from the industrial installation, mainly in the use of electric heating instead of a combustion system. The design is also intended to eliminate parasitic thermal capacitances (which are present in reality) to simplify the control studies. This has largely been achieved by avoiding the use of multiport control valves and by using light, stainless steel, vacuum-jacketted stoves. Fast temperature control loops should also ensure sharp stepwise temperature changes.

2.4 MATHEMATICAL DESCRIPTION

2.4.1 Introduction

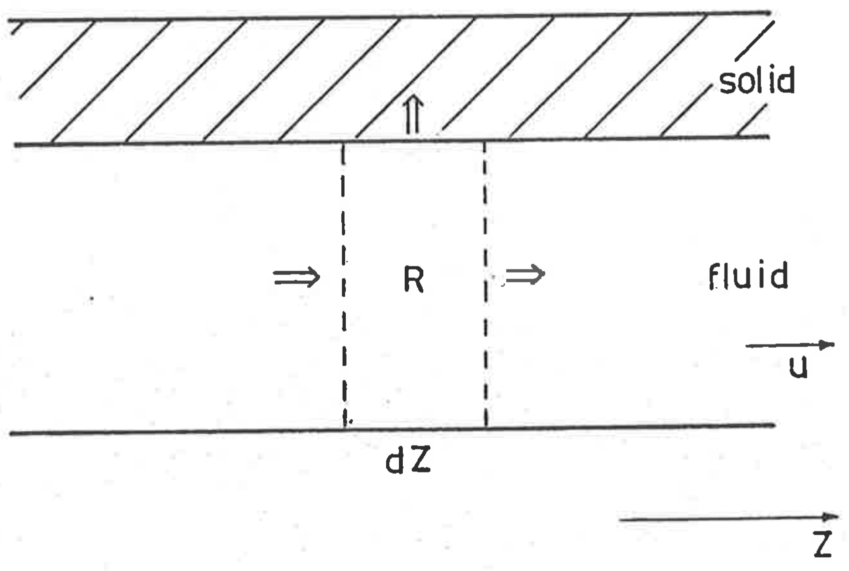
A comprehensive treatment of the principles in heat transfer has been developed by Jakob. Details of the operation and analysis of regenerators (Cowper stoves fall into this category) can be found in his book on heat transfer [7]. A brief derivation of the equations relevant to stove operation is given below. The nomenclature used has been chosen to be consistent with that adopted by Jeffreson [6].

2.4.2 Analysis

To simplify stove analysis, a one dimensional model (in space) is adopted. This results in distance (z) and time (θ) being the only independent variables in the differential equations.

Figure 2.5 is drawn to reflect this model, and relating to this the following symbols are defined.

u	..velocity of the fluid (m/s).
ρ	..density (kg/m^3).
M	..total mass of the chequerwork (kg).
C	..specific heat/unit mass for solid ($J/{}^\circ C$-kg).
S	..specific heat/unit mass for fluid (J/C°-kg).
k	..thermal conductivity (J/m-s-C°).
h	..heat transfer coefficient between solid and fluid (J/s-m^2-C°).
a	..cross-sectional area (m^2).
L	..length of chequerwork (or packing) (m).
l	..total contact length (cross-section) (m).
A	..total perimeter of flues (solid surface area, m^2).
Z	..distance independant variable (m).
θ	..time, independant variable (s).



Heat Transfer Diagram
FIGURE 2.5

T ..temperature of the solid, variable ($^\circ$C).

t ..temperature of the fluid, variable ($^\circ$C).

f,s.subscripts refer to fluid and solid respectively.

The functions $T = T(\theta,Z)$ and $t = t(\theta,Z)$ are to be determined, subject to forcing functions and the initial solid temperature distribution. Referring to figure 2.5, and applying the principle of heat balance, the net heat flow into region R is equal to the heat accumulation in R plus the heat transferred to the solid. Thus the heat balance equation for the fluid can be written,

$$a_f.dZ.k_f.\partial^2 t/\partial Z^2 = a_f.\rho_f.S.dZ.dt/d\theta + h.l.(t-T).dZ. \qquad ...(2.1)$$

Similarly for the solid,

$$a_s.dZ.k_s.\partial^2 T/\partial Z^2 = a_s.\rho_s.C.dZ.dT/d\theta + h.l.(T-t).dZ. \qquad ...(2.2)$$

Now we define,

$z = Z/L$ .... normalised length ...(2.3)

and $w = \rho_f.a_f.u$ .... fluid flow rate (kg/s) ...(2.4)

and using $d/d\theta = \partial/\partial\theta + u.\partial/\partial Z$ for the fluid, and $d/d\theta = \partial/\partial\theta$ for the solid (since the solid is stationary), equations (2.1) and (2.2) become,

$$\partial t/\partial z = h.A.(T-t)/(w.S) + (k_f.a_f/(w.L.S)).\partial^2 t/\partial z^2 - (L/u).\partial t/\partial\theta \qquad ...(2.5)$$

and $M.C.\partial T/\partial\theta = h.A.(t-T) + k_s.a_s.L.\partial^2 T/\partial z^2.$ ...(2.6)

Since the thermal capacity of the fluid is insignificant, the $\partial t/\partial\theta$ term in equation (2.5) can be removed. Also, taking typical values of stove paramaters (eg. [5] pp 35, 36) the 2nd order terms become insignificant, giving,

$$\partial t/\partial z = h.A.(T-t)/(w.S) \qquad ...(2.7)$$

and $M.C.\partial T/\partial\theta = h.A.(t-T)$. ...(2.8)

By solving these equations, subject to the relevant boundary conditions, the temperature functions can be determined. The paramaters h, A, M, C and S are characteristics of the stove material and gas. Hence they are not directly controllable by the operator. The inlet gas temperature and flow rate w are the variables that can be altered by the operator to influence stove operation.

2.4.3 By-pass Main Operation

Consider now a two stove 'by-pass main' system. Equations (2.7) and (2.8) must be applied separately to the 'hot blow' and 'cold blow' stoves. Using the subscripts 1 and 2 to denote 'cold blow' and 'hot blow' respectively, we have,

for 'hot blow'

$$\partial t_2/\partial z = h_2.A.(T-t_2)/(w_2.S_2) \qquad ...(2.9)$$

and $M.C.\partial T/\partial\theta = h_2.A.(t_2-T)$, ...(2.10)

for 'cold blow'

$$\partial t_1/\partial z = -h_1.A.(T-t_1)/(w_1.S_1) \qquad ...(2.11)$$

and $M.C.\partial T/\partial\theta = h_1.A.(t_1-T)$. ...(2.12)

The minus sign in equation (2.11) is necessary since flow is in the reverse direction for the 'cold blow'.

In addition to these equations, the 'by-pass' control during 'cold blow' results in a flow rate variation as follows (taking $t_{1in} = 0$, as the reference temperature),

$$w_1 = \hat{w}.t_B/t_{1x} \qquad \text{...(2.13)}$$

where,

$\hat{w}$ is the required blast furnace flow rate,

t_{1x} is the stove exit air temperature,

t_B is the required blast temperature.

This equation assumes that the specific heats of air at t_{1x} and t_B respectively are equal, and results from a heat balance over the mixing point.

This leaves three operator adjustable variables; the 'hot blow' inlet temperature (t_{2in}), flow rate (w_2) and the period P of the operation cycle. Having selected these, equations (2.9) and (2.10) can be solved to give a temperature profile (T) at the end of the 'hot blow'. This then becomes the initial profile in equations (2.11) and (2.12). These can be solved to give the temperature profile at the end of the 'cold blow', thus providing the initial profile for solving the 'hot blow' equations again. By repeating this procedure a 'cyclic equilibrium' is reached where the temperature profile is the same (for a given stove) at the beginning of any given hot or cold blow.

'Cyclic equilibrium' is the normal operating state of stove system and it is important that the operator selects w_2, t_{2in} and P so that equilibrium is possible. If, for example, t_{2in} is too small, the 'hot blow' stove will not store as much heat as is required. When switched to 'cold blow' it will not be able to meet the blast furnace temperature requirement for the full period P and so the stoves must be 'switched' earlier than desired. Because of this the other stove has had less time on 'hot blow' and thus aquires even less heat than stove 1. Thus the 'switching' period P must be decreased further. This self destructive mechanism will eventually lead to a failure referred to as 'collapse' (reference [8]).

2.4.4 Thermal Efficiency

It has been seen that the operator has three controllable variables (t_{2in}, w_2, and P) and providing these are chosen carefully a 'cyclic equilibrium' situation can be reached. It is clearly desirable, however, to choose these in such a way as to maximise the thermal efficiency of the stove system, while satisfying the 'demand' for hot blast air. In fact it would be preferable to be able to develop and implement control algorithms which would automatically achieve this result.

What effect do these variables have on thermal efficiency? Jeffreson [6] shows that the most efficient operation occurs by allowing P to 'float' (i.e. the stoves are switched only when the 'cold blow' stove can no longer meet the blast furnace requirements) and selecting $w_2 . t_{2in}$ as small as possible, consistent with 'cyclic equilibrium'.

In the case of zero changeover time this condition is equivalent to minimising the switching period (P).

2.4.5 Conclusion

Solving the stove equations is not possible analytically and so numerical methods are needed. Thus some digital computing elements will be necessary to predict and control the above variables to achieve maximum thermal efficiency.

3. BLAST FURNACE STOVE CONTROL

3.1 INTRODUCTION

Having considered the basic structure and operation of stove systems, the control problem can be now be examined in more detail. Because of the non-linear characteristics of stove system operation, and the nature of the heat transfer equations, any efficiency controls will necessarily involve numerical analysis.

This immediately establishes the need for some form of 'intelligent' digital control system. With the increasing availability and decreasing prices of a wide range of processors (mini and micro) and peripheral equipment, the digital control concept becomes an extremely attractive one.

3.2 CONTROL REQUIREMENTS

3.2.1 Introduction

Operation of a 'by-pass main' stove configuration involves a number of 'standard' feedback loops. In the case of the experimental stove system these loops can be seen in figure 2.4. They comprise a 'temperature' and a 'flow' feedback loop for both the 'cold blow' and 'hot blow' stoves.

Except for the 'square root' extraction in the flow loops, conventional PID (Proportional Integral Derivative) control is adequate to obtain the desired 'by-pass main' operation.

As has been described in the previous chapter, there remain three variables available for operator adjustment; the switching period (P), the 'hot blow' input gas temperature and flowrate (t_{2in}, w_2). It is our concern here to examine in more detail the selection of these variables so as to obtain the maximum thermal efficiency; this can be defined as the ratio of the total heat removed during 'cold blow' to that supplied during 'hot blow'.

3.2.2 Switching Period

Two distinct approaches exist in determining P. The first involves the selection of some predetermined value. The second, and more efficient, approach is to allow the period to 'float'. Switching only occurs when the 'cold blow' stove can no longer meet the blast furnace requirements.

In practice, of course, switching must occur before this limit point is reached. A convenient means to cater for this safety margin is to adopt the ratio described in reference [6],

$$K = w_1(P)/\hat{w}, \qquad \ldots(3.1)$$

that is, the fraction of air passing through the 'cold blow' stove at the end of its cycle. The limiting value is clearly one (no safety margin). Thus for a given K value the switching period P is defined; switching is initiated when the 'cold blow' flowrate w_1 reaches $K.\hat{w}$.

3.2.3 Hot Blow Flowrate and Inlet Gas Temperature

For a two stove system the thermal efficiency during 'cyclic equilibrium' can be written [6],

$$\Omega = \hat{w}.t_B/(w_{2avg}.t_{2in}) \qquad ...(3.2)$$

where w_{2avg} is the flowrate of the hot gas averaged over the whole cycle. This equation also assumes that all relevant specific heats are equal.

It is clear from this that the product $w_{2avg}.t_{2in}$ must be minimised. Additional to this, a very useful result has been derived by Kwakernaak in [9]. Here it is shown that thermal efficiency during the 'hot blow' is optimised if,

(1) the inlet temperature (t_{2in}) is set to its maximum value (this is a physical limitation) and,

(2) the flowrate (w_2) is held constant during this phase.

At present Jeffreson does not believe condition (2) to be important when the heat transfer coefficient is approximately proportional to the flowrate [13]. In any case, by developing a flexible control system, this and other considerations can be evaluated with the stove model.

In the context of overall operation (heating and cooling) it is not rigorously proved that thermal efficiency is maximum under these conditions. However, Kwakernaak feels from physical considerations that the above criteria should apply.

Thus, summarising these results, the conditions for maximum thermal efficiency can be stated,

(1) set the inlet temperature during 'hot blow' to the

maximum value, consistent with imposed physical limitations and,

(2) set the 'hot blow' flow rate to its minimum constant value so that 'cyclic equilibrium' can still be maintained.

The objective, therefore, of maximum efficiency control is to determine (beforehand) this minimum value of w_2, for a given blast furnace loading. Note that finite changeover time means that the actual manipulated hot gas flowrate w_2 will be greater than the average value w_{2avg} which defines the overall thermal efficiency.

3.2.4 Zero Changeover Time

For the situation where stove changeover takes zero time, the minimum w_2 can readily be determined as shown in reference [6]. Although the assumption of zero changeover is clearly not valid it is useful in yielding a lower limit value for w_2.

The approach taken to determine this value is based on the observation that the period (P) approaches zero as w_2 is decreased. Thus the minimum (most efficient) w_2 occurs in the limit as P approaches zero. Applying this criterion to equations (2.7) and (2.8) a solution becomes possible. For the case of a two stove 'by-pass main' system this takes the form (for $t_{1in} = 0$, as the reference temperature) [6],

$$t_B = t_{2in} .(1-e^{\beta})/(1-x.e^{\beta}) \qquad ...(3.3)$$

where,

$x = \hat{w}.S_1/(\bar{w}_2.S_2)$ ...(3.4)

$\beta = (x-1).\bar{\Lambda}_1/(1+\bar{h}_1/\bar{h}_2)$ ...(3.5)

and Λ_1, the 'reduced length' is defined as
$\bar{\Lambda}_1 = \bar{h}_1.A/(\hat{w}.S_1)$. ...(3.6)

Here $\bar{h}_1$ and $\bar{h}_2$ are reference values of the heat transfer coefficient during 'cold blow' and 'hot blow' cycles respectively.

The above equation can then be solved for w_{2min} once the values $\hat{w}$ and t_B are specified.

3.2.5 Non-Zero Changeover Time

Under realistic conditions, of non-zero changeover time, the problem of determining w_{2min} is considerably more difficult. A number of approaches have been investigated (to some degree) but all have their difficulties.

One approach [6] is to assume that the heat transfer coefficient is proportional to flowrate. Under such conditions, the effect of the 'hot blow' is determined by the area under the 'w_2 vs time' graph. Thus if the period is to be halved, the flowrate w_2 need only be doubled to maintain equilibrium. Hence, we can write for a 1-N stove system,

$w_2 = w_{20}.N.P/(N.P-P_c)$ ...(3.7)

where,

w_2 is the actual 'hot blow' flowrate,

w_{20} is the flowrate for $P_c = 0$,

and P is the changeover time.

However, not knowing the value of P beforehand means that w_{2min} cannot be determined from equation (3.7) alone. To overcome this, Jeffreson [6] has used an iterative approach in his stove simulations. This involves adjusting the value of w_2 in each new cycle as follows,

$$w_2^{(K+1)} = w_2^{(K)} . \bar{w}_{20} / w_{2avg}^{(K)} \qquad ...(3.8)$$

where,

$w_{2avg}^{(K)}$ is the integrated flow of the kth cycle,

and $\bar{w}_{20} = w_{20min} .N.P/(N.P-P_c)$. ...(3.9)

Note that $\bar{w}_{20}$ is just the minimum zero changeover value from equation (3.3), adjusted for the new period.

Such an approach, however may exhibit convergence problems. A possible refinement, not yet tried, may be to determine a close starting value for w_2 before applying equation (3.8). Consider first w_{20} as a linear function of P,

$$w_{20} = w_{20min} .(1+c.P) \qquad ...(3.10)$$

This is a good approximation over the normal operating range. The value w_{20min} is that determined in section 3.2.4 (the zero changeover case). The constant value 'c' could possibly be determined by simulation.

Further, by taking typical values of P and c (as can be derived from simulation results, eg. reference [6]) it is found that,

$$P \ll 1/c \qquad \ldots(3.11)$$

From equations (3.7) and (3.10) the flowrate can be expressed as,

$$w_2 = w_{20min} .(1+c.P).N.P/(N.P-P_c) \qquad \ldots(3.12)$$

Using the inequality (3.11) this can be minimized with respect to P to give,

$$w_{2min} = N.P.w_{20}/P_c \qquad \ldots(3.13)$$

where w_{20} is the flowrate defined in equation (3.10), and

$$P = \sqrt{P_c/(N.c)} + P_c/N \qquad \ldots(3.14)$$

The value of w_{2min} from equation (3.13) can then be used as the starting value in equation (3.8).

Another approach is to define thermal efficiency as the heat stored as a fraction of total heat input during any hot blow. Such a definition enables efficiency to be written as a function of the 'hot blow' exit temperature. Thus it becomes feasible that the value of w could be determined by appropriate feedback of this temperature (reference [12]).

3.3 CONCLUSION

Because of the complexity of the control problem, suitable schemes (algorithms) are still under investigation and development. Thus the control system needs to be flexible enough to incorporate the changing control algorithms, and in fact, is to be used in the development and verification of these algorithms.

It is clear that a computer based system is the only means whereby such flexibility can be introduced, as well as providing the means to cope with the problem complexity.

To meet the requirements of speed and flexibility a two level system was designed (described in chapter 5). The 'upper' level microcomputer is to handle higher level functions (such as determining w_{2min}) and the required loop control. The 'lower' level microcomputer handles basic I/O control and operator interaction. In fact, during manual control mode, the 'lower' level processor becomes a stand alone system (independent of the'upper' level processor) by which the operator can manually vary the controlled outputs.

4. SYSTEM REQUIREMENTS

4.1 INTRODUCTION

The following chapter defines the specific requirements of the control system as requested by the Chemical Engineering Department. Key decisions relating to system implementation are included together with their justifications.

The section is summarized with a brief description of the overall system structure chosen to meet the above requirements.

4.2 REQUIREMENTS

4.2.1 Overall Objectives

The experimental work in blast furnace stove modelling in the Chemical Engineering Department required the following:

(1) Equipment to yield clear and unambiguous experimental verification of theoretical mathematical models of thermal regenerator system dynamics, particularly under the variable flow conditions which prevail in industrial installations. This part of the work requires only one experimental stove.

(2) Once the experimental difficulties associated with one stove had been isolated and overcome, a further two or three stoves would be added. At this stage, the focus of the work would transfer from identification and modelling of system dynamics to the longer term objective of testing and extending control strategies for the optimal operation of

stove systems under conditions of variable heat demand.

4.2.2 Computational Requirements (Original Concepts)

As can be seen from chapters 2 and 3, the various aspects of stove operation combine to present a complex control problem. This, together with flexibility requirements, suggested some form of real time digital control. The original concept (1979) included a multi-processor system based on the Intel 8080 (as development facilities were available for this series of processor). One processor would be assigned to each stove, with communication proceeding via a common bus and memory area.

This arrangement had the attraction of providing adequate computing power by sharing computation, and also introduces a means of including a degree of fault tolerance (necessary in an industrial system). Each processor could be made capable of taking over the basic functions of another 'failed' processor.

4.2.3 Computational Requirements (Later Developments)

In the later half of 1979 the Chemical Engineering Department secured an LSI-11/03 (DEC) computer system together with DDACS (a real time operating system tailored to control applications). Calculation of the expected loop rates and estimation of the desired number of loops to be controlled indicated that the LSI-11/03 processor running DDACS would be sufficient for control of the initial stove system.

This processor and DDACS software was presented to the author virtually as an 'engineering' constraint, in that now it was necessary to tailor the system around these items. A microprocessor interface could now be used for the following purposes:

(1) Provision of the required number of A/D and D/A channels, allowing for expansion necessary for multiple stoves.

(2) Provision of bumpless auto/manual and manual/auto transfers with 'loop select' facilities.

Also, changes to DDACS software were to be avoided, since it was originally available only in 'executable image' form.

Although an 8 bit microprocessor could handle standard A/D, D/A and other I/O (input/output) control it was also desirable to be able to perform scaling and perhaps other pre-processing of data. Considering also expansion to multiple stoves, a single 8 bit processor was thought to be inadequate.

At this time INTEL released their 8086, 16 bit microprocessor. This is four to ten times more powerful than the 8080 (throughput varies according to application). In addition it provided hardware multiply/divide facilities and so seemed ideal to handle the low level I/O tasks. Its, more than adequate, processing power meant that the flexibility existed for assigning more complex tasks to this level, as required.

Thus a system configuration was chosen consisting of the LSI-11/03 microcomputer as an upper level controller, responsible for high level control and optimising tasks, with the SDK-86 (an 8086 based development kit from INTEL) as a low level I/O processor responsible for A/D and D/A control together with appropriate scaling and 'loop select' and auto/manual control.

4.3 PROCESS DESIGN

4.3.1 Introduction

This section briefly describes the design of the experimental stove system insofar as it effects the design of the computer system.

4.3.2 Thermal Design

The overall "Process Instrumentation Flow" (PI) diagram has been shown in figure 2.4. In essence the packing is first heated by a stream of hot air (shown flowing down through the stove on the right of the diagram) and then cooled by a flow of cold air which "extracts" the heat from the previous "hot blow".

On an industrial scale, the flow reversals are applied by means of a system of three-way valves on the inlet and outlet. Experience in the Chemical Engineering Department on measurements of "single blow", unidirectional, packed bed dynamics [PhD Thesis, C.P. Jeffreson] showed that, for small scale equipment, the thermal capacity of three-way valves and even fine wire heating elements prevents the application of the sharp, square-wave temperature "waves" assumed by the theoretical model. The slow, long time constant, release or absorption of heat, following the initial step is also a problem. This process (called "tailing") can be overcome, to a large extent, by incorporating a temperature control loop around the heater and three-way valves, thus eliminating long term temperature drift. Nevertheless, oscillation and overshoot become significant on the time scale of the thermal time constant of the packing, unless the thermal capacitances of the elements inside the inlet temperature control loop are reduced to a minimum. Furthermore, mechanical problems associated with sealing under high stress conditions would be expected with such solenoid or air-actuated three-way valves.

These considerations and others led to the design of figure 2.4 with two variable speed blowers per stove and an inlet temperature control loop for the hot blow part of the cycle which is closed around a fine wire heater. Because of the speed of response required for this inlet temperature control loop, a sampling interval of about 320 ms was chosen. Degradation in performance occured when the sampling interval was increased significantly above this value.

At first it was thought that close control over room temperature and the absence of large thermal capacitances on the inlet during the "cold blow", would eliminate the need for feedback control over inlet temperature during this part of the cycle. It has been found in practise that for the temperature rise chosen at the heater power available (2 kW) long term variations in room temperature do cause significant drift. The most recent design (1982) adds a further inlet temperature controller to control the cold blow inlet temperature.

This change serves to illustrate the need for sufficient flexibility and capacity in the control system if it is to be useful as a research tool. This approach differs somewhat when designing for a fixed application (eg. an industrial system) where the control system requirements can be specified more exactly.

4.3.3 Flow Control

Since the system involves variable flow control of the cooling air according to the optimal strategies to be devised, variable flow is best achieved by closing the loop around each fan. The differential pressure across the packing becomes the measured variable and the armature current, the manipulated variable. This loop is also "fast" by process control standards; a sampling interval of 80ms has been found necessary to avoid undesirable oscillation and overshoot.

4.4 OPERATOR/MACHINE INTERFACE

4.4.1 General Requirements

The considerations in the specifications for the operator interface are as follows:

(1) Because of the time taken for each regenerator to reach equilibrium, the system must be capable of unattended operation for long periods of time.

(2) It must be possible to start the system with any desired combination of loops on "manual". In this case, it should be possible to independently raise or lower outputs to the final control element of any "manual" loop. Note that the term "loop" in this context is used to refer to any control path (with or without feedback).

(3) Because of requirement (2) above, automatic, bumpless transfer from manual to automatic operation, and back again is essential.

(4) Because of the flexibility required of the system in configuring various combinations of feedforward, feedback, cascade and sequencing control, some method is required to associate any given D/A output and/or A/D input channel(s) with any specified control loop or control strategy.

4.4.2 Processor to Processor Interface

Given the processor arrangement as discussed in 4.2.3 it is necessary to provide an interface between the LSI-11/03 and the SDK-86. Clearly the simplest means of doing this would be to use a serial communication's link (such as RS232).

This was not acceptable for two main reasons:

(1) Without abandoning the simplicities inherent in a serial interface such as RS232 the data transfer rate is limited to about 9600 baud. With synchronous operation this is equivalent to 1200 bytes/sec. For a four stove system this would be currently acceptable. However, the system flexibility becomes severely limited, since higher sampling rates and more complex control strategies may be precluded. As such, the system would not be very useful as an investigative, research tool.

(2) The DDACS control software (for the LSI-11/03) has been designed to work with the standard DEC D/A and A/D boards (AAV11-A and ADV11-A). This involves, essentially, parallel communication. Thus, to use a serial interface, DDACS software changes would be required.

With the above considerations, it was decided that the best approach would be to use an interface that made the SDK-86 look like the standard DEC D/A (communication from LSI-11/03 to SDK-86) and A/D (communication from SDK-86 to LSI-11/03) boards. This involved interfacing the SDK-86 directly onto the LSI processor bus.

Such an approach means that no changes need be made to the DDACS control software and the data transfer speed will be more than adequate. In addition, there is the convenience of being able to treat the SDK-86 as just another (albeit intelligent) DEC peripheral.

Aside from the increase in complexity the approach chosen has one other disadvantage compared to a serial interface. It means that the two processors must be close to each other. This could prove unsatisfactory in an industrial, distributed system where a number of "low level" processors need to be located remotely from each other.

As well as the main interface described above (the IPI) a "status" interface is required to enable the LSI-11/03 to get necessary "loop" status data (see section 4.4.4).

4.4.3 "Analogue" Transfers through the Inter-Processor Interface

For a four stove system a minimum requirement is:

(1) Three analogue inputs per stove (two "temperature transducers" and one "differential pressure transducer"). That is 12 analogue inputs.

(2) Three analogue outputs per stove (one SCR for temperature control, and two armature current controllers). That is 12 analogue outputs.

4.4.4 Digital Transfers through the Inter-Processor Interface

As well as providing the appropriate control and feedback values, the SDK-86 needs to communicate with the LSI-11/03 regarding the operating status of each "loop". This can be done by using the PPI (programmable peripheral interface) of the SDK-86 directly interfaced with the DEC digital I/O unit. The information required by the LSI-11/03 can be encoded into 3 bits as follows:

(1) "Loop" status, auto or manual, using 1 bit.

(2) Two bits to inform the DDACS software of the "change state" ,

0 0	"Hold"
1 0	"Raise"
0 1	"Lower".

This information would be used by the DDACS system to alter setpoint values and manual control settings, as well as initialising the PI or PID controllers.

4.4.5 Priorities and Interrupt Considerations

The clock scheduler of the DDACS operating system is required to ensure that each SCHEME or task runs strictly at the desired sample rate. If delays were to occur, say in performing the A/D or D/A conversions, a "timeout" would follow, resulting in a system halt. Clearly this must be avoided.

It follows that A/D and D/A requests through the IPI to the SDK-86 must be given a high priority through interrupt control. The interrupt control circuitry uses a standard INTEL controller chip. This has been implemented by Mr. R.W. Korbel, together with a software "ring buffer" to stack interrupts when necessary.

Auto/Manual or Raise/Lower requests from the operator may be given a much lower priority. Hence no provision need be made for the SDK-86 to interrupt the DDACS system. Instead, regular polling of the auto/manual and raise/lower status bits by an appropriate DDACS SCHEME will be adequate.

4.5 SUMMARY

Summarizing the system, as defined so far, two processor levels can be defined:

(1) The higher level LSI-11/03 running the DDACS operating control software. This level supervises the various "loops" controlling such things as sampling rates, feedback values auto/manual and manual/auto transitions.

(2) The lower level SDK-86 which provides the operator interface (via keypad and LED display) and controls the A/D and D/A functions, as well as scaling and any pre-processing (or post-processing) of data. The operator must control the designation of "loops" as well as auto/manual transitions, setpoint values and output to manual "loops".

The interface between the two levels will be functionally divided into two areas:

(1) The IPI (Inter-Processor Interface) which will provide the high speed parallel communication path for D/A and A/D data and channel select control. This must provide for direct interfacing to the LSI bus so that the SDK-86 "looks" like standard DEC A/D and D/A modules. Thus the DDACS software will be directly compatible with the interface. The SDK-86 must be interruptable by DDACS.

(2) The "digital" interface which enables the LSI-11/03 to obtain required status information from the SDK-86 (and therefore, from the operator). This involves a direct interface between the SDK-86 PPI (peripheral processor interface) and the DEC digital interface.

5. PROCESSOR SYSTEM

5.1 INTRODUCTION

The two level processor system chosen for the control of the experimental stoves is described in this chapter. Essentially it consists of an 'upper level' PDP-11/03 interfacing with a 'lower level' INTEL SDK-86 microcomputer. The higher level control functions are handled by the PDP-11/03 under control of a software operating system called DDACS (Direct Digital Automatic Control System), developed by the Central Electricity Board, NE Region Scientific Services Department [10]. The lower level functions, including the house keeping of the D/A and A/D conversions, are handled by the SDK-86.

Note that the SDK-86 software was developed in two stages. Firstly, a simple, single loop control program was written with a view to testing the hardware and interface functions. In this stage the 'bumpless' auto/manual transfer facility was incorporated at the SDK-86 level. Secondly, as a result of a review of the system specifications (section 4) the SDK-86 software was reviewed. This later work was largely done by Mr. R. Korbel. In the stage two system, the auto/manual transfer facility was incorporated at the PDP-11/03 level.

The choice of the PDP-11/03 followed the decision to use the DDACS software since it was available (at the time) only in DEC MACRO-11 assembly language. This choice of software followed by processor is a curious turnabout and well illustrates the growing trend to avoid or minimise software effort. This reflects the increase in software development costs and the relative decrease in hardware costs.

5.2 HARDWARE SYSTEM STRUCTURE

5.2.1 Introduction

A diagram of the hardware system structure is shown in figure 5.1. From this it can be seen that the structure is hierarchical with the PDP-11/03 acting as a flexible higher level processor. It can handle slower PID control loops, stove sequencing and also supply to the 8086 set points for flow control.

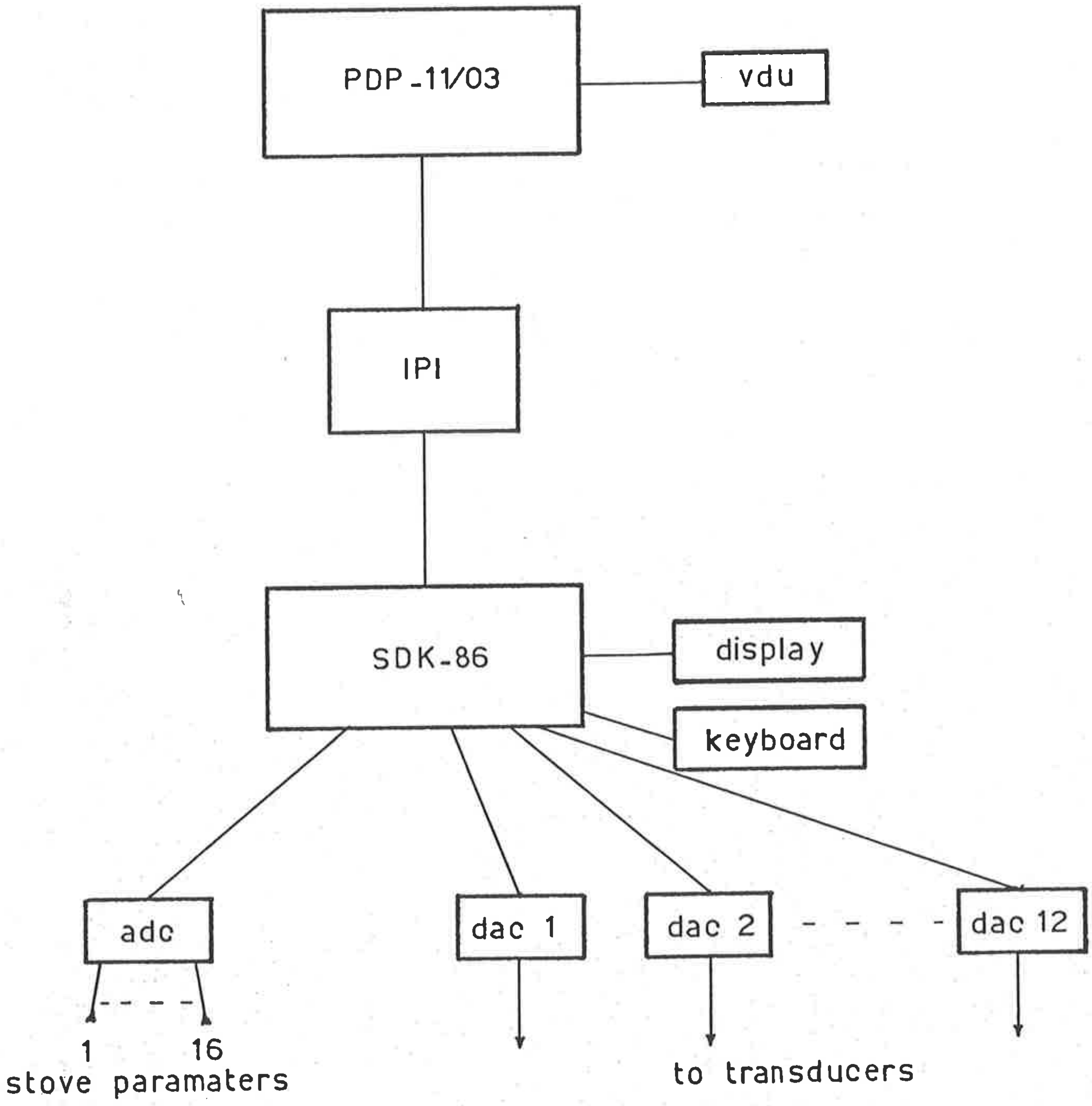
Control over the 16 ADC (analogue to digital convertor) and 12 DAC (digital to analogue convertor) channels is exercised by the 8086, as well as auto/manual and local/remote transfers, and set point ramping and display.

Note that the structure of figure 5.1 is readily expandable to a multi-processor system where the PDP-11/03 oversights several 8086 processors (see figure 5.2). The IPI (inter-processor interface) is designed so that each processor can readily be addressed as just another peripheral.

Details of all relevant circuits and diagrams have been included in appendix A.

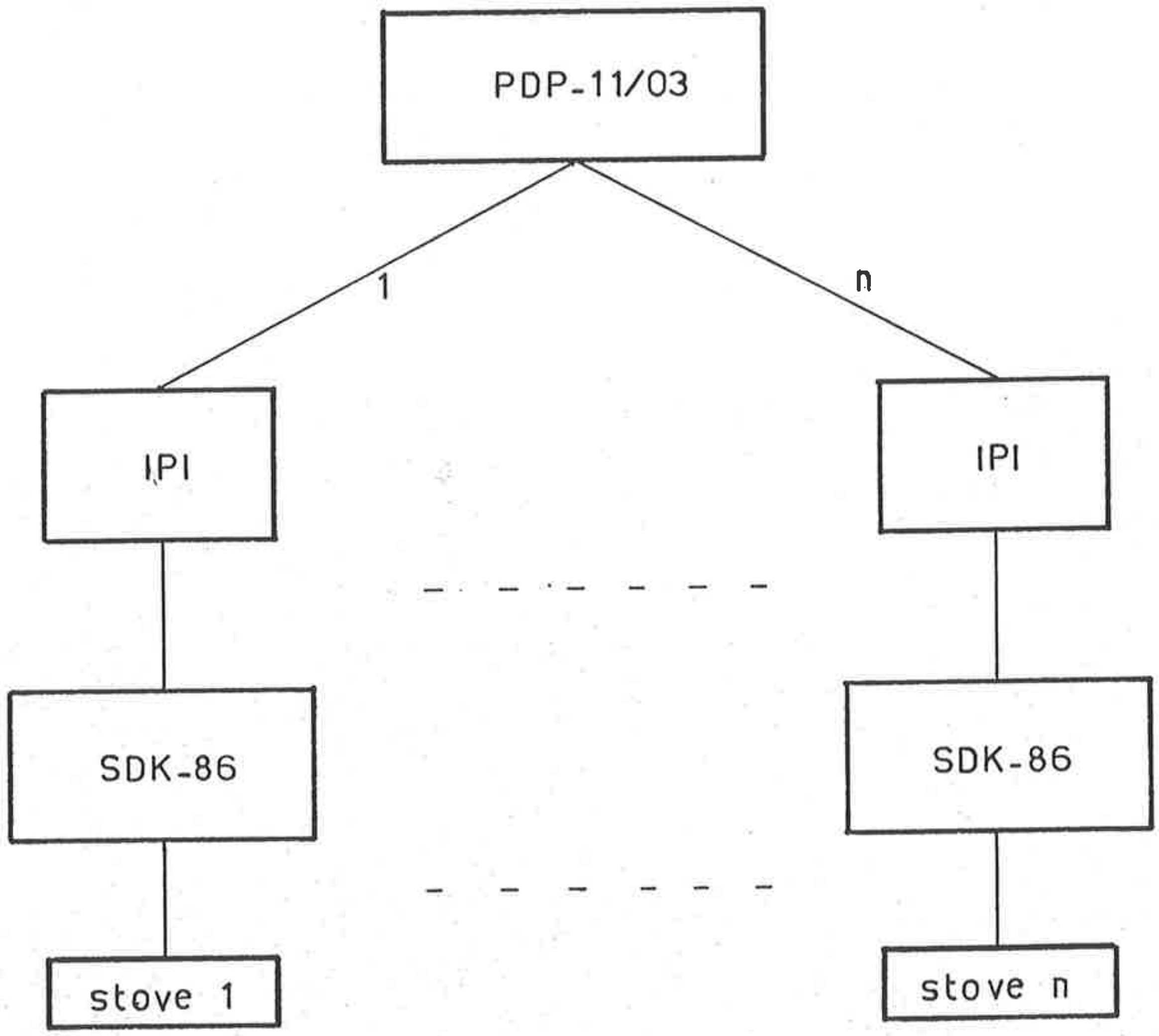
5.2.2 PDP-11/03 Microcomputer

This microcomputer is based on DEC's (Digital Equipment)



Control system Configuration

FIGURE 5.1



Multi-Processor Configuration

FIGURE 5.2

LSI-11 16 bit microprocessor. The maximum direct address space is 32K words.

A dual floppy disk drive provides the 'mass' storage area and operator interaction occurs via a standard RS-232 serial interface.

Because the assembly language is equivalent to that used in the standard PDP minicomputer series, the system software support is extensive. This will prove useful for further software development at this level. At present the DDACS control software system is to provide the higher level control facilities required. In particular this will include the 'feedforward' control of 'hot blow' flow rate to achieve maximum thermal efficiency.

5.2.3 SDK-86 Microcomputer

The SDK-86 is a small design board incorporating the INTEL 16 bit 8086 microprocessor. It has a direct address space of 1 Mbyte and provides sufficient computing power to handle the required 'low level' control of the DACs, ADCs and the setpoint ramping and display. As the system develops further this processor could take more load from the PDP by handling PID loops and also incorporating some degree of digital filtering of the A/D inputs.

A real time control application of the 8086 has already been reported by Newell and Bartlett [11]. Their system involves the use of the 8086 to provide an intelligent interfacing terminal which can be connected to any multi-user system. To achieve this flexibility a serial line is used between the 'host' computer and their 'intelligent' terminal.

This configuration was not possible in our system because of the real time responsibilities of the 'upper' level processor. To provide the necessary speed of communication a parallel inter-processor interface (IPI) was designed.

5.2.4 Inter-Processor Interface

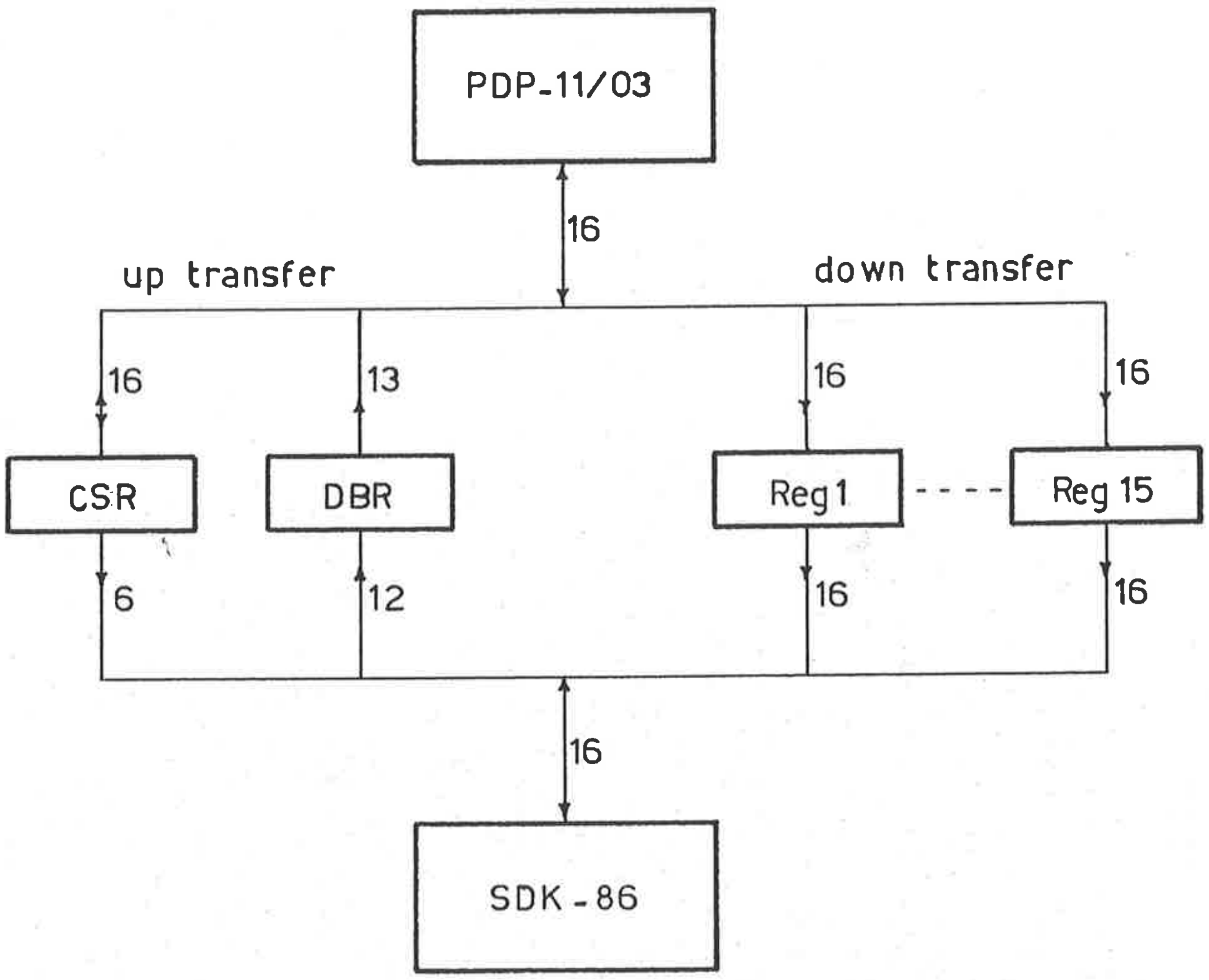
The hardware interface has been built to provide a parallel, high speed communications path between the two processors. This inter-processor interface (IPI) enables the PDP-11/03 to control the activities of the 8086, as it would any other device. The difference, of course, is that the 8086 can behave as a highly intelligent peripheral.

Although the data can be transmitted in both directions the mode of operation differs in each case and is controlled by two distinct sections of the IPI. This is described below with reference to figure 5.3.

The development of the IPI was the most time consuming part of the project, although conceptually simple. Because it is functionally simple the description that follows is short. The hardware details of the functional blocks can be found in appendix A.2.

5.2.5 Up Transfer

Data transfer from the 8086 to the PDP-11/03 is in the form



Inter-Processor Interface (IPI)

FIGURE 5.3

of 12 bit words and is handled by two intermediate IPI registers; the command status register (CSR) and the data buffer register (DBR). Both function in the same way as the CSR and DBR registers in the standard DEC analogue to digital convertor module (ADV11-A). This is a welcome convenience since it means that a programmer familiar with PDP systems is already equipped to write the interface control software.

When the PDP requires data from the 8086, it sets bit 0 (least significant bit) of the CSR and sends a channel address (bits 11, 10, 9, 8). This can specify one of 16 channels. The 8086 detects bit 0 set (either on a scan basis, or as an interrupt) and so knows that data is required from the specified channel. This data is written to the DBR (a 12 bit word). When this is received in the DBR bit 7 of the CSR is set and the 'start bit' (bit 0) is cleared.

The PDP can determine that data is available in two ways;

(1) firstly, it can scan the CSR and test bit 7 or,

(2) it can 'condition' the transfer to operate on an interrupt basis by previously setting bit 6. In this case bit 7 will generate an interrupt when set.

There is also facility for setting an error flag (bit 15 of the CSR) when the PDP;

(1) attempts to request data before a previous request is honoured or,

(2) fails to read requested data before further data arrives.

Bit 15 can also be made to interrupt the PDP by setting bit 14 (i.e. interrupt on error).

5.2.6 Down Transfer

Figure 5.3 illustrates the 'down transfer' section of the IPI. This enables 16 bit words of data to be transferred from the PDP-11/03 to the 8086, along one of 15 separate parallel channels through intermediate registers.

The PDP-11/03 simply writes to each register as a separate memory location whenever it is necessary to send data. There is no direct facility to inform the 8086 when data has been sent, although bits in the CSR could be used for this purpose.

However, this added complexity was not considered necessary. The 8086 need only treat these registers as the source of predefined (by software) parameters and data which it reads as necessary (eg. flowrate value). The PDP-11/03 is left responsible for updating these registers.

5.2.7 A/D and D/A Convertors

For a full four stove system 12 A/D and 12 D/A channels would be required. To meet these requirements Analogue Device's AD363 data acquisition system was chosen to perform the A/D conversions and Burr-Brown DAC80 D/A convertors were chosen to perform the D/A function (see appendix A.4 for the relevant data sheets).

Both devices are 12 bit convertors and have been incorporated on one SBC-80 board (INTEL standard). The hardware details, including addressing information etc. has been included in appendix A.

The AD363 includes a 16 channel multiplexer and control logic to provide 16 single-ended or 8 differential inputs. Its throughput is typically 30kHz, thus meeting the necessary speed requirements.

5.3 SOFTWARE

5.3.1 Introduction

From the hardware structure described, it can be seen that two software 'packages' are required. For the PDP-11/03 a software operating system called DDACS (Direct Digital Automatic Control System) was employed. As has been mentioned, this has been developed by the Central Electricity Board, NE Region Scientific Services Department as a general purpose operating system for control applications.

5.3.2 PDP-11/03 Software (DDACS)

A detailed description of DDACS, including operating instructions, can be found in the manual written by the software authors [10]. A summary of DDACS operation and facilities has been incorporated in appendix B.1. Essentially it is a self-contained operating system designed to utilise a 'building block' concept to implement general purpose control systems in a fairly straight forward manner.

The operating system includes a real time executive with clock scheduler, an editor and the DDACS compiler. Using the editor the building 'blocks' (in the main these are simply calls to subroutines from a standard library) can be linked together to form a 'loop' which runs at a specified time rate. Any number of these control 'loops' can be brought together in a 'scheme'. While selected 'schemes' are running the operator can be constructing/altering other 'schemes' as a background function.

There are 70 'blocks' available for building the control loops and include the usual arithmetic and logic operators, an integrator, a first order lag and an absolute PID controller block, input/output blocks and functions such as SQRT, SIN, COS, EXP etc.

As well as the facility to reconfigure 'schemes' while on-line, DDACS provides a number of other useful features; full propagation of data errors in a fail safe manner, automatic sequencing of 'loops' of different period and the capability to prevent reset or integral wind-up and to ensure bumpless manual/auto transfer.

Although a machine independent version of DDACS is under development (in CORAL 66) the current version was written in DEC MACRO-11 assembly language. Hence the choice of 'high level' processor.

5.3.3 SDK-86 Software

The first version of the SDK-86 software (stage 1) was written to handle one control loop only, to enable simple testing of the hardware and processor interaction. The auto/manual transfer facility was incorporated at this level during stage 1. The software was written in PLM-86 (a high level block structured language developed by INTEL) and was designed using a 'state machine' approach. Each keyboard entry is assigned a number which acts as a pointer in the current 'state' of the 'parser table'. The entry in this table determines the next 'state' to enter and the required action to be taken (if any). A listing of the software and a description of the 'parser table' are included in appendices B.2 and B.3.

By appropriate keyboard selections the operator can;

(1) examine the setpoint value (or transducer input if in manual) as a percentage of full span,

(2) change the above values in selected increments or decrements,

(3) perform auto/manual transfers.

The software was written to incorporate the algorithms as describe in section 5.4.

Stage two of the SDK-86 involved a total revision of the software to more accurately reflect the newly defined specifications (chapter 4). Again it was developed using PLM-86 (largely by Mr. R. Korbel). Because of the existing comprehensive facilities provided by DDACS the auto/manual transfer facility was incorporated at the higher level in stage two. Software description and listings are included in appendices B.4 and B.5.

5.4 OPERATION - STAGE 1

5.4.1 Introduction

The operator, under normal conditions, interacts with the control process via the keyboard and 8 digit display of the SDK-86. The keyboard will be used to select control loops, to display and, if necessary, alter set points (when in auto mode) or loop outputs (when in manual mode). The actual keyboard operation required to accomplish the above functions, under the current software, is detailed in appendix B.

The selection of auto or manual modes (described below) is also accomplished via this keyboard.

When the 'feedforward' control procedure is incorporated (thermal efficiency control) the operator will also be able to use the keyboard to select 'local' or 'remote' operating schemes for 'hot blow' auto loops. Under 'local' mode, the 'hot blow' flowrate set point is set by the operator. Under 'remote' mode it will be provided by the PDP-11/03 in accordance with its 'feedforward' calculations.

The 8 digit display on the SDK-86 board is used to indicate the operating mode and the set point or regulator output value.

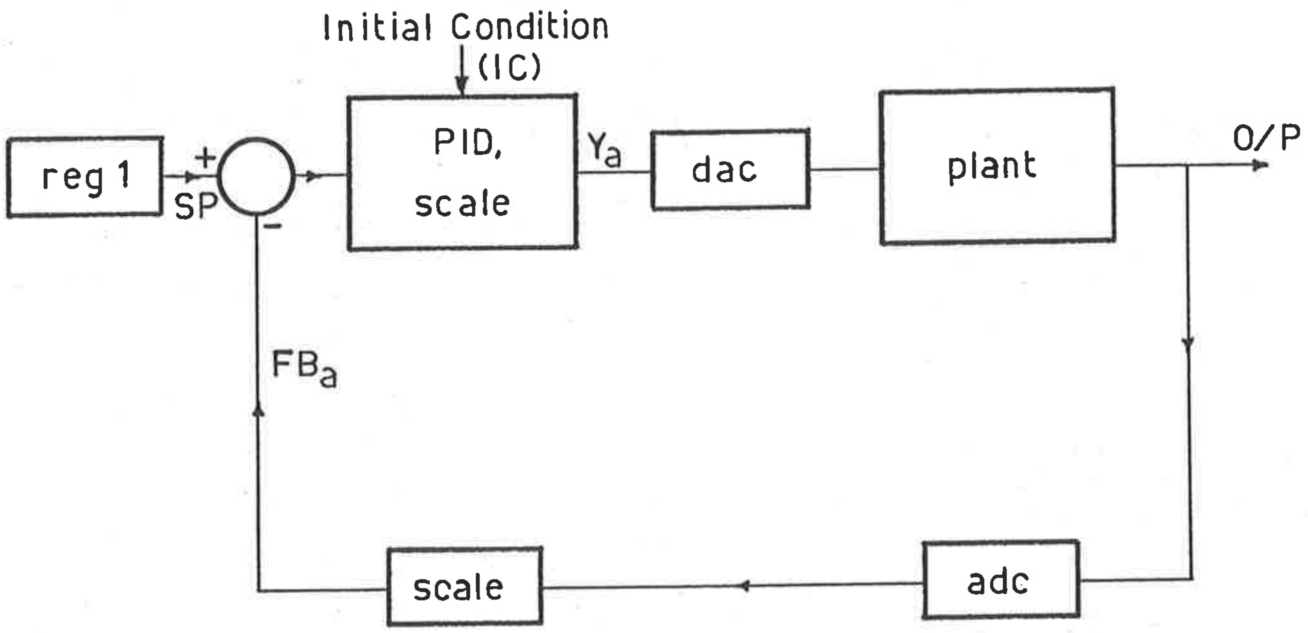
5.4.2 Auto/Manual Transfer

The existence of the auto/manual option serves two functions. It provides a convenient facility for 'start up' of the control system by enabling the operator to manually bring the stoves into an acceptable operating region before switching to auto control. Secondly, once auto operation is achieved, individual loops can be singled out for manual control when desired; this manual back-up is essential in an industrial system in case of failure of the higher level control functions.

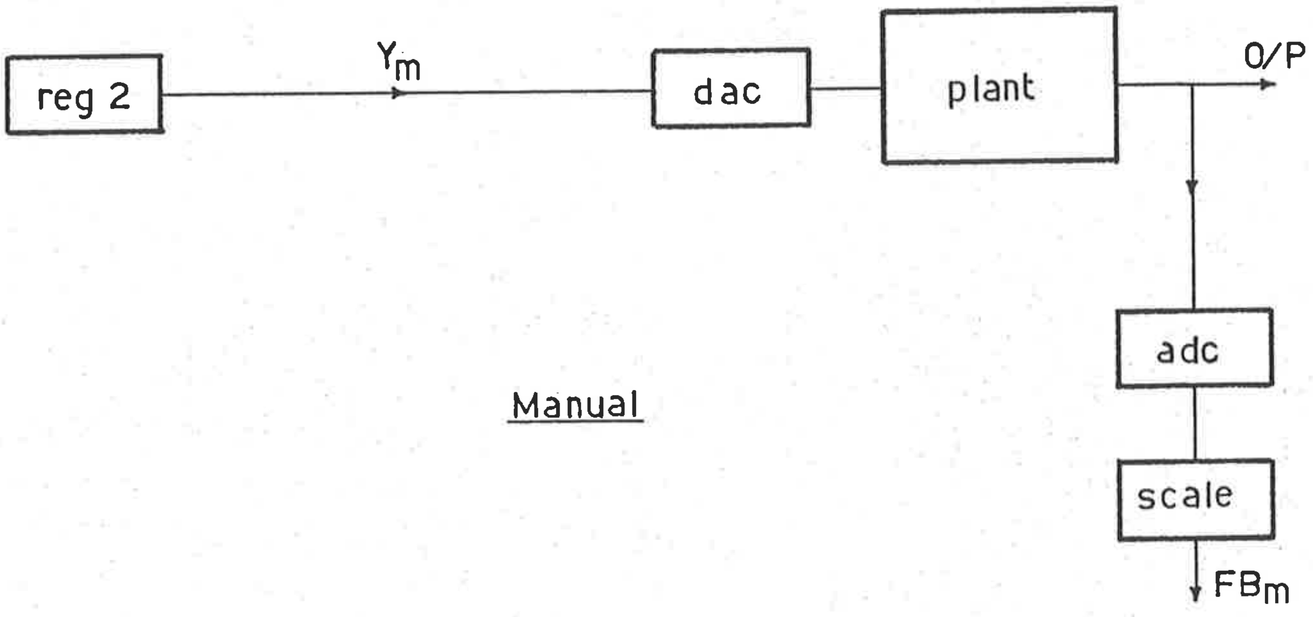
In order to understand operation in each of the modes, consider figure 5.4. During auto operation a set point value (SP) is taken from reg 1. This becomes the input of a single feedback control loop.

During manual operation the input (taken from reg 2) directly controls the output to the 'final control element'. There is no feedback as in the auto case.

The problem then presents itself as to how switching between the two modes is to be accomplished. The essential of such a transfer is that there be no 'jump' in the plant output. In some analogue controllers this 'bumpless' transfer is dependent upon the operator properly adjusting set points



Auto



Manual

Auto/Manual Operation

FIGURE 5.4

before switching. It is clearly more desirable to make smooth transition independent of what the operator may do.

The simple, yet effective, scheme of figure 5.5 was developed to deal with this. Essential to this scheme is the method adopted to change set points (reg 1 and 2 contents). Rather than loading a set point value directly into the registers (reg), a value (entered via the SDK-86 keyboard) is used to increment or decrement their contents.

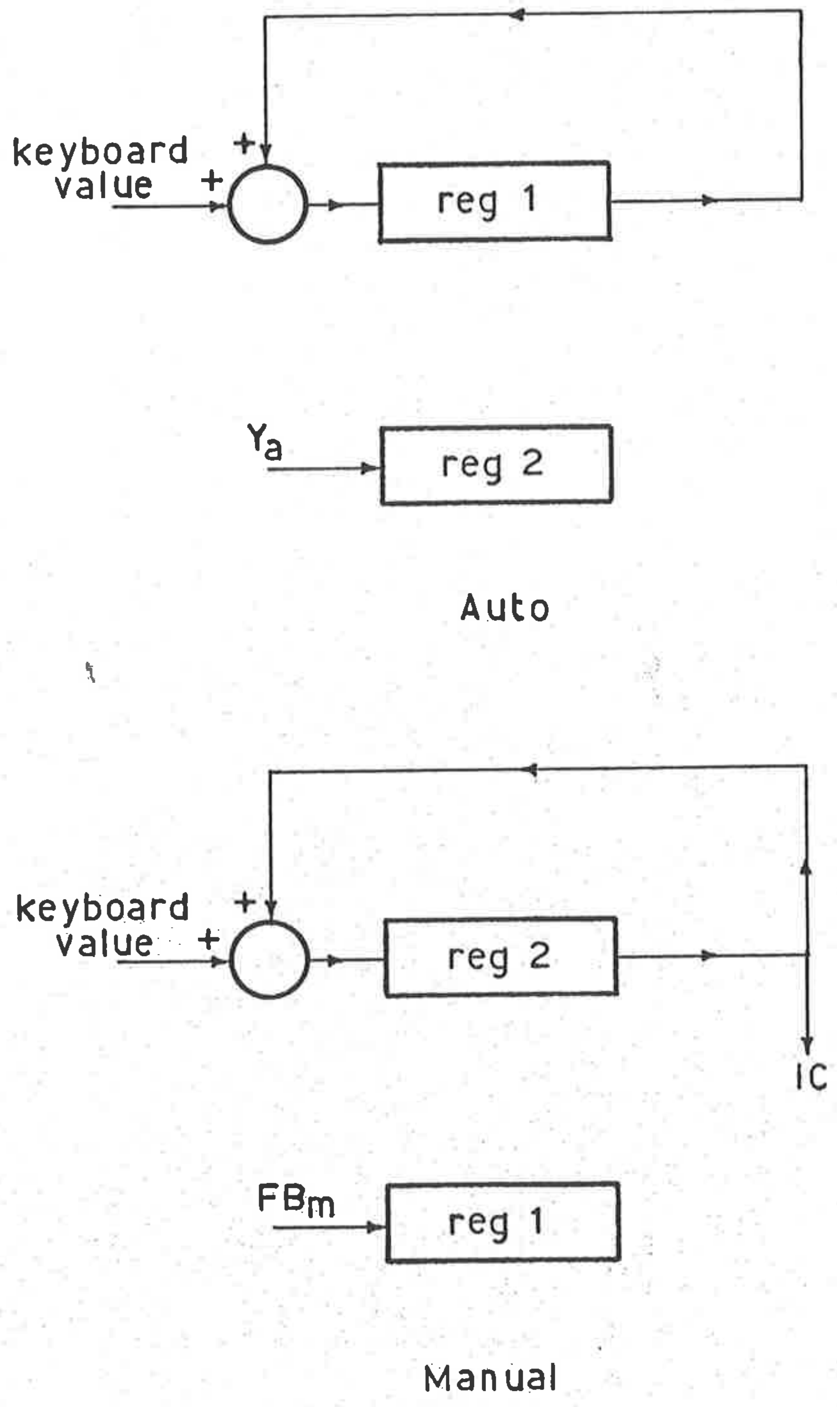
When in auto mode (figures 5.4 and 5.5) Y_a is continually used to update reg 2. Thus when a switch is made to manual, reg 2 contains the last PID output which then becomes the plant input. This means that the plant input is unchanged during transfer.

During manual operation Y_m is used to update the PID block IC (initial condition) and FB_m is used to update reg 1. This ensures that, when transfer back to auto is made, the output of the plant is unchanged and the error input to the PID block is zero. Thus again 'bumpless' transfer is accomplished.

5.5 OPERATION - STAGE 2

5.5.1 Introduction

This software was developed largely by Mr. R. Korbell. It is relevant to this thesis, however, in that it forms part of the overall system as originally conceived. Also, it has made it possible to confirm the feasibility of the system



Auto/Manual Transfer

FIGURE 5.5

for use as a 'control research tool'.

Again the operator interacts via the keypad, as in stage 1. This stage of software, however, enables multiple loops to be controlled. Basically the operator assigns a 'loop number' to a DAC and ADC channel (not necessarily the same channel numbers). In this way a given control loop can be identified by such a 'loop number'. Having done this, set points can be examined and changed and auto/manual transfer can be initiated.

The DDACS 'loops' must of course be configured in manner consistent with the SDK-86 'loop' assignments.

All loop processing (i.e. determination of DAC output values in a feedback loop, auto/manual transfer and any higher order processing) is still, of course, done by DDACS.

5.5.2 Operation

The low level process control is done using the keypad on the SDK-86. Three levels (or modes) of operation exist:

(1) The Select Mode. This is the 'highest' level mode. Basically, it provides the operator the means to enter either of the other modes (channel or loop). This is accomplished as follows;

- press "," to toggle between the 'channel' select state and the 'loop' select state (not mode),
- press a digit to define the 'channel' or 'loop' number of interest,

- press "." to enter the mode as defined above.

(2) The Channel Mode. This mode is always associated with a particular channel number as defined in the 'Select Mode'. From here the operator can examine the percentage span or hexadecimal value of the specified DAC (output value) or ADC (input value). This is done as follows;;

- press ":" to toggle between percentage span and hexadecimal display,
- press "," to toggle between ADC and DAC display.

In addition the following commands are available;

- press "+" to examine the next channel,
- press "-" to examine the previous channel,
- press "." to return to the 'Select Mode'.

In each case the variables being displayed (including the channel number) are identified on the display.

(3) The Loop Mode. This mode is always associated with a particular loop number as specified in the 'Select Mode'. From here the operator can assign any ADC and DAC channel to the current loop. In addition auto/manual transfer and setpoint changes can be effected. The commands are as follows;

- press "+" to raise the setpoint (ramp),
- press "-" to lower the setpoint (ramp),
- press ":" to cancel the raise or lower functions,
- press "REG" to initiate the 'loop assignment' procedure; this is followed by "DAC channel number",

".", "ADC channel number", "." to complete the assignment,

- press "," to toggle between 'auto' and 'manual' loop status,
- press "." to return to the 'Select Mode'.

In each case the variables being displayed are identified on the LED display.

5.6 SYSTEM PERFORMANCE

Although exhaustive tests have not been performed on the system the criteria specified in chapter 4 have all been met. In particular, the throughput of the IPI is sufficiently high to cope with the maximum input/output capability of the PDP-11/03.

6. CONCLUSION

A basic system structure for control of a laboratory model of a blast furnace stove has been designed and developed. The hardware has been built and tested. The IPI is sufficiently fast so as not to limit PDP-11/03 input/output speed.

Stage 2 of the software is complete and has enabled initial loop tests to be tried, demonstrating the feasibility of the system.

The result has been that sufficient processing power and interface speed is available to produce a flexible research tool that can readily handle more demanding control applications.

The system configuration chosen is such that it can readily be adapted to a multiprocessor system of one low level microcomputer per stove; this would be the recommended approach for an industrial installation. In addition, the processing power of the 8086 lends itself to the possibility of further processing of data at the low level. Facilities that could be added include;

(1) handling some local PID loops,

(2) noise filtering of A/D input data,

(3) testing for validity of input data,

(4) limiting inputs and performing other functional mapping.

In retrospect, a number of observations can be made with regard to the project. Firstly, because of the time extent of the work, technological progress in the electronics industry makes some of the choices seem inappropriate. For example, the 8086 processor was the only 16 bit microprocessor on the market at the time of decision. Since then, new developments have made other processors available which may have proved more appropriate (eg. the MC68000, see reference [14]).

However, this problem is characteristic of any longer term development in this field.

Secondly, a need exists for a rebuilding of the hardware (IPI and the D/A, A/D boards). The prototypes have proven the validity of the design (in light of the specifications) but are a definite maintenance liability. The preferred approach would be to develop printed circuit board assemblies.

REFERENCES

1. J. Beets, H. Elshout and G. deJong, "Computer Control of a Hot Blast Stove System," Journal A (Belgium), vol. 18, no. 1, pp 31-37, 1976.

2. C.P. Jeffreson, "A Computer Control System for Blast Furnace Stoves," Seventh Australian Conference on Chemical Engineering, pp 22-24, Aug. 1979.

3. P. Budimir and C.P. Jeffreson, "Microprocessor Control of an Experimental Stove System," Conference on Microprocessor Systems, pp 107-110, Nov. 1979.

4. J.C. Buker and N.F. Simcic, "Blast Furnace Stove Analysis and Control," ISA Trans., vol. 2, pp 160-167, 1963.

5. H. Kwakernaak, P. Tijssen and R.C.W. Strijbos, "Optimal Operation of Blast Furnace Stoves," Automatica, vol. 6, pp 33-40, 1972.

6. C.P. Jeffreson, "Feedforward Control of Blast Furnace Stoves," Automatica, vol. 15, pp 149-159, 1979.

7. M. Jakob, "Heat Transfer," vol. 2. New York: John Wiley, 1957.

8. P. Zuidema, "Non-stationary Operation of a Staggered Parallel System of Blast Furnace Stoves," Int. J. Heat Mass Transfer, vol. 15, pp 433-442, 1972.

9. H. Kwakernaak, R.C.W. Strijbos and P. Tijssen, "Optimal Operation of Thermal Regenerators," IEEE Trans. Automatic Control, vol. 14, pp 728-731, 1969.

10. L.R. Johnstone, C.R. Marsland and S.T. Pringle, "A Distributed Computer Control System for a 120 MW Boiler," Proc. Intern. Conf. on Distributed Computer Control (Inst. Elec. Engrs. London), pp 114-119, 1977.

11. R.B. Newell and E.G. Bartlett, "A Computer Independent Real-Time Process Interface Unit for Multi-User Computers," Aust. Jl. Inst. Control, pp 56-60, June 1979.

12. C.P. Jeffreson, "Dynamic Simulation of Thermal Regenerator Systems under Variable Flow Conditions," accepted for publication subject to revision, The Chem. Eng. Jl., 1981.

13. Personal Communication.

14. R.D Grappel and J.E. Hemenway, "A Tale of Four MPU's: Benchmarks Quantify Performance," EDN Magazine, April 1, 1981.

APPENDIX A

HARDWARE DESCRIPTION

APPENDIX A

A.1 INTRODUCTION

The 'lower level' processor and associated hardware reside on three boards;

(1) the SDK-86 comes on a single board,

(2) the ADC, DACs and decoding hardware resides on a standard SBC-80 board,

(3) the IPI resides on a standard SBC-80 board.

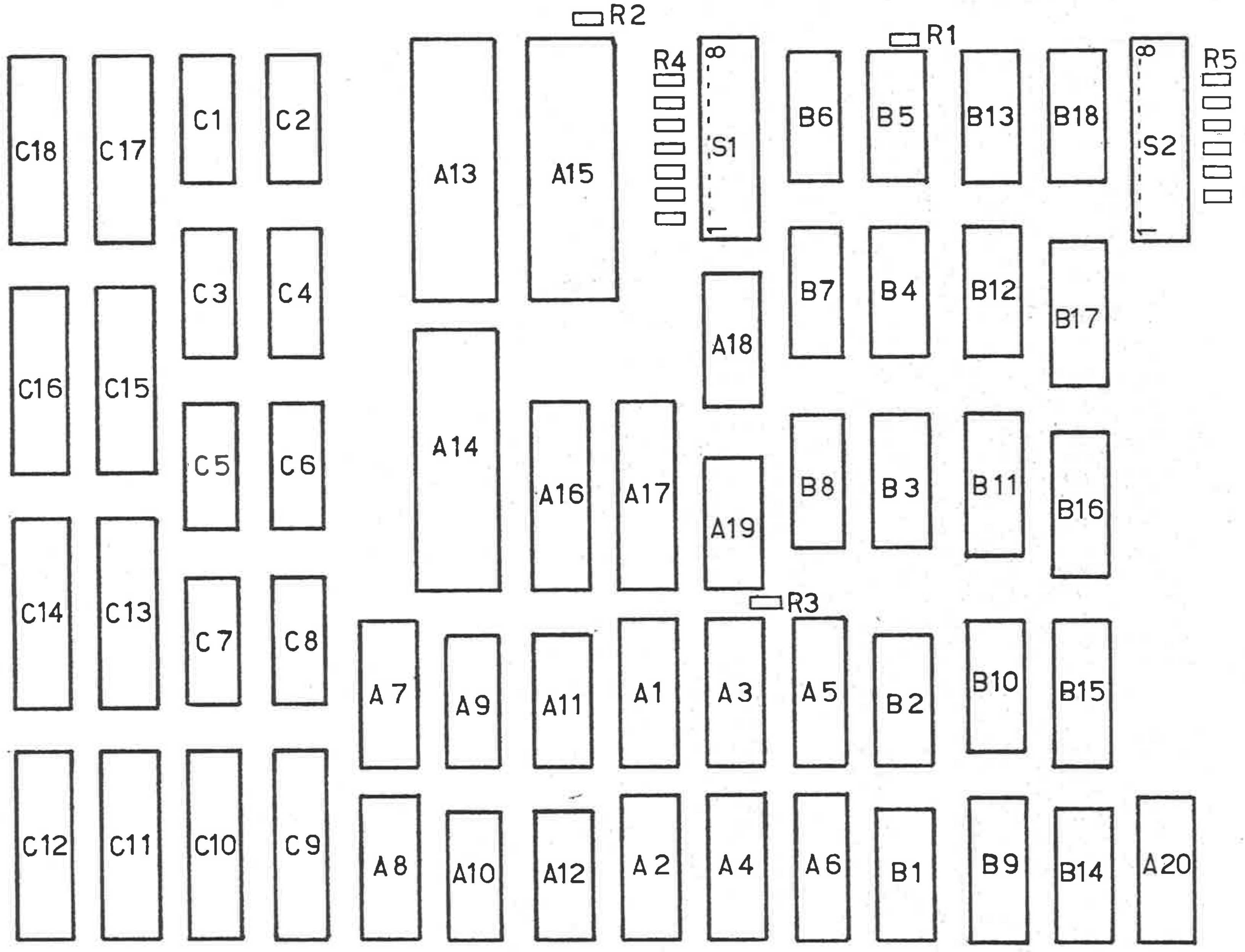
These all fit into a four slot carriage with a common back plane which shares the SDK-86 signals (described in A.2.2).

The power supplies are separate units.

A.2 INTER-PROCESSOR INTERFACE BOARD

APPENDIX A

A.2.1 BOARD LAYOUT



SDK-86 signals PDP signals

A.2.2 EDGE CONNECTIONS

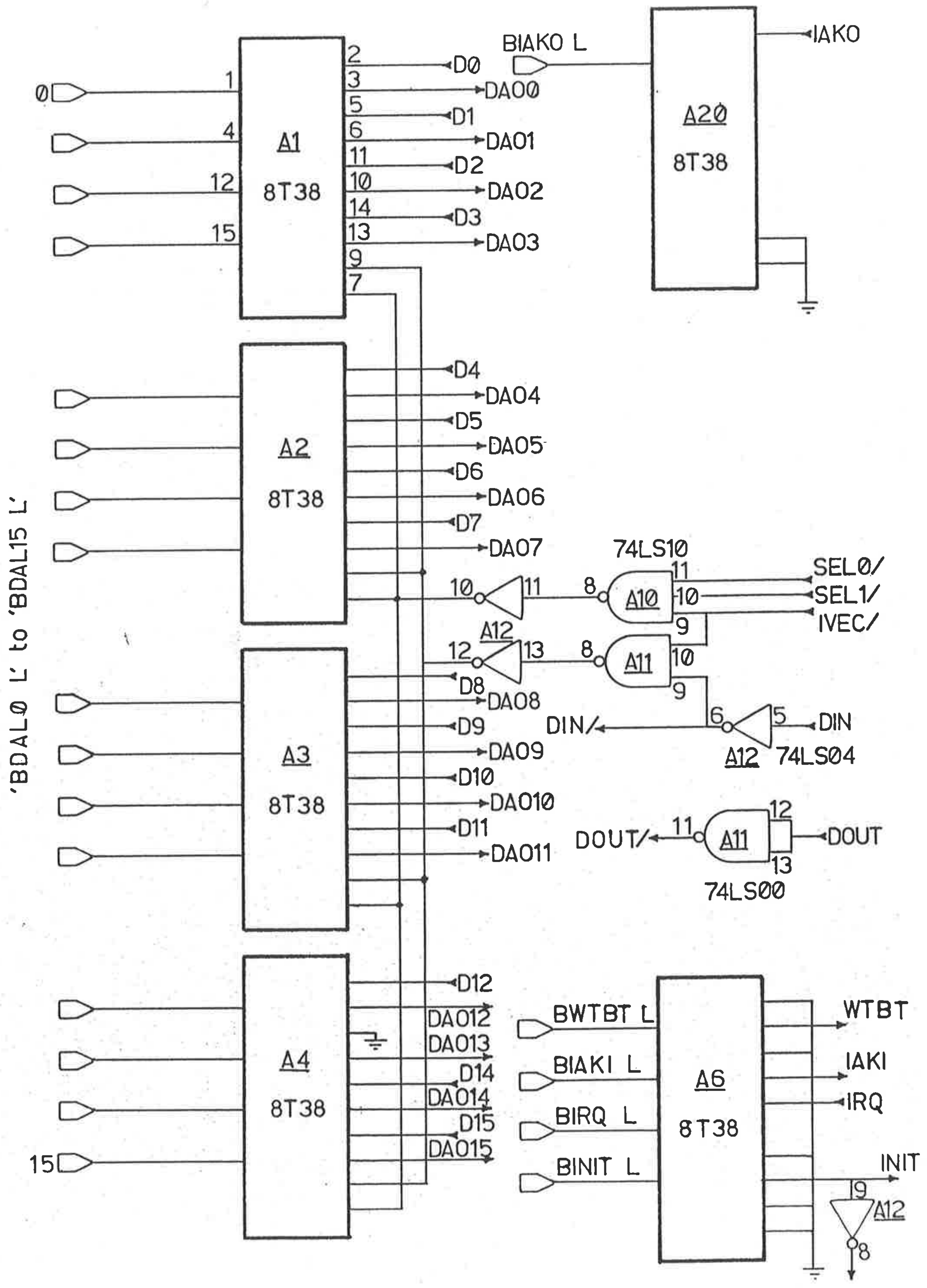
PDP SIGNALS

PIN NUMBER	SIGNAL	PIN NUMBER	SIGNAL
1	BDAL0 L	3	BDAL1 L
5	BDAL2 L	7	BDAL3 L
9	BDAL4 L	11	BDAL5 L
13	BDAL6 L	15	BDAL7 L
17	BDAL8 L	19	BDAL9 L
21	BDAL10 L	23	BDAL11 L
25	BDAL12 L	27	BDAL13 L
29	BDAL14 L	31	BDAL15 L
33	BWTBT L	35	BDOUT L
37	BRPLY L	39	BDIN L
41	BSYNC L	43	BIRQ L
45	BIAKI L	47	BINIT L
49	BIAKO L	EVEN PINS	EARTH

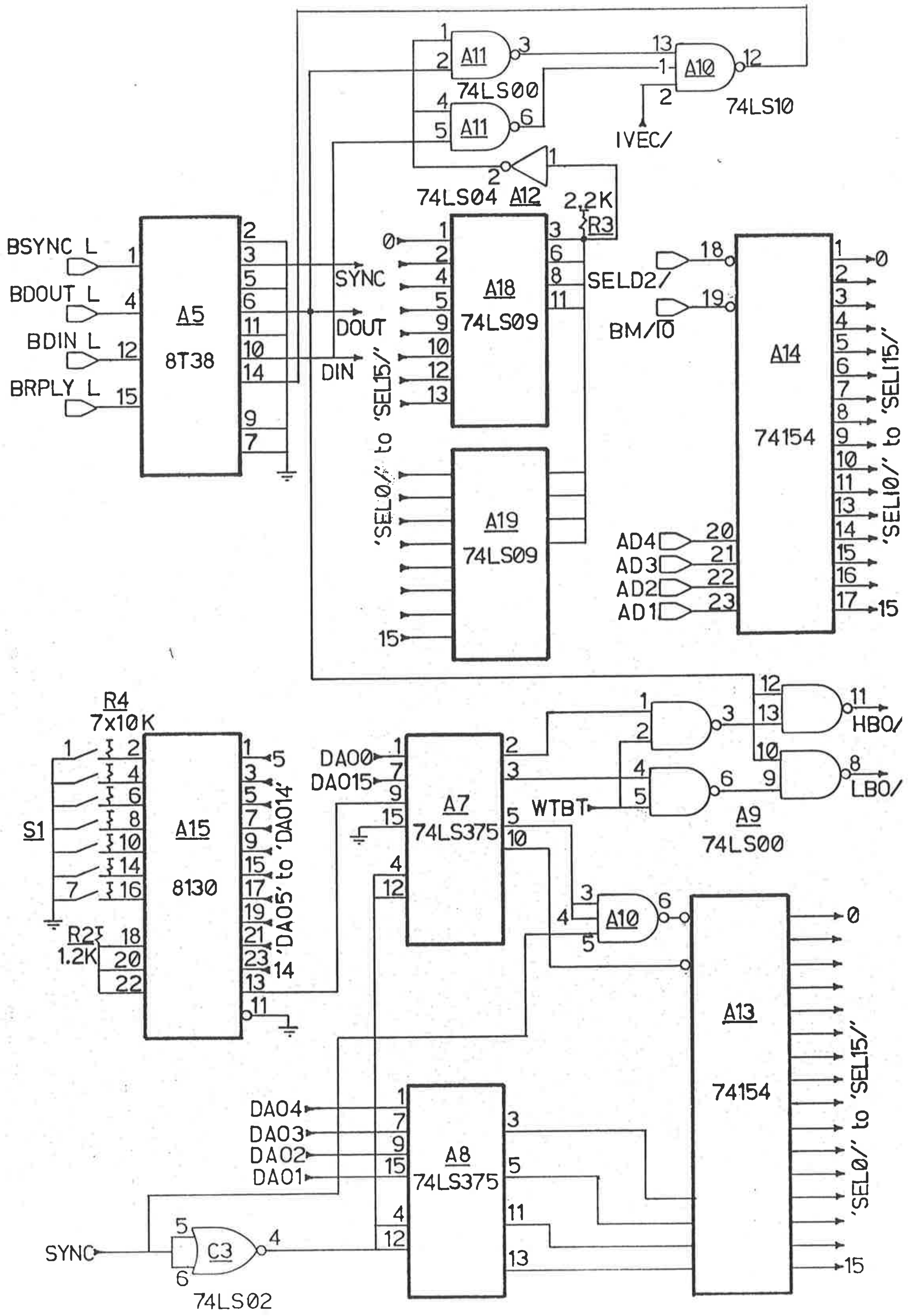
SDK-86 SIGNALS

PIN NUMBER	SIGNAL	PIN NUMBER	SIGNAL
1	EARTH	2	EARTH
3	+ 5V	4	+ 5V
5	+ 5V	6	+ 5V
11	EARTH	12	EARTH
13	+ 15V	14	+ 15V
15	EARTH	16	EARTH
17	- 15V	18	- 15V
34	INTR	35	SELD0/
36	SELD1/	37	SELD2/
38	SELD3/	39	BM/IO
40	BRD/	41	BWR/
43	AD15	44	AD14
45	AD13	46	AD12
47	AD11	48	AD10
49	AD9	50	AD8
51	AD7	52	AD6
53	AD5	54	AD4
55	AD3	56	AD2
57	AD1	58	AD0
59	BD15	60	BD14
61	BD13	62	BD12
63	BD11	64	BD10
65	BD9	66	BD8
67	BD7	68	BD6
69	BD5	70	BD4
71	BD3	72	BD2
73	BD1	74	BD0
75	EARTH	76	EARTH
81	+ 5V	82	+ 5V
83	+ 5V	84	+ 5V
85	EARTH	86	EARTH

A.2.3 ADDRESS DECODING AND BUS INTERFACE

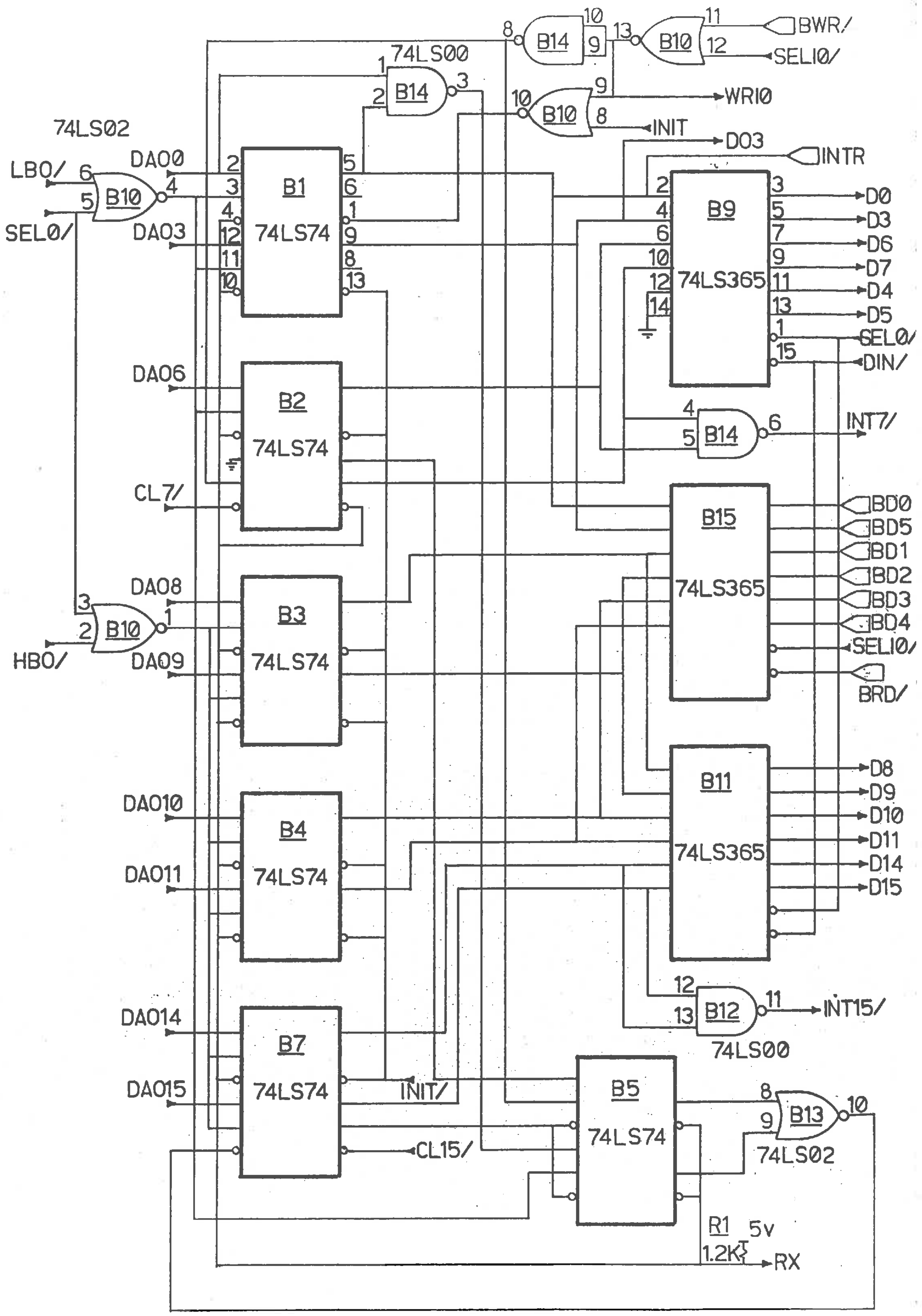


0
A1
8T38
1
4
12
15
2
3
5
6
11
10
14
13
9
7
D0
DA00
D1
DA01
D2
DA02
D3
DA03
BIAKO L
A20
8T38
IAKO
A2
8T38
D4
DA04
D5
DA05
D6
DA06
D7
DA07
74LS10
A10
SEL0/
SEL1/
IVEC/
A12
A11
DIN/
DIN
74LS04
'BDAL0 L' to 'BDAL15 L'
A3
8T38
D8
DA08
D9
DA09
D10
DA010
D11
DA011
DOUT/
DOUT
74LS00
A4
8T38
15
D12
DA012
DA013
D14
DA014
D15
DA015
BWTBT L
BIAKI L
BIRQ L
BINIT L
A6
8T38
WTBT
IAKI
IRQ
INIT

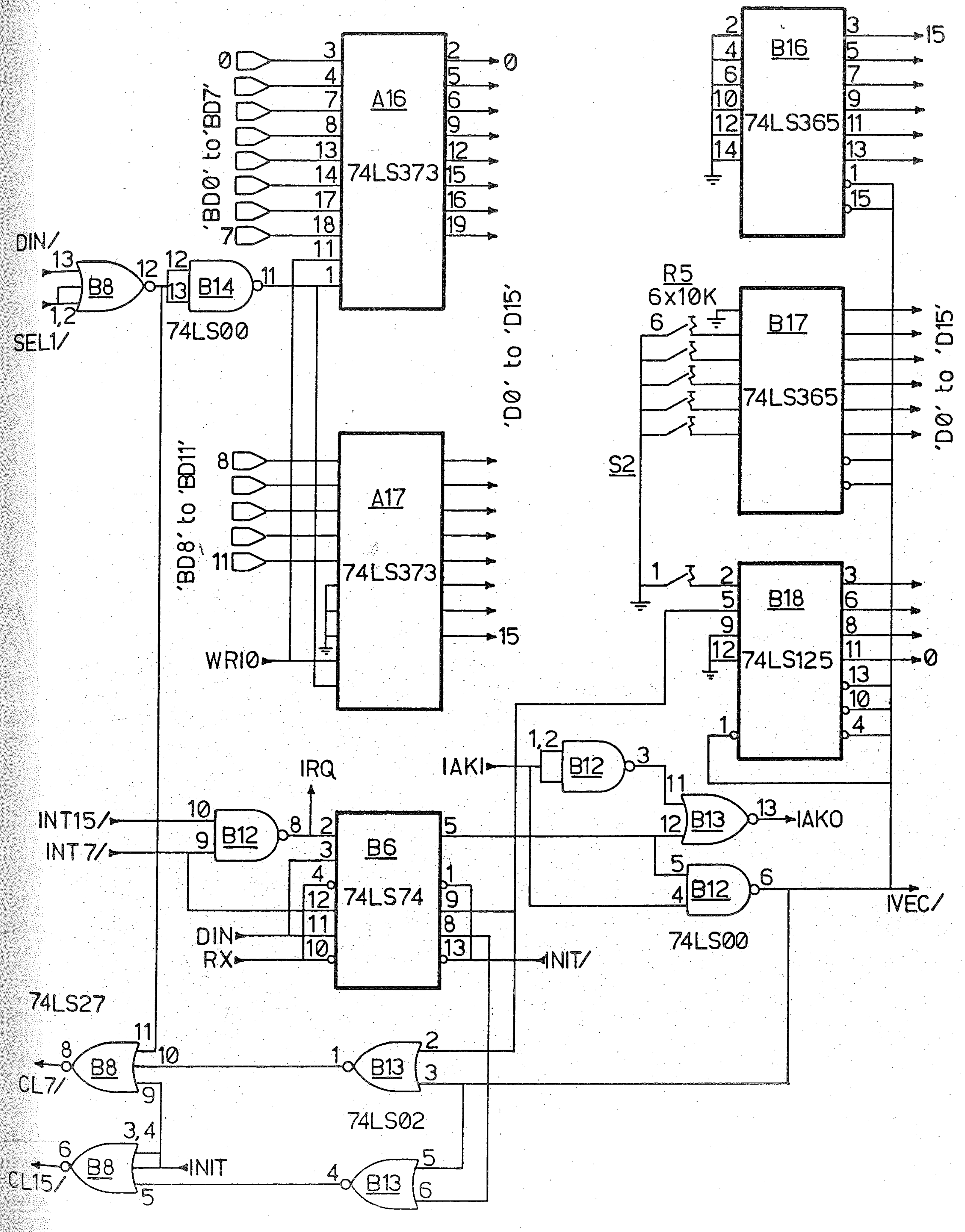


BSYNC L
BDOUT L
BDIN L
BRPLY L
A5
8T38
SYNC
DOUT
DIN
A11
74LS00
A11
A10
74LS10
IVEC/
74LS04 A12
2,2K
R3
A18
74LS09
SELD2/
A19
74LS09
'SEL0/' to 'SEL15/'
BM/IO
A14
74154
AD4
AD3
AD2
AD1
'SEL0/' to 'SEL15/'
R4
7x10K
S1
R2
1.2K
A15
8130
'DA05' to 'DA014'
DA00
DA015
A7
74LS375
WTBT
A9
74LS00
HBO/
LBO/
A10
A13
74154
'SEL0/' to 'SEL15/'
DA04
DA03
DA02
DA01
A8
74LS375
SYNC
C3
74LS02

A.2.4 UP TRANSFER (CSR AND DBR HARDWARE)

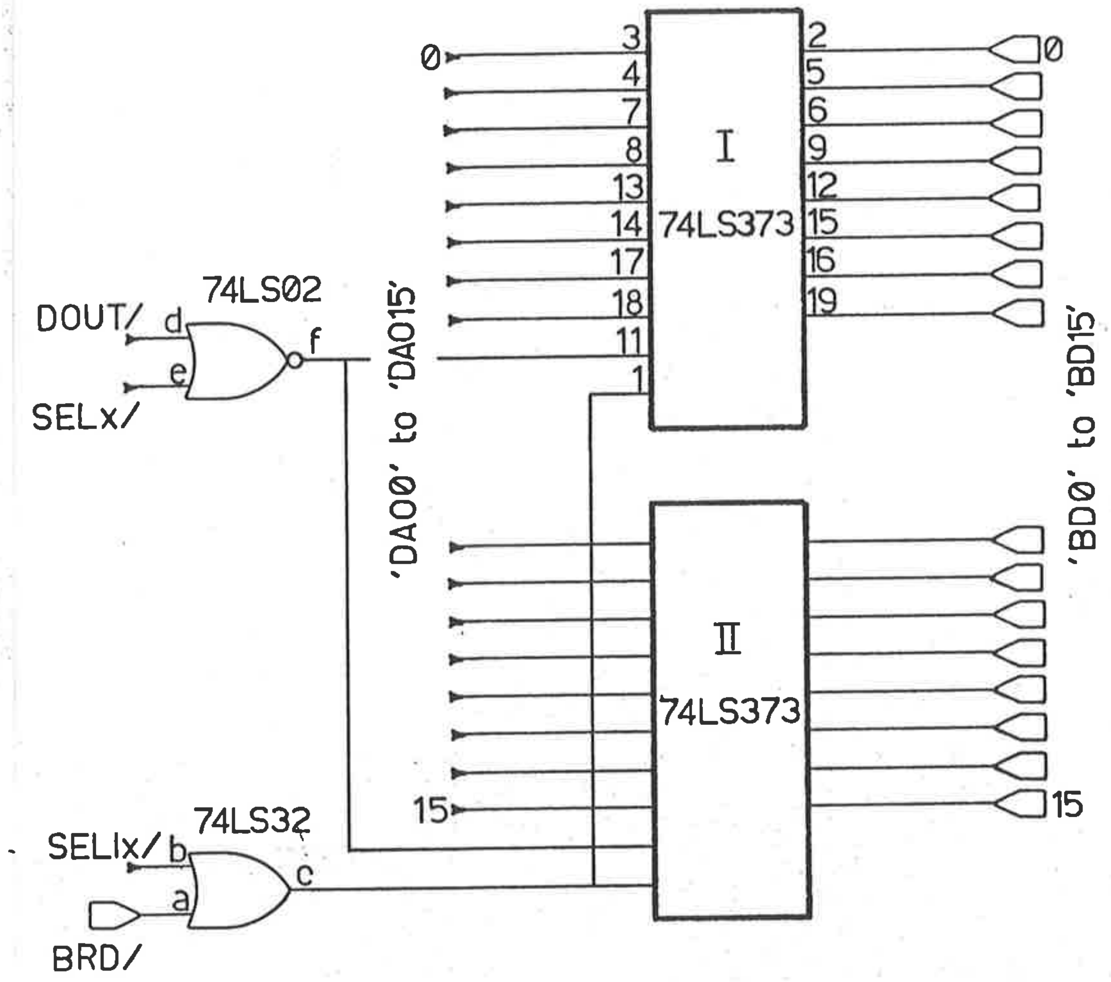


74LS00
74LS02
B14
B10
BWR/
SEL10/
WR10
INIT
D03
INTR
LB0/
SEL0/
DA00
DA03
B1
74LS74
B9
74LS365
D0
D3
D6
D7
D4
D5
SEL0/
DIN/
DA06
B2
CL7/
INT7/
B15
BD0
BD5
BD1
BD2
BD3
BD4
BRD/
DA08
DA09
HB0/
B3
B4
DA010
DA011
B11
D8
D9
D10
D11
D14
D15
B12
INT15/
DA014
DA015
B7
INIT/
CL15/
B5
B13
R1
1.2K
5v
RX



A16
74LS373
A17
74LS373
'BD0' to 'BD7'
'BD8' to 'BD11'
'D0' to 'D15'
B16
74LS365
B17
74LS365
B18
74LS125
R5
6x10K
S2
DIN/
SEL1/
B8
B14
74LS00
WRIO
IRQ
INT15/
INT7/
B12
B6
74LS74
DIN
RX
INIT/
IAKI
B13
IAKO
IVEC/
74LS00
74LS27
CL7/
CL15/
INIT
74LS02

A.2.5 DOWN TRANSFER HARDWARE



<u>KEY</u>

x		a	b	c		d	e	f	I	II
1	<u>C5</u>	2	1	3	<u>C1</u>	3	2	1	<u>C9</u>	<u>C10</u>
2	<u>C5</u>	5	4	6	<u>C1</u>	6	5	4	<u>C11</u>	<u>C12</u>
3	<u>C5</u>	12	13	11	<u>C1</u>	11	12	13	<u>C13</u>	<u>C14</u>
4	<u>C5</u>	9	10	8	<u>C1</u>	8	9	10	<u>C15</u>	<u>C16</u>
5	<u>C6</u>	2	1	3	<u>C2</u>	3	2	1	<u>C17</u>	<u>C18</u>
6	<u>C6</u>	5	4	6	<u>C2</u>	6	5	4		
7	<u>C6</u>	12	13	11	<u>C2</u>	11	12	13		
8	<u>C6</u>	9	10	8	<u>C2</u>	8	9	10		
9	<u>C7</u>	2	1	3	<u>C3</u>	3	2	1		
10	<u>C7</u>	9	10	8	<u>C3</u>	8	9	10		
11	<u>C7</u>	12	13	11	<u>C3</u>	11	12	13		
12	<u>C8</u>	5	4	6	<u>C4</u>	6	5	4		
13	<u>C8</u>	2	1	3	<u>C4</u>	3	2	1		
14	<u>C8</u>	9	10	8	<u>C4</u>	8	9	10		
15	<u>C8</u>	12	13	11	<u>C4</u>	11	12	13		

A.2.6 SDK-86 ADDRESSING

The I/O addressing of the IPI registers, by the SDK-86 is as follows (addresses in hexadecimal);

(1) read CSR register FF40,

(2) write to DBR register FF40,

(3) read from 'down transfer' registers,

register 1	FF42
register 2	FF44
.	
.	
register 15	FF5E.

A.2.7 PDP-11/03 ADDRESSING AND INTERRUPTS

The I/O addressing of the IPI registers by the PDP-11/03 depends upon the switch settings of switch S1. The address word (binary) is;

```
1111sssssssaaaaa
    7654321
    switches
```

The bits sssssss are set by the switches S1 as indicated to choose the required address range. The bits aaaaa are used to specify the IPI options as follows (low byte, high byte);

(1) write to CSR register 00000, 00001

(2) read CSR register 00000, 00001

(3) read DBR register 00010, 00011

(4) write to 'down transfer' registers,

register 1 00010, 00011

```
register 2    00100, 00101

    .

    .

register 15   11110, 11111
```

The interrupt vector word from the IPI depends on the settings on switch S2 as follows;

```
0000000sssssst00

       654321
      switches
```

The bit 't' specifies the type of interrupt;

(1) t = 0 when a 'DONE' interrupt (resulting from the SDK-86 writing to the DBR),

(2) t = 1 when an 'ERR' interrupt (resulting from an error condition as described in section 5.2.5).

Note that interrupt (1) has highest priority.

A.2.8 IPI REGISTER DESCRIPTIONS (CSR AND DBR)

The DBR (Data Buffer Register) bits are defined as follows (bit 0 is the least significant bit);

bits 0 to 11, data transferred from SDK-86 to PDP-11/03,

bit 12, reflects the state of bit 3 of the CSR.

The CSR (Command Status Register) bits are defined as follows;

bit 0, informs the SDK-86 that data is required (cleared when the SDK-86 writes data to the DBR),

bit 3, if set then bit 12 of the DBR will also be set,

bit 6, when set, an interrupt will be generated when the SDK-86 writes to the DBR, bit 7, set when the SDK-86 writes to the DBR,

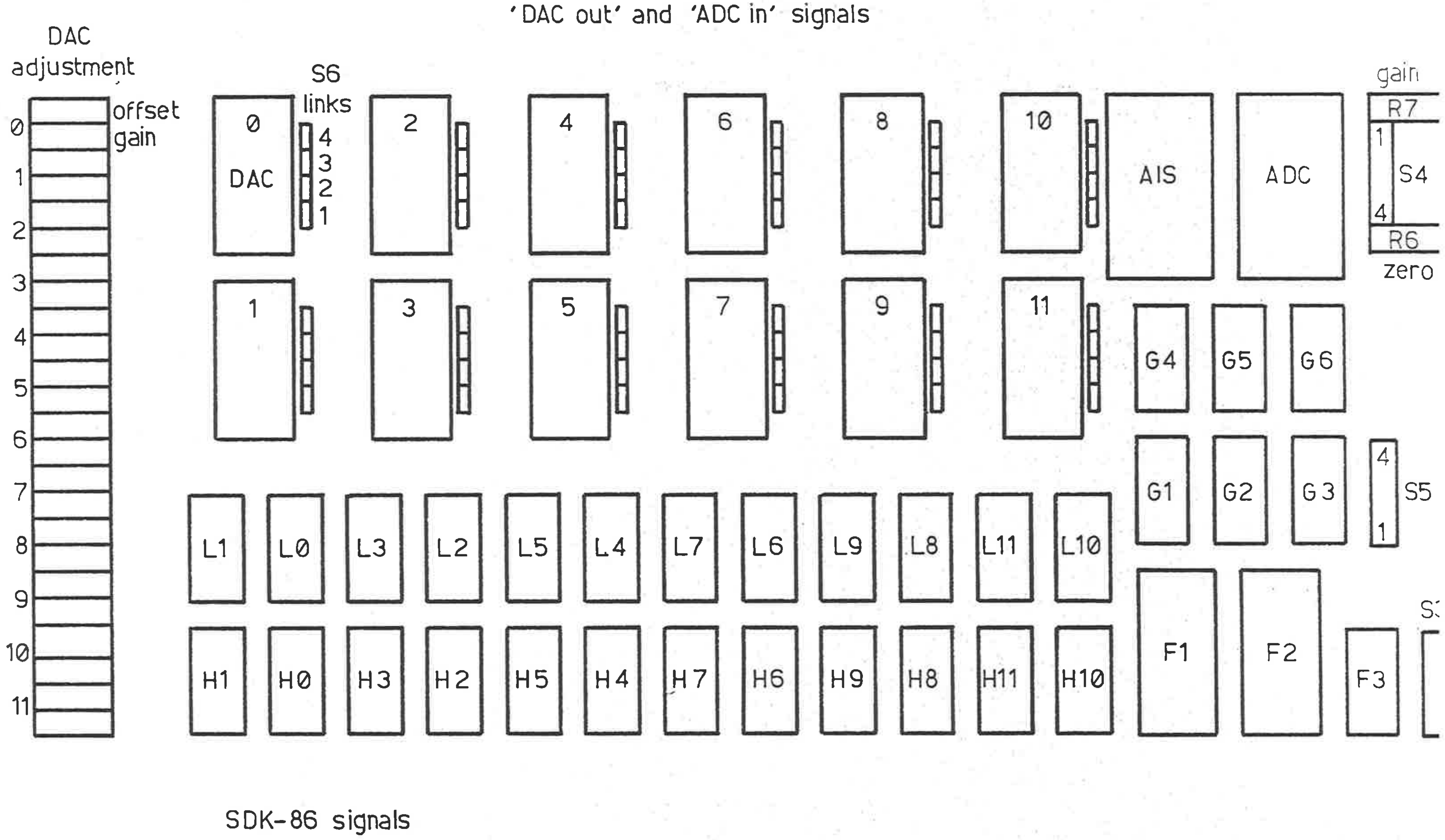
bits 8 to 11, channel address (bit 8 is the LSB),

bit 14, when set, an interrupt will be generated on an error condition (as defined in section 5.2.5),

bit 15, set on an error condition (section 5.2.5).

A.3 DAC/ADC BOARD

A.3.1 BOARD LAYOUT


'DAC out' and 'ADC in' signals
DAC adjustment
offset
gain
0
1
2
3
4
5
6
7
8
9
10
11
S6 links
4
3
2
1
0
DAC
2
4
6
8
10
1
3
5
7
9
11
AIS
ADC
gain
R7
S4
R6
zero
G4
G5
G6
G1
G2
G3
S5
L1
L0
L3
L2
L5
L4
L7
L6
L9
L8
L11
L10
H1
H0
H3
H2
H5
H4
H7
H6
H9
H8
H11
H10
F1
F2
F3
SDK-86 signals

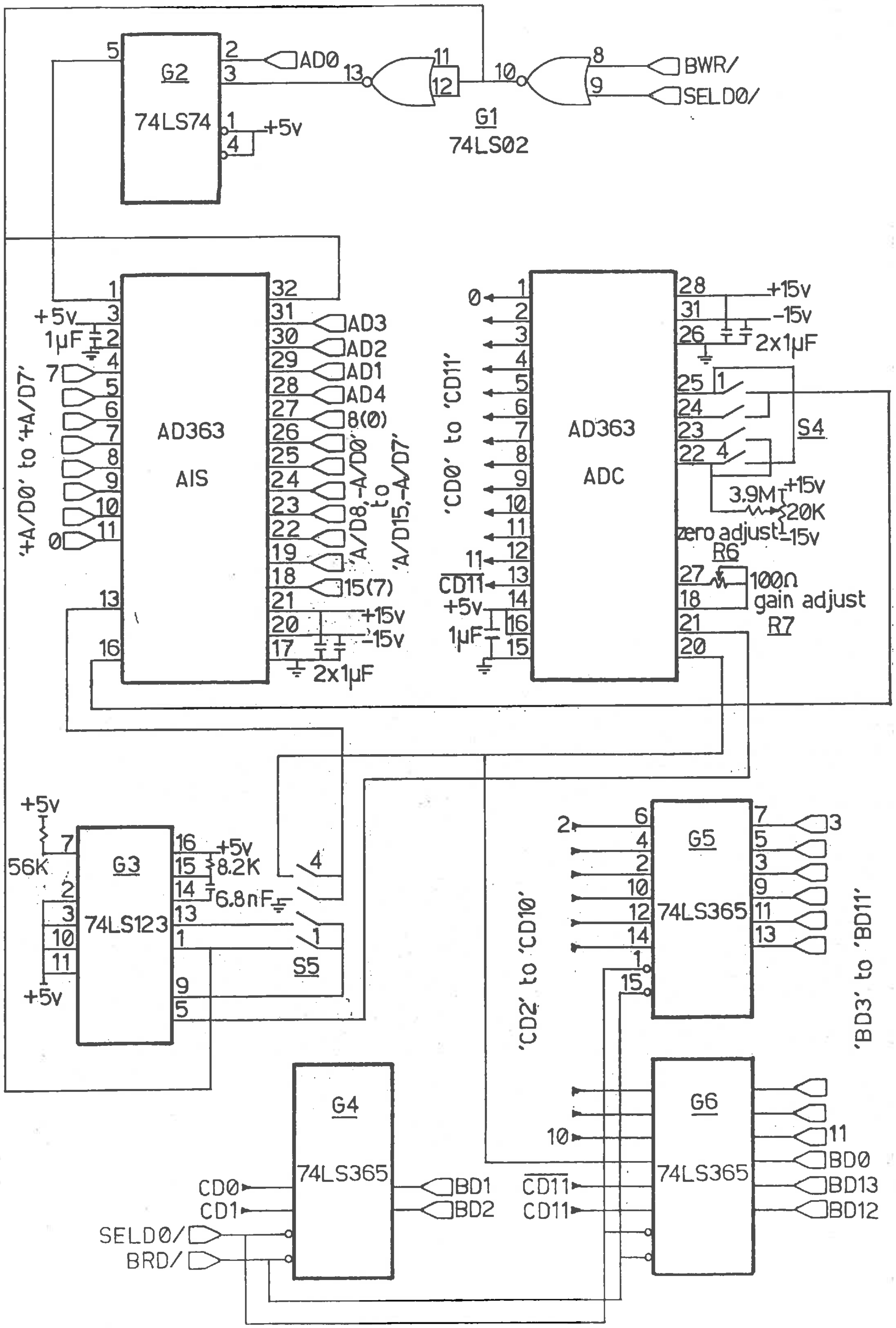
A.3.2 EDGE CONNECTIONS

The SDK-86 connections are the same as for the IPI board as they both share the same back plane.

The other signals come out of the top edge of the board as follows;

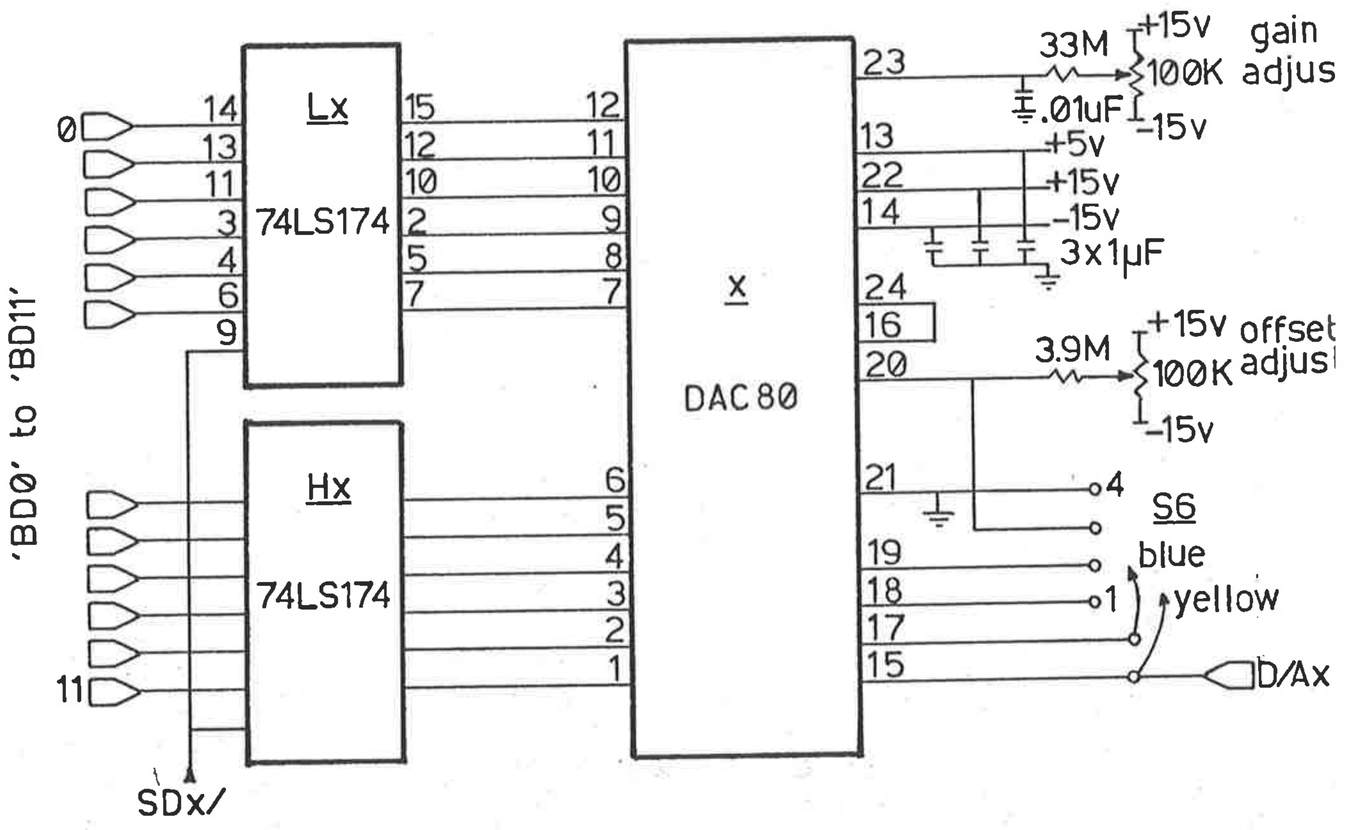
PIN NUMBER	SIGNAL	PIN NUMBER	SIGNAL
1	+A/D0	3	+A/D1
5	+A/D2	7	+A/D3
9	+A/D4	11	+A/D5
13	+A/D6	15	+A/D7
17	+A/D8,-A/D0	19	+A/D9,-A/D1
21	+A/D10,-A/D2	23	+A/D11,-A/D3
25	+A/D12,-1/D4	27	+A/D13,-A/D5
29	+A/D14,-A/D6	31	+A/D15,-A/D7
77	D/A11	79	D/A10
81	D/A9	83	D/A8
85	D/A7	87	D/A6
89	D/A5	91	D/A4
93	D/A3	95	D/A2
97	D/A1	99	D/A0
EVEN PINS			
2 to 32	EARTH		
78 to 100	EARTH		

A.3.3 ADC SECTION HARDWARE

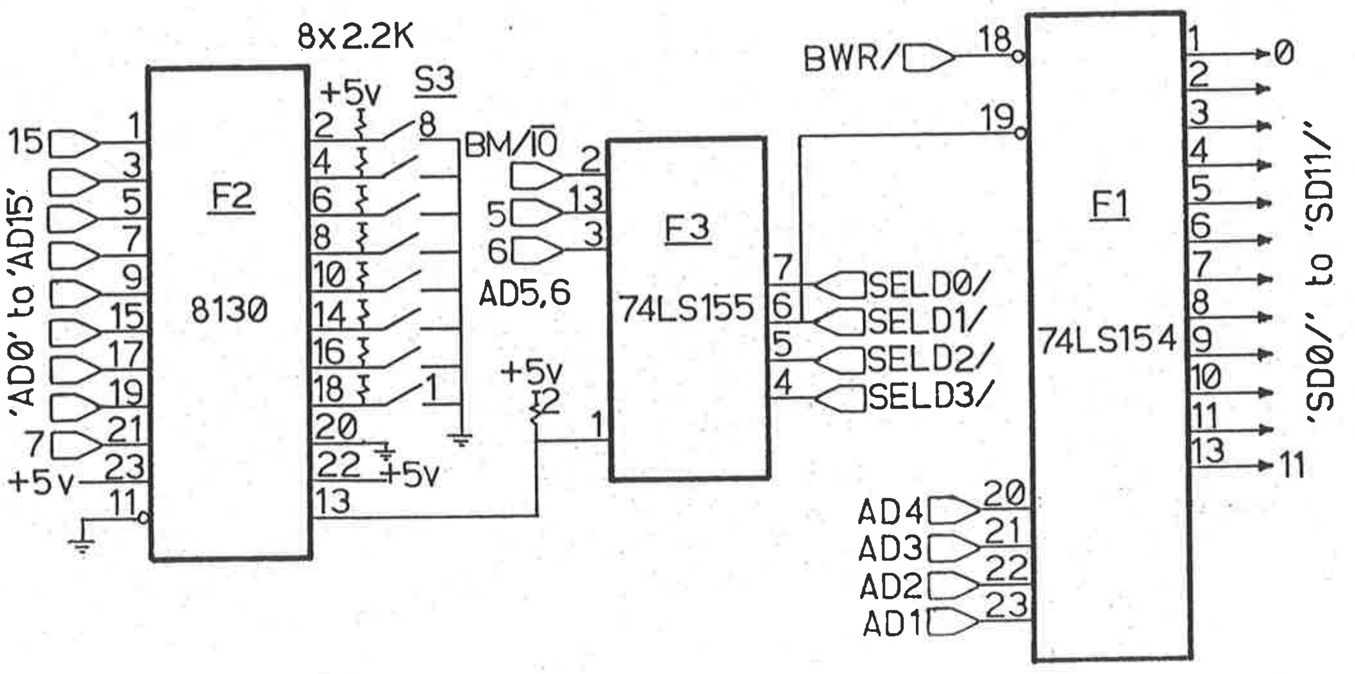

G2
74LS74
AD0
+5v
G1
74LS02
BWR/
SELD0/
AD363
AIS
'+A/D0' to '+A/D7'
+5v
1µF
AD3
AD2
AD1
AD4
8(0)
15(7)
'-A/D8,-A/D0' to '-A/D15,-A/D7'
+15v
-15v
2x1µF
AD363
ADC
'CD0' to 'CD11'
CD11
+5v
1µF
+15v
-15v
2x1µF
S4
3.9M
20K
zero adjust
R6
100Ω
gain adjust
R7
G3
74LS123
+5v
56K
8.2K
6.8nF
S5
G5
74LS365
'CD2' to 'CD10'
'BD3' to 'BD11'
G4
74LS365
CD0
CD1
SELD0/
BRD/
BD1
BD2
G6
74LS365
CD11
CD11
BD0
BD13
BD12

A.3.4 DAC AND DECODE SECTION HARDWARE

x=0 to 11



Lx
74LS174
Hx
74LS174
x
DAC80
'BD0' to 'BD11'
0
11
SDx/
33M
100K
+15v
-15v
gain adjus
.01uF
+5v
+15v
-15v
3x1μF
3.9M
100K
offset adjus
S6
blue
yellow
D/Ax



8x2.2K
+5v
S3
F2
8130
'AD0' to 'AD15'
15
7
BM/IO
AD5,6
F3
74LS155
SELD0/
SELD1/
SELD2/
SELD3/
BWR/
F1
74LS154
AD4
AD3
AD2
AD1
'SD0/' to 'SD11/'
0
11

A.3.5 ADDRESSING

This board contains the major decoding hardware (for signals SELD0 to SELD3) as well as the local decoding hardware. The address word is as follows;

```
ssssssss0aabbbbb

87654321
switches
```

The switch values are set by switch S3 (s = 0 when off, s = 1 when on). Bits aa are used to select the device as follows;

(1) 00 selects the ADC (SELD0),

(2) 01 selects the DAC (SELD1),

(3) 10 selects the IPI (SELD2),

(4) 11 not used.

The bits bbbbb are used for further decoding as described in sections A.3.6 and A.3.7.

A.3.6 A/D CONVERSION

An A/D conversion consists of two steps. Firstly, the SDK-86 must initiate the conversion by addressing the ADC as specified in A.3.5. The bits bbbbb are used to specify the channel number (high order 4 bits) and the conversion mode (the LSB is set to 0 for single ended mode and 1 for differential mode).

Secondly, the SDK-86 reads from the ADC (bits bbbbb can be chosen arbitrarily). Bit 0 (LSB) is 0 if the conversion is complete. When complete, bits 12 to 1 contain the converted value.

A.3.7 D/A CONVERSION

To perform a D/A conversion, the SDK-86 simply writes the data to be converted to the address as specified in A.3.5 above. The four high order bits of bbbbb are used to select the required convertor (15 channels).

A.3.8 ADC RANGE SELECTION

The ADC range is selected by switches S4 as follows;

range	switches			
	4	3	2	1
0 to 5v	on	off	on	off
0 to 10v	off	off	on	off
-2.5 to 2.5v	on	on	on	off
-5 to 5v	off	on	on	off
-10 to 10v	off	on	off	on

A.3.9 DAC RANGE SELECTION

The range selection for the DACs is made using two wire straps taken to 4 sockets (labelled S6 on the circuit diagram) as follows;

range	wire straps		
	yellow	blue	
-10 to 10v	2	3	
-5 to 5v	1	3	
-2.5 to 2.5v	1	3	(also bridge 2 and 3)
0 to 10v	1	4	
0 to 5v	1	4	(also bridge 2 and 3)

A.3.10 ADC SAMPLING MODE

Two sampling modes are selectable;

(1) continual sampling for slow signals,

(2) sample and hold for fast signals.

The selection is made using the switches S5 as follows;

	switches			
	4	3	2	1
continual sampling	off	on	off	on
sample and hold	on	off	on	off

A.4 DATA SHEETS (AD363 and DAC80)

Complete 16-Channel 12-Bit Integrated Circuit Data Acquisition System

ANALOG DEVICES

AD363

FEATURES

Versatility
- Complete System in Reliable IC Form
- Small Size: Two 32 Pin Metal DIP's
- 16 Single-Ended or 8 Differential Channels with Switchable Mode Control
- Military/Aerospace Temperature Range: −55°C to +125°C (AD363S) MIL-STD-883B Processing Available
- Versatile Input/Output/Control Format
- Short-Cycle Capability

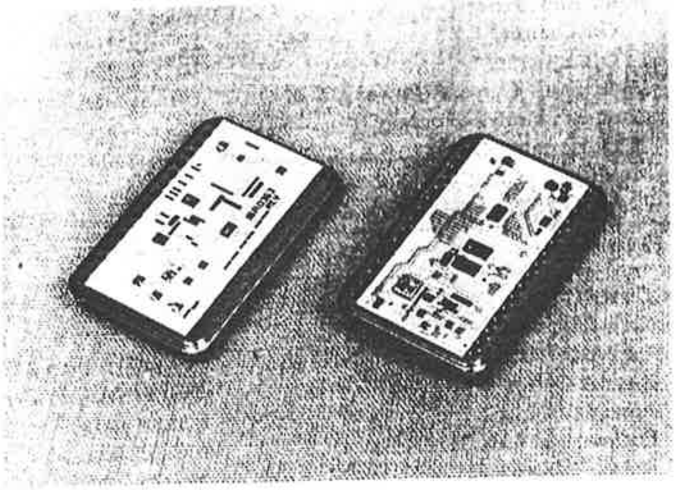
Performance
- True 12 Bit Operation: Nonlinearity ≤±0.012%
- Guaranteed No Missing Codes Over Temperature Range
- High Throughput Rate: 30kHz
- Low Power: 1.7W
- Hermetically-Sealed, Electrostatically-Shielded Metal DIP's

Value
- Complete: No Additional Parts Required
- Reliable: Hybrid IC Construction, Hermetically Sealed by Welding. All Inputs Fully Protected.
- Precision +10.0 ±0.005 Volt Reference for External Application
- Fast Precision Buffer Amplifier for External Application
- Low Cost

PRODUCT DESCRIPTION

The AD363 is a complete 16 channel, 12 bit data acquisition system in integrated circuit form. By applying large-scale linear and digital integrated circuitry, thick and thin film hybrid technology and active laser trimming, the AD363 equals or exceeds the performance and versatility of previous modular designs.

The AD363 consists of two separate functional blocks, each hermetically-sealed in an electrostatically-shielded 32 pin metal dual-in-line package. The analog input section contains two eight-channel multiplexers, a differential amplifier, a sample-and-hold, a channel address register and control logic. The multiplexers may be connected to the differential amplifier in either an 8-channel differential or 16-channel single-ended configuration. A unique feature of the AD363 is an internal user-controllable analog switch that connects the multiplexers in either a single-ended or differential mode. This allows a single device to perform in either mode without hard-wire programming and permits a mixture of single-ended and differential sources to be interfaced to an AD363 by dynamically switching the input mode control.



The Analog-to-Digital Converter Section contains a complete 12-bit successive approximation analog-to-digital converter, including internal clock, precision 10 volt reference, comparator, buffer amplifier and a proprietary-design 12 bit D/A converter. Active laser trimming of the reference and D/A converter results in maximum linearity errors of ±0.012% while performing a 12 bit conversion in 25 microseconds.

Analog input voltage ranges of ±2.5, ±5.0, ±10, 0 to +5 and 0 to +10 volts are user-selectable. Adding flexibility and value are the precision 10 volt reference (active-trimmed to a tolerance of ±5mV) and the internal buffer amplifier, both of which may be used for external applications. All digital signals are TTL/DTL compatible and output data is positive-true in parallel and serial form.

System throughput rate is as high as 30kHz at full rated accuracy. The AD363K is specified for operation over a 0 to +70°C temperature range while the AD363S operates to specification from −55°C to +125°C. Processing to MIL-STD-883B is available for the AD363S. Both device grades are guaranteed to have no missing codes over their specified temperature ranges.

SPECIFICATIONS (typical @ +25°C, ±15V and +5V with 2000pF hold capacitor as provided unless otherwise not

MODEL	AD363K	AD363S
ANALOG INPUTS		
Number of Inputs	16 Single-Ended or 8 Differential (Electronically Selectable)	*
Input Voltage Ranges		
Bipolar	±2.5V, ±5.0V, ±10.0V	*
Unipolar	0 to +5V, 0 to +10V	*
Input (Bias) Current, Per Channel	±50nA max	*
Input Impedance		
On Channel	$10^{10}\Omega$, 100pF	*
Off Channel	$10^{10}\Omega$, 10pF	*
Input Fault Current (Power Off or On)	20mA, max, Internally Limited	*
Common Mode Rejection		
Differential Mode	70dB min (80dB typ) @ 1kHz, 20V p-p	*
Mux Crosstalk (Interchannel, Any Off Channel to Any On Channel)	–80dB max (–90dB typ) @ 1kHz, 20V p-p	*
RESOLUTION	12 BITS	*
ACCURACY		
Gain Error[1]	±0.05% FSR (Adj. to Zero)	*
Unipolar Offset Error	±10mV (Adj to Zero)	*
Bipolar Offset Error	±20mV (Adj to Zero)	*
Linearity Error	±½LSB max	*
Differential Linearity Error	±1LSB max (±½LSB typ)	*
Relative Accuracy	±0.025% FSR	*
Noise Error	1mV p-p, 0 to 1MHz	*
TEMPERATURE COEFFICIENTS		
Gain	±30ppm/°C max (±10ppm/°C typ)	±25ppm/°C max (±15ppm/°C typ)
Offset, ±10V Range	±10ppm/°C max (±5ppm/°C typ)	±8ppm/°C max (±5ppm/°C typ)
Differential Linearity	No Missing Codes Over Temperature Range	*
SIGNAL DYNAMICS		
Conversion Time[2]	25μs max (22μs typ)	*
Throughput Rate, Full Rated Accuracy	25kHz min (30kHz typ)	*
Sample and Hold		
Aperture Delay	100ns max (50ns typ)	*
Aperture Uncertainty	500ps max (100ps typ)	*
Acquisition Time		
To ±0.01% of Final Value for Full Scale Step	18μs max (10μs, typ)	*
Feedthrough	–70dB max (–80dB typ) @ 1kHz	*
Droop Rate	2mV/ms max (1mV/ms typ)	*
DIGITAL INPUT SIGNALS[4]		
Convert Command (to ADC Section, Pin 21)	Positive Pulse, 200ns min Width. Leading Edge ("0" to "1") Resets Register, Trailing Edge ("1" to "0") Starts Conversion.	*
	1TTL Load	*
Input Channel Select (To Analog Input Section, Pins 28-31)	4 Bit Binary, Channel Address.	*
	1LS TTL Load	*
Channel Select Latch (To Analog Input Section, Pin 32)	"1" Latch Transparent	*
	"0" Latched	*
	4LS TTL Loads	*

MODEL	AD363K	AD363S
DIGITAL INPUT SIGNALS, cont.		
Sample-Hold Command (To Analog Input Section Pin 13 Normally Connected To ADC "Status", Pin 20)	"0" Sample Mode	*
	"1" Hold Mode	*
	2LS TTL Loads	*
Short Cycle (To ADC Section Pin 14)	Connect to +5V for 12 Bits Resolution.	*
	Connect to Output Bit n + 1 For n Bits Resolution.	*
	1TTL Load	*
Single Ended/Differential Mode Select (To Analog Input Section, Pin 1)	"0": Single-Ended Mode	*
	"1": Differential Mode	*
	3TTL Loads	*
DIGITAL OUTPUT SIGNALS[4] (All Codes Positive True)		
Parallel Data		
Unipolar Code	Binary	*
Bipolar Code	Offset Binary/Two's Complement	*
Output Drive	2TTL Loads	*
Serial Data (NRZ Format)		
Unipolar Code	Binary	*
Bipolar Code	Offset Binary	*
Output Drive	2TTL Loads	*
Status ($\overline{\text{Status}}$)	Logic "1" ("0") During Conversion	*
Output Drive	2TTL Loads	*
Internal Clock		
Output Drive	2TTL Loads	*
Frequency	500kHz	*
INTERNAL REFERENCE VOLTAGE	+10.00V, ±5mV	*
Max External Current	±4mA	*
Voltage Temp. Coefficient	±20ppm/°C, max	±10ppm/°C, max
POWER REQUIREMENTS		
Supply Voltages/Currents	+15V, ±5% @ +45mA max (+38mA typ)	*
	-15V, ±5% @ -45mA max (-38mA typ)	*
	+5V, ±5% @ +136mA max (+113mA typ)	*
Total Power Dissipation	2 watts max (1.7 watts typ)	*
TEMPERATURE RANGE		
Specification	0 to +70°C	-55°C to +125°C
Storage	-55°C to +85°C[3]	-55°C to +150°C

NOTES:
[1] With 50Ω, 1% fixed resistor in place of Gain Adjust pot; see Figures 7 and 8.
[2] Conversion time of ADC Section.
[3] AD363K External Hold Capacitor is limited to +85°C; Analog Input Section and ADC Section may be stored at up to +150°C.
[4] One TTL Load is defined as I_{IL} = -1.6mA max @ V_{IL} = 0.4V, I_{IH} = 40μA max @ V_{IH} = 2.4V.
One LS TTL Load is defined as I_{IL} = -0.36mA max @ V_{IL} = 0.4V, I_{IH} = 20μA max @ V_{IH} = 2.7V.

Specifications subject to change without notice.

ABSOLUTE MAXIMUM RATINGS (ALL MODELS)

+V, Digital Supply	+5.5V
+V, Analog Supply	+16V
−V, Digital Supply	-16V
V_{IN}, Signal	±V, Analog Supply
V_{IN}, Digital	0 to +V, Digital Supply
A_{GND} to D_{GND}	±1V

PIN FUNCTION DESCRIPTION

ANALOG INPUT SECTION

Pin Number	Function
1	Single-End/Differential Mode Select
	"0": Single-Ended Mode
	"1": Differential Mode
2	Digital Ground
3	Positive Digital Power Supply, +5V
4	"High" Analog Input, Channel 7
5	"High" Analog Input, Channel 6
6	"High" Analog Input, Channel 5
7	"High" Analog Input, Channel 4
8	"High" Analog Input, Channel 3
9	"High" Analog Input, Channel 2
10	"High" Analog Input, Channel 1
11	"High" Analog Input, Channel 0
12	Hold Capacitor (Provided, See Figure 1)
13	Sample-Hold Command
	"0": Sample Mode
	"1": Hold Mode
	Normally Connected to ADC Pin 20
14	Offset Adjust (See Figure 6)
15	Offset Adjust (See Figure 6)
16	Analog Output
	Normally Connected to ADC
	"Analog In" (See Figure 1)
17	Analog Ground
18	"High" ("Low") Analog Input, Channel 15 (7)
19	"High" ("Low") Analog Input, Channel 14 (6)
20	Negative Analog Power Supply, −15V
21	Positive Analog Power Supply, +15V
22	"High" ("Low") Analog Input, Channel 13 (5)
23	"High" ("Low") Analog Input, Channel 12 (4)
24	"High" ("Low") Analog Input, Channel 11 (3)
25	"High" ("Low") Analog Input, Channel 10 (2)
26	"High" ("Low") Analog Input, Channel 9 (1)
27	"High" ("Low") Analog Input, Channel 8 (0)
28	Input Channel Select, Address Bit AE
29	Input Channel Select, Address Bit A0
30	Input Channel Select, Address Bit A1
31	Input Channel Select, Address Bit A2
32	Input Channel Select Latch
	"0": Latched
	"1": Latch "Transparent"

ANALOG TO DIGITAL CONVERTER SECTION

Pin Number	Function
1	Data Bit 12 (Least Significant Bit) Out
2	Data Bit 11 Out
3	Data Bit 10 Out
4	Data Bit 9 Out
5	Data Bit 8 Out
6	Data Bit 7 Out
7	Data Bit 6 Out
8	Data Bit 5 Out
9	Data Bit 4 Out
10	Data Bit 3 Out
11	Data Bit 2 Out
12	Data Bit 1 (Most Significant Bit) Out
13	$\overline{\text{Data Bit 1}}$ ($\overline{\text{MSB}}$) Out
14	Short Cycle Control
	Connect to +5V for 12 Bits
	Connect to Bit (n+1) Out for n Bits
15	Digital Ground
16	Positive Digital Power Supply, +5V
17	$\overline{\text{Status}}$ Out
	"0": Conversion in Progress
	(Parallel Data Not Valid)
	"1": Conversion Complete
	(Parallel Data Valid)
18	+10Volt Reference Out (See Figures 3, 7, 8, 11)
19	Clock Out (Runs During Conversion)
20	Status Out
	"0": Conversion Complete
	(Parallel Data Valid)
	"1": Conversion in Progress
	(Parallel Data Not Valid)
21	Convert Start In
	Reset Logic : ___↑‾‾‾
	Start Convert : ‾‾‾↓___
22	Comparator In (See Figures 3, 7, 8)
23	Bipolar Offset
	Open for Unipolar Inputs
	Connect to ADC Pin 22 for
	Bipolar Inputs
	(See Figure 8)
24	10V Span R In (See Figure 7)
25	20V Span R In (See Figure 8)
26	Analog Ground
27	Gain Adjust (See Figures 7 and 8)
28	Positive Analog Power Supply, +15V
29	Buffer Out (For External Use)
30	Buffer In (For External Use)
31	Negative Analog Power Supply, −15V
32	Serial Data Out
	Each Bit Valid On Trailing (‾‾↓__)
	Edge Clock Out, ADC Pin 19

AD363 DESIGN

Concept

The AD363 consists of two separate functional blocks as shown in Figure 1; each is packaged in a hermetically-sealed 32 pin metal DIP.

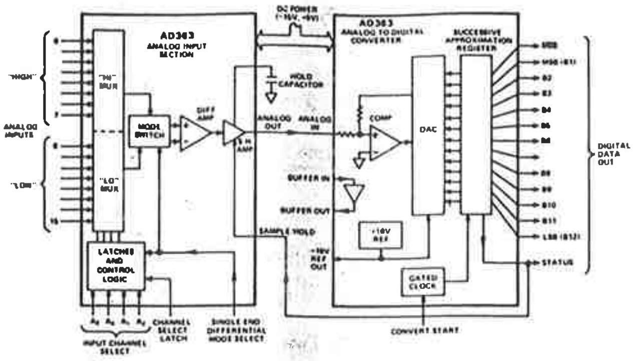


Figure 1. AD363 Functional Block Diagram

The Analog Input Section contains multiplexers, a differential amplifier, a sample-and-hold, a channel address register and control logic. Analog-to-digital conversion is provided by a 12 bit, 25 microsecond "ADC" which is also available separately as the AD572.

By dividing the data acquisition task into two sections, several important advantages are realized. Performance of each design is optimized for its specific function. Production yields are increased thus decreasing costs. Furthermore, the standard configuration 32 pin packages plug into standard sockets and are easier to handle than larger packages with higher pin counts.

Analog Input Section Design

Figure 2 is a block diagram of the AD363 Analog Input Section (AIS).

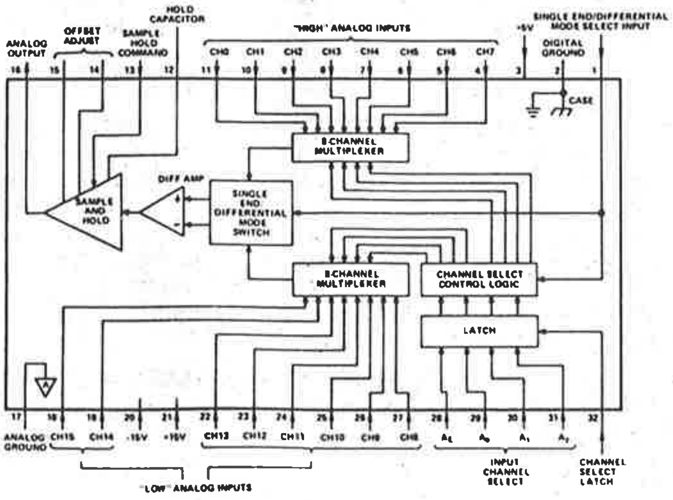


Figure 2. AD363 Analog Input Section Functional Block Diagram and Pinout

The AIS consists of two 8-channel multiplexers, a differential amplifier, a sample-and-hold, channel address latches and control logic. The multiplexers can be connected to the differential amplifier in either an 8-channel differential or 16-channel single-ended configuration. A unique feature of the AD363 is an internal analog switch controlled by a digital input that performs switching between single-ended and differential modes. This feature allows a single product to perform in either mode without external hard-wire interconnections. Of more significance is the ability to serve a mixture of both single-ended and differential sources with a single AD363 by dynamically switching the input mode control.

Multiplexer channel address inputs are interfaced through a level-triggered ("transparent") input register. With a Logic "1" at the Channel Select Latch input, the address signals feed through the register to directly select the appropriate input channel. This address information can be held in the register by placing a Logic "0" on the Channel Select Latch input. Internal logic monitors the status of the Single-Ended/Differential Mode input and addresses the multiplexers accordingly.

A differential amplifier buffers the multiplexer outputs while providing high input impedance in both differential and single-ended modes. Amplifier gain and common mode rejection are actively laser-trimmed.

The sample-and-hold is a high speed monolithic device that can also function as a gated operational amplifier. Its uncommitted differential inputs allow it to serve a second role as the output subtractor in the differential amplifier. This eliminates one amplifier and decreases drift, settling time and power consumption. A Logic "1" on the Sample-and-Hold Command input will cause the sample-and-hold to "freeze" the analog signal while the ADC performs the conversion. Normally the Sample-and-Hold Command is connected to the ADC Status output which is at Logic "1" during conversion and Logic "0" between conversions. For slowly-changing inputs, throughput speed may be increased by grounding the Sample-and-Hold Command input instead of connecting it to the ADC status.

A Polystyrene hold capacitor is provided with each commercial temperature range system (AD363K) while a Teflon capacitor is provided with units intended for operation at temperatures up to 125°C (AD363S). Use of an external capacitor allows the user to make his own speed/accuracy tradeoff; a smaller capacitor will allow faster sample-and-hold response but will decrease accuracy while a larger capacitor will increase accuracy at slower conversion rates.

The Analog Input Section is constructed on a substrate that includes thick-film resistors for non-critical applications such as input protection and biasing. A separately-mounted laser-trimmed thin-film resistor network is used to establish accurate gain and high common-mode rejection. The metal package affords electromagnetic and electrostatic shielding and is hermetically welded at low temperatures. Welding eliminates the possibility of contamination from solder particles or flux while low temperature sealing maintains the accuracy of the laser-trimmed thin-film resistors.

Analog-to-Digital Converter Design

Figure 3 is a block diagram of the Analog-to-Digital Converter Section (ADC) of the AD363.

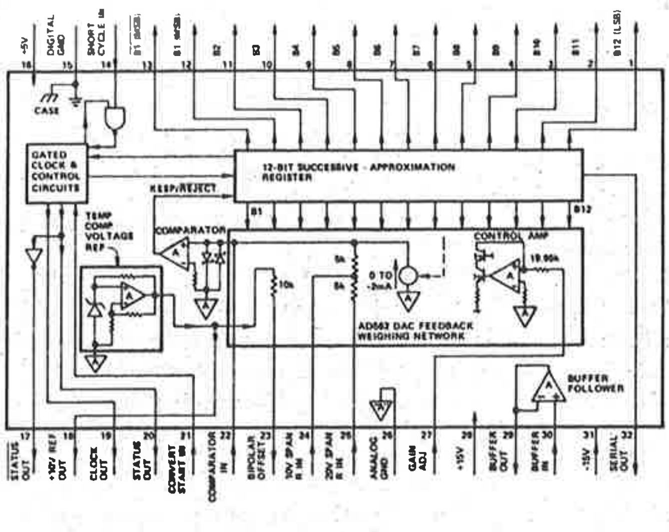


Figure 3. AD363 ADC Section (AD572) Functional Diagram and Pinout

Available separately as the AD572, the ADC is a 12 bit, 25 microsecond device that includes an internal clock, reference, comparator and buffer amplifier.

The +10V reference is derived from a low T.C. zener reference diode which has its zener voltage amplified and buffered by an op amp. The reference voltage is calibrated to +10V, ±5mV by active laser-trimming of the thin-film resistors which determine the closed-loop gain of the op amp. 4mA of current is available for external use. The reference circuit is constructed on its own thin-film substrate which is, in turn, mounted on the thick-film ADC main substrate.

The DAC feedback weighting network is comprised of a proprietary 12 bit analog current switch chip and silicon-chromium thin-film ladder network. (Packaged separately, this DAC is available as the AD562.) This ladder network is active laser-trimmed to calibrate all bit ratio scale factors to a precision of 0.005% of FSR (full-scale range) to guarantee no missing codes over the operating temperature range. The design of the ADC includes scaling resistors that provide user-selectable analog input signal ranges of ±2.5, ±5, ±10, 0 to +5, or 0 to +10 volts.

Other useful features include true binary output for unipolar inputs, offset binary and two's complement output for bipolar inputs, serial output, short-cycle capability for lower resolution, higher speed measurements, and an available high input impedance buffer amplifier which may be used elsewhere in the system.

As in the Analog Input Section, the ADC main substrate includes thick-film resistors in non-critical areas. Thin-film substrates are separately mounted to assure accurate and stable reference and DAC performance. Packaging considerations are the same as for the AIS.

THEORY OF OPERATION

System Timing

Figure 4 is a timing diagram for the AD363 connected as shown shown in Figure 1 and operating at maximum conversion rate.

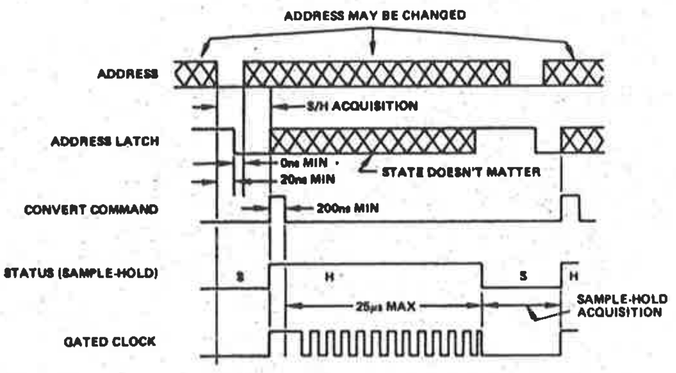


Figure 4. AD363 Timing Diagram

The normal sequence of events is as follows:

1. The appropriate Channel Select Address is latched into the address register. Time is allowed for the multiplexers to settle.
2. A Convert Start command is issued to the ADC which indicates that it is "busy" by placing a Logic "1" on its Status line.
3. The ADC Status controls the sample-and-hold. When the ADC is "busy" the sample-and-hold is in the hold mode.
4. The ADC goes into its 25 microsecond conversion routine. Since the sample-and-hold is holding the proper analog value, the address may be updated during conversion. Thus multiplexer settling time can coincide with conversion and need not effect throughput rate.
5. The ADC indicates completion of its conversion by returning Status to Logic "0". The sample-and-hold returns to the sample mode.
6. If the input signal has changed full-scale (different channels may have widely-varying data) the sample-and-hold will typically require 10 microseconds to "acquire" the next input to sufficient accuracy for 12 bit conversion.

After allowing a suitable interval for the sample-and-hold to stabilize at its new value, another Convert Start command may be issued to the ADC.

ADC Operation

On receipt of a Convert Start command, the analog-to-digital converter converts the voltage at its analog input into an equivalent 12-bit binary number. This conversion is accomplished as follows:

The 12-bit successive-approximation register (SAR) has its 12-bit outputs connected both to the respective device bit output pins and to the corresponding bit inputs of the feedback DAC.

The analog input is successively compared to the feedback DAC output, one bit at a time (MSB first, LSB last). The decision to keep or reject eaoh bit is then made at the completion of each bit comparison period, depending on the state of the comparator at that time.

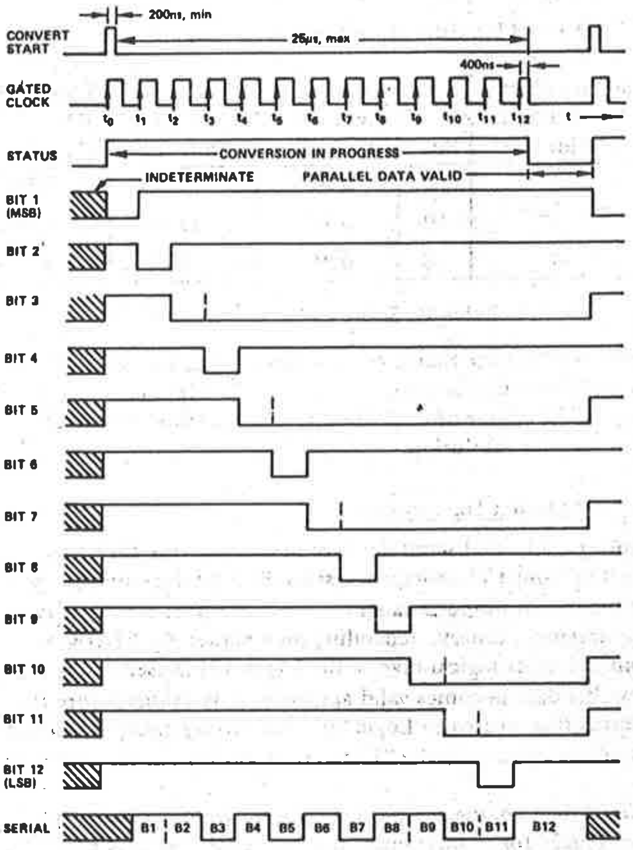


Figure 5. ADC Timing Diagram (Binary Code 110101011001)

The timing diagram is shown in Figure 5. Receipt of a Convert Start signal sets the Status flag, indicating conversion in progress. This, in turn, removes the inhibit applied to the gated clock, permitting it to run through 13 cycles. All SAR parallel bit and Status flip-flops are initialized on the leading edge, and the gated clock inhibit signal removed on the trailing edge of the Convert Start signal. At time t0, B1 is reset and B2-B12 are set unconditionally. At t1 the Bit 1 decision is made (keep) and Bit 2 is unconditionally reset. At t2, the Bit 2 decision is made (keep) and Bit 3 is reset unconditionally. This sequence continues until the Bit 12 (LSB) decision (keep) is made at t12. After 400ns delay period, the Status flag is reset, indicating that the conversion is complete and that the parallel output data is valid. Resetting the Status flag restores the gated clock inhibit signal, forcing the clock output to the Logic "0" state.

Corresponding serial and parallel data bits become valid on the same positive-going clock edge. Serial data does not change and is guaranteed valid on negative-going clock edges, however; serial data can be transferred quite simply by clocking it into a receiving shift register on these edges.

Incorporation of this 400ns delay period guarantees that the parallel (and serial) data are valid at the Logic "1" to "0" transition of the Status flag, permitting parallel data transfer to be initiated by the trailing edge of the Status signal.

The versatility and completeness of the AD363 concept results in a large number of user-selectable configurations. This allows optimization of most systems applications.

Single-Ended/Differential Mode Control

The 363 features an internal analog switch that configures the Analog Input Section in either a 16-channel single-ended or 8-channel differential mode. This switch is controlled by a TTL logic input applied to pin 1 of the Analog Input Section:

"0":	Single-Ended (16 channels)
"1":	Differential (8 channels)

When in the differential mode, a differential source may be applied between corresponding "High" and "Low" analog input channels.

It is possible to mix SE and DIFF inputs by using the mode control to command the appropriate mode. Figure 11 illustrates an example of a "mixed" application. In this case, four microseconds must be allowed for the output of the Analog Input Section to settle to within ±0.01% of its final value, but if the mode is switched concurrent with changing the channel address, no significant additional delay is introduced. The effect of this delay may be eliminated by changing modes while a conversion is in progress (with the sample-and-hold in the "hold mode"). When SE and DIFF signals are being processed concurrently, the DIFF signals must be applied between corresponding "High" and "Low" analog input channels. Another application of this feature is the capability of measuring 16 sources individually and/or measuring differences between pairs of those sources.

Input Channel Addressing

Table 1 is the truth table for input channel addressing in both the single-ended and differential modes. The 16 single-ended channels may be addressed by applying the corresponding digital number to the four Input Channel Select address bits, AE, A0, A1, A2 (Analog Input Section, pins 28—31). In the differential mode, the eight channels are addressed by applying the appropriate digital code to A0, A1 and A2; AE must be enabled with a Logic "1". Internal logic monitors the status of the SE/DIFF Mode input and addresses the multiplexes singly or in pairs as required.

ADDRESS				ON CHANNEL (Pin Number)		
					Differential	
AE	A2	A1	A0	Single Ended	"Hi"	"Lo"
0	0	0	0	0 (11)	None	
0	0	0	1	1 (10)	None	
0	0	1	0	2 (9)	None	
0	0	1	1	3 (8)	None	
0	1	0	0	4 (7)	None	
0	1	0	1	5 (6)	None	
0	1	1	0	6 (5)	None	
0	1	1	1	7 (4)	None	
1	0	0	0	8 (27)	0 (11)	0 (27)
1	0	0	1	9 (26)	1 (10)	1 (26)
1	0	1	0	10 (25)	2 (9)	2 (25)
1	0	1	1	11 (24)	3 (8)	3 (24)
1	1	0	0	12 (23)	4 (7)	5 (23)
1	1	0	1	13 (22)	5 (6)	5 (22)
1	1	1	0	14 (19)	6 (5)	6 (19)
1	1	1	1	15 (18)	7 (4)	7 (18)

Table 1. Input Channel Addressing Truth Table

When the channel address is changed, six microseconds must be allowed for the Analog Input Section to settle to within ±0.01% of its final output (including settling times of all elements in the signal path). The effect of this delay may be eliminated by performing the address change while a conversion is in progress (with the sample-and-hold in the "hold" mode).

Input Channel Address Latch

The AD363 is equipped with a latch for the Input Channel Select address bits. If the Latch Control pin (pin 32 of the Analog Input Section) is at Logic "1", input channel select address information is passed through to the multiplexers. A Logic "0" "freezes" the input channel address present at the inputs at the time of the "1" to "0" transition.

This feature is useful when input channel address information is provided from an address, data or control bus that may be required to service many devices. The ability to latch an address is helpful whenever the user has no control of when address information may change.

Sample-and-Hold Mode Control

The Sample-and-Hold Mode Control input (Analog Input Section, pin 13) is normally connected to the Status output (pin 20) from the ADC section. When a conversion is initiated by applying a Convert Start command to the ADC (pin 21), Status goes to Logic "1", putting the sample-and-hold into the "hold" mode. This "freezes" the information to be digitized for the period of conversion. When the conversion is complete, Status returns to Logic "0" and the sample-and-hold returns to the sample mode. Eighteen microseconds must be allowed for the sample-and-hold to acquire ("catch up" to) the analog input to within ±0.01% of the final value before a new Convert Start command is issued.

The purpose of a sample-and-hold is to "stop" fast changing input signals long enough to be converted. In this application, it also allows the user to change channels and/or SE/ DIFF mode while a conversion is in progress thus eliminating the effects of multiplexer, analog switch and differential amplifier settling times. If maximum throughput rate is required for slowly changing signals, the Sample-and-Hold Mode Control may be wired to ground (Logic "0") rather than to ADC Status thus leaving the sample-and-hold in a continuous sample mode.

Hold Capacitor

A 2000pF capacitor is provided with each AD363. One side of this capacitor is wired to the Analog Input Section pin 12, the other to analog ground as close to pin 17 as possible. The capacitor provided with the AD363K is Polystyrene while the wider operating temperature range of the AD363S demands a Teflon capacitor (supplied).

Larger capacitors may be substituted to minimize noise, but acquisition time of the sample-and-hold will be extended. If less than 12 bits of accuracy is required, a smaller capacitor may be used. This will shorten the S/H acquisition time. In all cases, the proper capacitor dielectric must be used; i.e., Polystyrene (AD363K only) or Teflon (AD363K or S). Other types of capacitors may have higher dielectric absorption (memory) and will cause errors. CAUTION: Polystyrene capacitors will be destroyed if subjected to temperatures above +85°C. No capacitor is required if the sample-and-hold is not used.

Short Cycle Control

A Short Cycle Control (ADC Section, pin 14) permits the timing cycle shown in Figure 5 to be terminated after any number of desired bits has been converted, permitting somewhat shorter conversion times in applications not requiring full 12-bit resolution. When 12-bit resolution is required, pin 14 is connected to +5V (ADC Section, pin 10). When 10-bit resolution is desired, pin 14 is connected to Bit 11 output pin 2. The conversion cycle then terminates, and the Status flag resets after the Bit 10 decision (t10 + 400ns in timing diagram of Figure 2). Short Cycle pin connections and associated maximum 12, 10 and 8-bit conversion times are summarized in Table 2.

Connect Short Cycle Pin 14 to Pin:	Bits	Resolution (% FSR)	Maximum Conversion Time (μs)	Status Flag Reset at: (Figure 5)
16	12	0.024	25	t_{12} +400ns
2	10	0.10	21	t_{10} +400ns
4	8	0.39	17	t_8 +400ns

Table 2. Short Cycle Connections

One should note that the calibration voltages listed in Table 4 are for 12-bit resolution only, and are not those corresponding to the center of each discrete quantization interval at reduced bit resolution.

Digital Output Data Format

Both parallel and serial data are in positive-true form and outputted from TTL storage registers. Parallel data output coding is binary for unipolar ranges and either offset binary or two's complement binary, depending on whether Bit 1 (ADC Section pin 12) or its logical inverse $\overline{\text{Bit}}$ 1 (pin 13) is used as the MSB. Parallel data becomes valid approximately 200ns before the Status flag returns to Logic "0", permitting parallel data transfer to be clocked on the "1" to "0" transition of the Status flag.

Serial data coding is binary for unipolar input ranges and offset binary for bipolar input ranges. Serial output is by bit (MSB first, LSB last) in NRZ (non-return-to-zero) format. Serial and parallel data outputs change state on positive-going clock edges. Serial data is guaranteed valid on all negative-going clock edges, permitting serial data to be clocked directly into a receiving register on these edges. There are 13 negative-going clock edges in the complete 12-bit conversion cycle, as shown in Figure 5. The first edge shifts an invalid bit into the register, which is shifted out on the 13th negative-going clock edge. All serial data bits will have been correctly transferred at the completion of the conversion period.

Analog Input Voltage Range Format

The AD363 may be configured for any of 3 bipolar or 2 unipolar input voltage ranges as shown in Table 3.

Range	Connect Analog Input To ADC Pin:	Connect ADC Span Pin:	Connect Bipolar ADC Pin 23 To:
0 to +5V	24	25 to 22	—
0 to +10V	24	—	
-2.5V to +2.5V	24	25 to 22	22
-5V to +5V	24	—	
-10V to +10V	25	—	

Table 3. Analog Input Voltage Range Pin Connections

Analog Input - Volts (Center of Quantization Interval)			Input Normalized to FSR		Digital Output Code (Binary for Unipolar Ranges; Offset Binary for Bipolar Ranges)
0 to +10V Range	-5V to +5V Range	-10V to +10V Range	Unipolar Ranges	Bipolar Ranges	B1 (MSB) … B12 (LSB)
+9.9976	+4.9976	+9.9951	+FSR-1 LSB	+½FSR-1 LSB	1 1 1 1 1 1 1 1 1 1 1 1
+9.9952	+4.9952	+9.9902	+FSR-2 LSB	+½FSR-2 LSB	1 1 1 1 1 1 1 1 1 1 1 0
⋮	⋮	⋮	⋮	⋮	⋮
+5.0024	+0.0024	+0.0049	+½FSR+1 LSB	+1 LSB	1 0 0 0 0 0 0 0 0 0 0 1
+5.0000	+0.0000	+0.0000	+½FSR	ZERO	1 0 0 0 0 0 0 0 0 0 0 0
⋮	⋮	⋮	⋮	⋮	⋮
+0.0024	-4.9976	-9.9951	+1 LSB	-½FSR+1 LSB	0 0 0 0 0 0 0 0 0 0 0 1
+0.0000	-5.0000	-10.0000	ZERO	-½FSR	0 0 0 0 0 0 0 0 0 0 0 0

Table 4. Digital Output Codes vs Analog Input For Unipolar and Bipolar Ranges

The resulting input-output transfer functions are given by Table 4.

Analog Input Section Offset Adjust Circuit

The offset voltage of the AD363 may be adjusted at either the Analog Input Section or the ADC Section. Normally the adjustment is performed at the ADC but in some special applications, it may be helpful to adjust the offset of the Analog Input Section. An example of such a case would be if the input signals were small (<10mV) relative to Analog Input Section voltage offset and gain was inserted between the Analog Input Section and the ADC. To adjust the offset of the Analog Input Section, the circuit shown in Figure 6 is recommended.

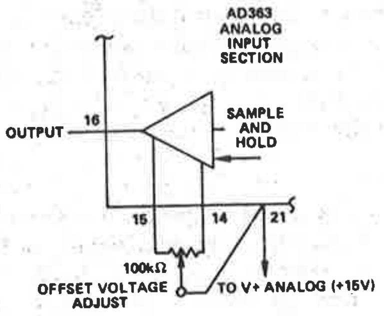
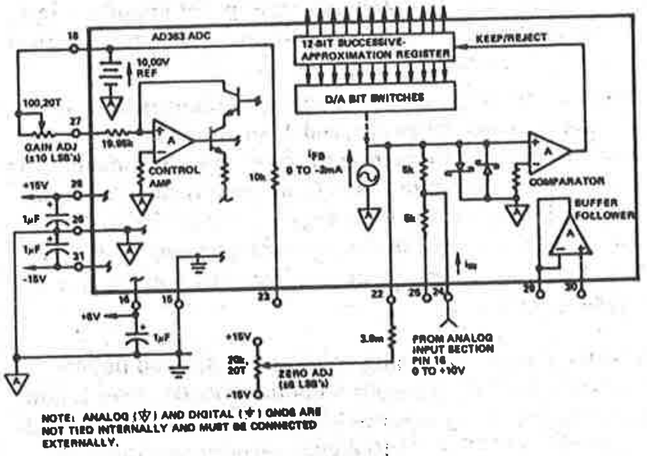


Figure 6. Analog Input Section Offset Voltage Adjustment

Under normal conditions, all calibration is performed at the ADC Section.

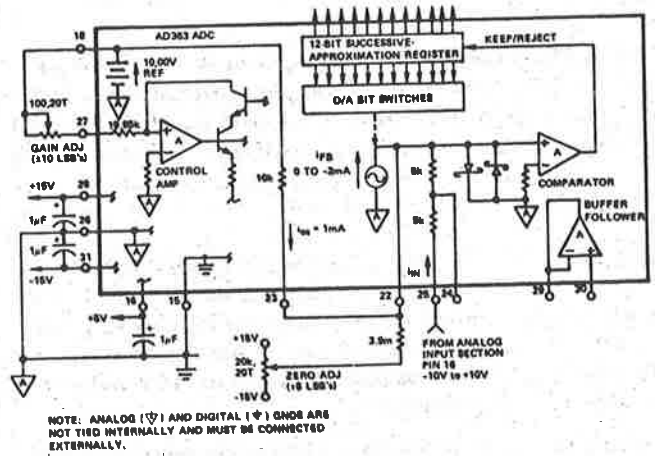
ADC Offset Adjust Circuit

Analog and power connections for 0 to +10V unipolar and -10V to +10V bipolar input ranges are shown in Figures 7 and 8, respectively. The Bipolar Offset, ADC pin 23 is open-circuited for all unipolar input ranges, and connected to Comparator input (ADC pin 22) for all bipolar input ranges. The zero adjust circuit consists of a potentiometer connected across $\pm V_S$ with its slider connected through a 3.9MΩ resistor to Comparator input (ADC pin 22) for all ranges. The tolerance of this fixed resistor is not critical, and a carbon composition type is generally adequate. Using a carbon composition resistor having a -1200ppm/°C tempco contributes a worst-case offset tempco of $8 \times 244 \times 10^{-6} \times 1200$ppm/°C = 2.3ppm/°C of FSR, if the OFFSET ADJ potentiometer is set at either end of its adjustment range. Since the maximum offset adjustment required is typically no more than ±4LSB, use of a carbon composition offset summing resistor normally contributes no more than 1ppm/°C of FSR offset tempco.



AD363 ADC
12-BIT SUCCESSIVE-APPROXIMATION REGISTER
KEEP/REJECT
D/A BIT SWITCHES
10.00V REF
100,20T
GAIN ADJ (±10 LSB's)
CONTROL AMP
COMPARATOR
BUFFER FOLLOWER
0 TO -2mA
+15V
-15V
1µF
3.9M
FROM ANALOG INPUT SECTION PIN 16 0 TO +10V
ZERO ADJ (±8 LSB's)
20k, 20T
NOTE: ANALOG AND DIGITAL GNDS ARE NOT TIED INTERNALLY AND MUST BE CONNECTED EXTERNALLY.

Figure 7. ADC Analog and Power Connections for Unipolar 0 to +10V Input Range



AD363 ADC
12-BIT SUCCESSIVE-APPROXIMATION REGISTER
KEEP/REJECT
D/A BIT SWITCHES
10.00V REF
100,20T
GAIN ADJ (±10 LSB's)
CONTROL AMP
COMPARATOR
BUFFER FOLLOWER
0 TO -2mA
= 1mA
+15V
-15V
1µF
3.9M
FROM ANALOG INPUT SECTION PIN 16 -10V to +10V
ZERO ADJ (±8 LSB's)
20k, 20T
NOTE: ANALOG AND DIGITAL GNDS ARE NOT TIED INTERNALLY AND MUST BE CONNECTED EXTERNALLY.

Figure 8. ADC Analog and Power Connections for Bipolar -10V to +10V Input Range

An alternate offset adjust circuit, which contributes negligible offset tempco if metal film resistors (tempco <100 ppm/°C) are used, is shown in Figure 9.

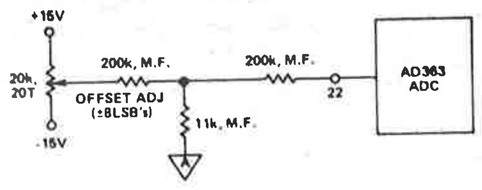


Figure 9. Low Tempco Zero Adj Circuit

In either zero adjust circuit, the fixed resistor connected to ADC pin 22 should be located close to this pin to keep the connection runs short, since the Comparator input (ADC pin 22) is quite sensitive to external noise pick-up.

Gain Adjust

The gain adjust circuit consists of a 100Ω potentiometer connected between +10V Reference Output pin 18 and Gain Adjust Input (ADC pin 27) for all ranges. Both GAIN and ZERO ADJ potentiometers should be multi-turn, low tempco types; 20T cermet (tempco = 100ppm/°C max) types are recommended. If the 100Ω GAIN ADJ potentiometer is replaced by a fixed 50Ω resistor, absolute gain calibration to ±0.1% of FSR is guaranteed.

Calibration

Calibration of the AD363 consists of adjusting offset and gain. Relative accuracy (linearity) is not affected by these adjustments, so if absolute zero and gain error is not important in a given application, or if system intelligence can correct for such errors, calibration may be unnecessary.

External ZERO ADJ and GAIN ADJ potentiometers, connected as shown in Figures 7, 8, and 9, are used for device calibration. To prevent interaction of these two adjustments, Zero is always adjusted first and then Gain. Zero is adjusted with the analog input near the most negative end of the analog range (0 for unipolar and -½FSR for bipolar input ranges). Gain is adjusted with the analog input near the most positive end of the analog range.

0 to +10V Range: Set analog input to +1LSB = +0.0024V. Adjust Zero for digital output = 000000000001; Zero is now calibrated. Set analog input to +FSR -2LSB = +9.9952V. Adjust Gain for 111111111110 digital output code; full-scale (Gain) is now calibrated. Half-scale calibration check: set analog input to +5.0000V; digital output code should be 100000000000.

-10V to +10V Range: Set analog input to -9.9951V; adjust Zero for 000000000001 digital output (offset binary) code. Set analog input to +9.9902V; adjust Gain for 111111111110 digital output (offset binary) code. Half-scale calibration check: set analog input to 0.0000V; digital output (offset binary) code should be 100000000000.

Other Ranges: Representative digital coding for 0 to +10V, -5V to +5V, and -10V to +10V ranges is shown in Table 4. Coding relationships are calibration points for 0 to +5V and -2.5V to +2.5V ranges can be found by halving the corresponding code equivalents listed for the 0 to +10V and -5V to +5V ranges, respectively.

Zero and full-scale calibration can be accomplished to a precision of approximately ±¼LSB using the static adjustment procedure described above. By summing a small sine or triangular-wave voltage with the signal applied to the analog input, the output can be cycled through each of the calibration codes of interest to more accurately determine the center (or end points) of each discrete quantization level. A detailed description of this dynamic calibration technique is presented in "A/D Conversion Notes", D. Sheingold, Analog Devices, Inc., 1977, Part II, Chapter II-4.

Other Considerations

Grounding: Analog and digital signal grounds should be kept separate where possible to prevent digital signals from flowing in the analog ground circuit and inducing spurious analog signal noise. Analog Ground (Analog Input Section pin 17, ADC Section pin 26) and Digital Ground (Analog Input Section pin 2 and ADC Section pin 15) are not connected internally; these pins must be connected externally for the system to operate properly. Preferably, this connection is made at only one point, as close to the system as possible. The cases are connected internally to Digital Ground to provide good electrostatic shielding. If the grounds are not tied common on the same card with both system packages, the digital and analog grounds should be connected locally with back-to-back general-purpose diodes as shown in Figure 10. This will protect the AD363 from possible damage caused by voltages in excess of ±1 volt between the ground systems which could occur if the key grounding card should be removed from the overall system. The system will operate properly with as much as ±200mV between grounds, however this difference will be reflected directly as an input offset voltage.

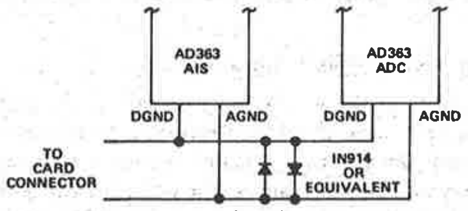


Figure 10. Ground-Fault Protection Diodes

Power Supply Bypassing: The ±15V and +5V power leads should be capacitively bypassed to Analog Ground and Digital Ground respectively for optimum device performance. 1µF tantalum types are recommended; these capacitors should be located close to the system. It is not necessary to shunt these capacitors with disc capacitors to provide additional high frequency power supply decoupling since each power lead is bypassed internally with a 0.039µF ceramic capacitor.

Applications

The AD363 contains several unique features that contribute to its application versatility. The more significant features include a precision +10V reference, an uncommitted buffer amplifier, the dynamic single-ended/differential mode switch and simple, uncommitted digital interfaces.

Transducer Interfacing

The precision +10V reference, buffer amplifier and mode switch can simplify transducer interfacing. Figure 11 illustrates how these features may be used to advantage.

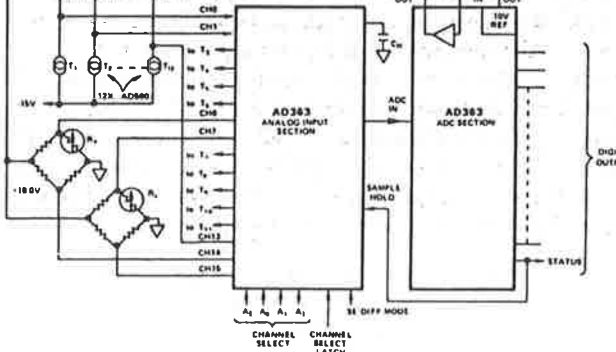


Figure 11. AD363 Transducer Interface Application

The AD590 is a temperature transducer that can be considered an ideal two-terminal current source with an output of one microamp per degree Kelvin (1μA/°K). With an offsetting current of 273μA sourced from the +5.46 volt buffered reference through 20kΩ resistors (R1-R12) each of the 12 AD590 circuits develop -20mV/°C. The outputs are monitored with the AD363 front-end in the single-ended mode (Logic "0" on the Mode Control input). The +5.46 volt reference is derived from the ADC +10 volt precision reference and voltage divider R13, R14. Low output impedance for this +5.46 volt reference is provided by the ADC internal buffer amplifier. (The 10μV/°C offset voltage drift of the buffer amplifier contributes negligible errors.) At 0°C, each temperature transducer circuit delivers a 0 volt output. At 125°C, the output is -2.5V; at -55°C, the output is +1.10V. By using the two's complement ADC output (complemented MSB or sign bit), the negative voltage versus temperature function is inverted and digital reading proportional to temperature in degrees centigrade is provided. Resolution is 0.061°C per least significant bit.

The precision +10 volt reference is also used to power several bridge circuits that require differential read-out. When addressing these bridge transducers, a Logic "1" at the mode control input will switch the AD363 to the differential mode. In many cases, this feature will eliminate the requirement for a differential amplifier for each bridge transducer.

Microprocessor Interfacing

Digital interfacing to the AD363 has been deliberately left uncommitted; every processor system and application has different interface requirements and designing for one specific processor could complicate other applications.

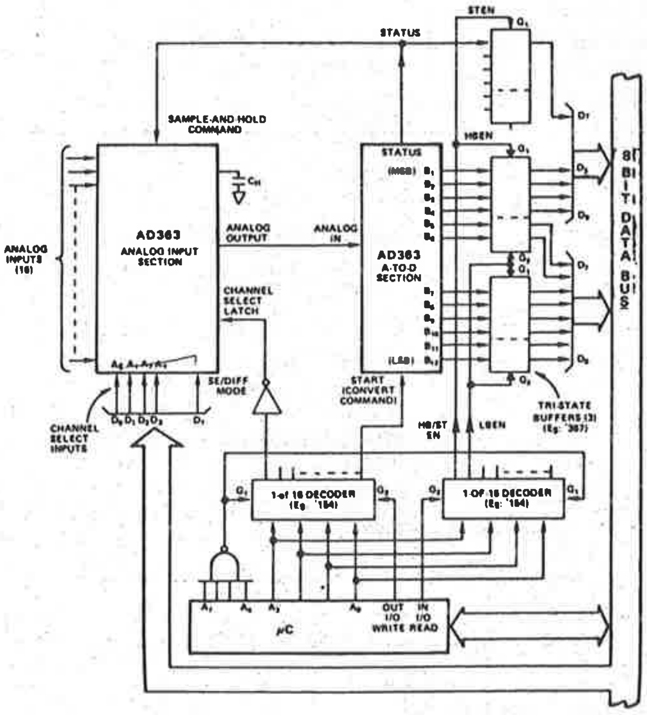


Figure 12. AD363 Microprocessor Interface Application

The addition of a small amount of hardware will satisfy most interface requirements; an example based on 8080-type architecture is shown in Figure 12.

In this system the data bus is used to transmit multiplexer channel selection and convert and read commands to the AD363. It is also possible to address the AD363 as memory using the address bus to perform channel selection, convert and read operations.

The address lines can be decoded to provide channel selection, ADC convert start, status and ADC data (2 bytes) locations. These are accessed with I/O read/write instructions.

The ADC outputs are buffered with tri-state drivers. Figure 12 shows the 4 most significant ADC data bits and status as one byte

FF_H:	STATUS	—	—	—	B1 (MSB)	B2	B3	B4
	D7							D0

and the 8 least significant ADC data bits as the second byte.

FE_H:	B5	B6	B7	B8	B9	B10	B11	B12 (LSB)
	D7							D0

Internal tri-state buffering is not provided because in many applications it would be better to have the first byte contain the 8 most significant bits. To accomodate both left and right justified formats would require more package pins and increase complexity.

The operating sequence for this system is as follows:

1	MVI	80_H	puts the address for channel 0 (including SE/DIFF mode) into accumulator
2	OUT	FF_H	puts 80_H on data bus and FF_H on address bus. Pulses I/O WRITE. OUT FF_H is decoded as a "LOAD ADDRESS" command to the channel select latches.
3	OUT	FO_H	puts FO_H on address bus and pulses I/O WRITE. This is decoded to issue a "CONVERSION START" to the ADC. Accumulator contents are of no significance.
4	IN	FF_H	puts FF_H on address bus and pulses I/O READ. This is decoded to enable the appropriate tri-states, thus putting status and the 4 most significant bits on the data bus.

The status may be examined for "0" (conversion complete). In that case, the 4 MSB's would be read.

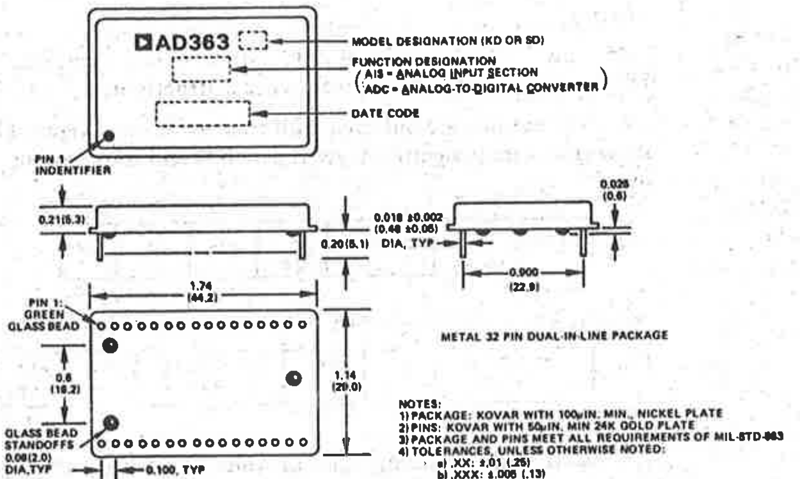
5	IN	FE_H	puts FE_H on address bus and pulses I/O READ. This is decoded to enable the appropriate tri-states, thus putting the 8 least significant bits on the data bus.

At this point, the multiplexer channel selection may be changed and another channel processed with the same instruction set (steps 2 through 5).

OUTLINE DIMENSIONS
PACKAGE SPECIFICATIONS

Dimensions shown in inches and (mm).

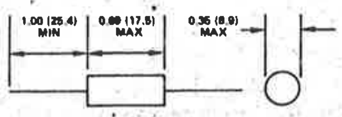
ANALOG INPUT SECTION AND ANALOG-TO-DIGITAL CONVERTER


AD363
MODEL DESIGNATION (KD OR SD)
FUNCTION DESIGNATION
(AIS = ANALOG INPUT SECTION
ADC = ANALOG-TO-DIGITAL CONVERTER)
DATE CODE
PIN 1 INDENTIFIER
0.21(5.3)
0.20(5.1)
0.018 ±0.002 (0.46 ±0.05) DIA, TYP
0.025 (0.6)
0.900 (22.9)
1.74 (44.2)
PIN 1: GREEN GLASS BEAD
0.6 (15.2)
1.14 (29.0)
GLASS BEAD STANDOFFS 0.08(2.0) DIA,TYP
0.100, TYP (2.5)
METAL 32 PIN DUAL-IN-LINE PACKAGE

NOTES:
1) PACKAGE: KOVAR WITH 100µIN. MIN., NICKEL PLATE
2) PINS: KOVAR WITH 50µIN. MIN 24K GOLD PLATE
3) PACKAGE AND PINS MEET ALL REQUIREMENTS OF MIL-STD-883
4) TOLERANCES, UNLESS OTHERWISE NOTED:
a) .XX: ±.01 (.25)
b) .XXX: ±.005 (.13)

MATING SOCKET: SET OF TWO 16 PIN SOCKET STRIPS (ORDER P/N AC1H72) T. ONE SET IS REQUIRED FOR EACH PACKAGE.

HOLD CAPACITOR



*THIS DIMENSION IS FOR POLYSTYRENE CAPACITOR SUPPLIED WITH K GRADE.

MAX BODY LENGTH OF TEFLON CAPACITOR SUPPLIED WITH S GRADE IS 1.00"

PROCESSING FOR HIGH RELIABILITY

STANDARD PROCESSING

As part of the standard manufacturing procedure, all models of the AD363 receive the following processing:

PROCESS	CONDITIONS
1) 100% pre-cap Visual Inspection	In-house Criteria
2) Stabilization Bake	24 hours @ +150°C
3) Seal Test, Gross Leak	Method 1014 Test Condition C
4) Operating Burn-In	48 hours @ +125°C

PROCESSING TO MIL-STD-883

All models of AD363 ordered to the requirements of MIL-STD-883B, Method 5008 are identified with a /883B suffix and receive the following processing:

PROCESS		CONDITIONS
1) 100% pre-cap Visual Inspection		2017.1
2) Stabilization Bake		1008, 24 hours @ +150°C
3) Temperature Cycle		1010, Test Condition C, 10 cycles, -65°C to +150°C
4) Constant Acceleration		2001, Y1 Plane, 1000G
5) Visual Inspection		Visible Damage
6) Operating Burn-In		1015, Test Condition B 160 hours @ +125°C
7) Seal Test:	Fine Leak	1014, Test Condition A, 5 x 10^{-7}std cc/sec
	Gross Leak	1014, Condition C
8) Final Electrical Test		Per Data Sheet
9) External Visual Inspection		2009

AD363 ORDERING GUIDE

Model	Specification Temp Range	Max Gain T.C.	Max Reference T.C.	Guaranteed Temp Range No Missing Codes
AD363KD	0 to +70°C	±30ppm/°C	±20ppm/°C	0 to +70°C
AD363SD	-55°C to +125°C	±25ppm/°C	±10ppm/°C	-55°C to +125°C
AD363SD/883B	Meets all AD363SD specifications after processing to the requirements of MIL-STD-883B, Method 5008.			

NOTE: D Suffix = Dual-In-Line package designator.

BB

DIGITAL-to-ANALOG CONVERTERS

Specifications typical at 25°C and rated supply voltage unless otherwise noted.

MODEL	UNITS	DAC80 LOW COST IC		DAC85C ECONOMY IC		DAC85 GENERAL PURPOSE IC	
RESOLUTION							
Binary	Bits	12		12		12	
Decimal	Digits		3		3		3
INPUT							
INPUT CODES(1) (2)							
Binary		CBI		CBI		CBI	
Decimal			CCD		CCD		CCD
TRANSFER CHARACTERISTICS							
ACCURACY							
Linearity Error, max @ 25°C							
Binary Models	% of FSR	±0.012		±0.012		±0.012	
Decimal Models	% of FSR		±0.05		±0.05		±0.05
Gain Error (Adj. to zero)	% of FSR	±0.1		±0.1		±0.1	
Unipolar Offset Error (Adj. to zero)	% of FSR	±0.05		±0.05		±0.05	
ACCURACY DRIFT							
Gain Drift, max	ppm/°C	±30		±20		±20	
Offset Drift, – Unipolar	ppm of FSR/°C	±1		±1		±1	
Combined Gain & Offset Drift, max	ppm of FSR/°C	—		—		—	
Linearity Error Over Temperature	% of FSR	±0.012†		±0.012†	±0.05†	±0.012†	±0.05†
Specified Operating Temperature	°C	0 to +70		0 to +70		-25 to +85	
CONVERSION SPEED							
Settling Time to ±1/2 LSB(Unipolar)	μsec	3 (V_{out}), 0.3 (I_{out})		3(V_{out}), 0.3(I_{out})		3 (V_{out}), 0.3 (I_{out})	
Slew Rate	V/μsec	20		20		20	
OUTPUT							
VOLTAGE RANGE							
Unipolar	Volts	0 to +5, 0 to +10					
Bipolar	Volts	±2.5, ±5, ±10					
Current, min	mA	±5					
Output Impedance	Ω	0.05					
CURRENT RANGE							
Unipolar	mA	0 to -2					
Bipolar	mA	±1					
Compliance (Unipolar/Bipolar)	Volts	±2.5					
Impedance (Unipolar/Bipolar)	Ω	15k / 4.4k					
POWER SUPPLY							
Voltages (rated)	Volts	±15, +5(5)					
Current Drain ±15V Supply, +5V Supply	mA	±25, +20					
Sensitivity	% of FSR/%	±0.002(3), ±0.02(4)					
PACKAGE DRAWING (See pages 92 – 101)		(28) A 0.8" x 1.4" x 0.25" CERAMIC		(27) A 0.8" x 1.4" x 0.22" METAL			
PRICE (1 - 9)		\$26.50	\$26.50	\$69.00	\$69.00	\$89.00	\$89.00

(1) All input codes are TTL compatible.

Prices and specifications are subject to change without notice.

(2) Input codes are designated:

CBI - Complementary Binary
BIN - Straight Binary
BOB - Bipolar Offset Binary
BTC - Bipolar Two's Complement
CCD - Complementary BCD
BCD - Binary Coded Decimal

†Maximum; monotonicity guaranteed over operating temperature range.

ORDERING INFORMATION

DAC70

DAC70X – XXX – X

Basic Model Number
DAC70
or
DAC70C

Input Code
CSB = 16 Bit Complementary straight Binary
COB = 16 Bit Compl. offset Binary
CCD = 4 digit compl. BCD

Output
I = Current

DAC90

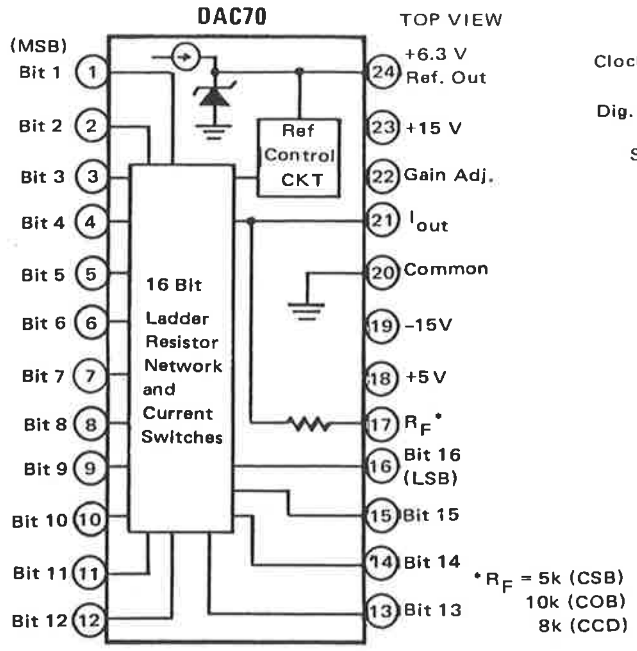
DAC90 X X

Basic Model Number

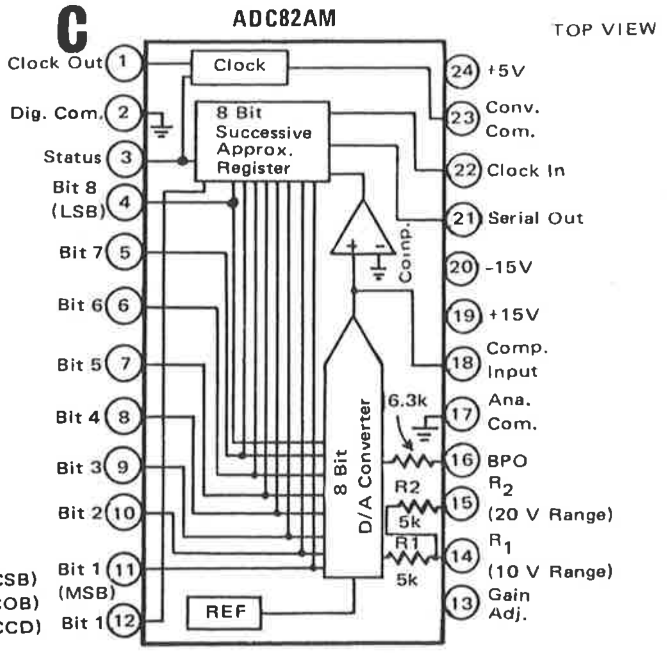
Grade
J = 0.8% 0 to +70°C
K = 0.4% 0 to +70°C
L = 0.2% 0 to +70°C
R = 0.8% -55 to +125°C
S = 0.4% -55 to +125°C
T = 0.2% -55 to +125°C

Package
P = Plastic
(J, K, L Grades)
G = Ceramic
(R, S, T Grades)

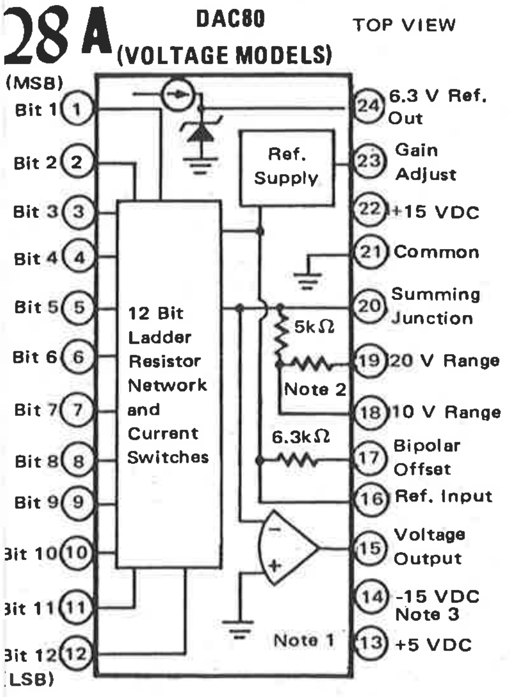
B



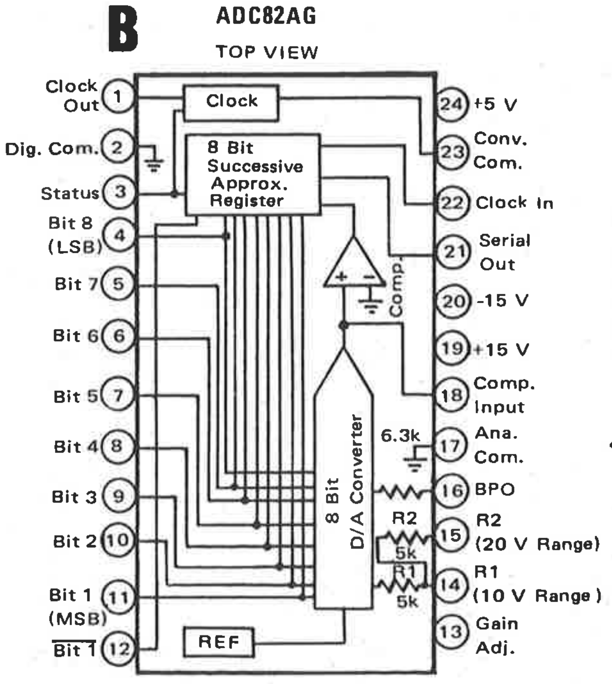
C



28 A



B



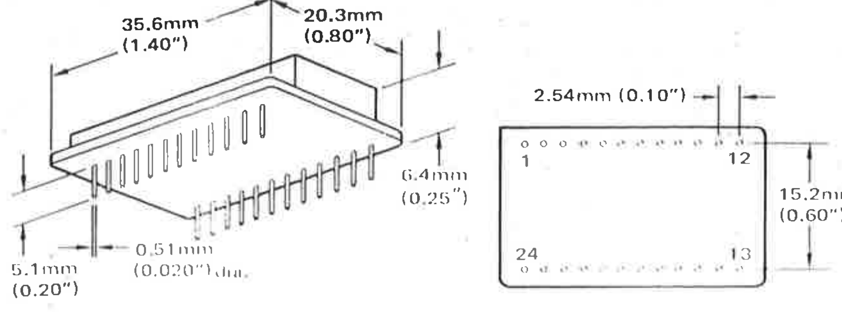
C **4127**

1	I_{Ref} Output
2	I_{Ref} Input
4	+I Input
5	Current Inverter Output
7	Current Inverter Input
9	Op Amp +Input
10	Op Amp –Input
11	Op Amp Output
13	Make No Connection
14	Negative Supply
18	Log Output
19	Gain Adjust
21	Common
22	Positive Supply
23	I_{Ref} Bias

*No Pin on 3,6,8,12,15,16, 17,20,24

te 1: Amplifier not included in current output models.
te 2: 3kΩ for CCD models
5kΩ for CBI models
te 3: +5 V supply input may be connected to +5 V supply if +5 V supply is not available
is will increase internal power dissipation by 200 mW.

Connector: 245MC



BOTTOM VIEW
Case: Black Ceramic

APPENDIX B

SOFTWARE DESCRIPTION

APPENDIX B

B.1 DDACS DESCRIPTION

The following description of DDACS has been extracted from a user's guide compiled by Dr. C.P. Jeffreson as part of a course on "Digital Process Control" given by the Chemical Engineering Department, University of Adelaide.

APPENDIX 1

UNIVERSITY OF ADELAIDE
CHEMICAL ENGINEERING DEPARTMENT

DDACS USERS GUIDE

1. INTRODUCTION

DDACS (Direct Digital Automatic Control System) is a self contained special purpose operating system developed by the Central Electricity Generating Board (N.E. Region Scientific Services Department [1]) designed to run on PDP11 series computers. Although intended originally for steam generating plant, it is also useful controlling more general processes and for teaching. This manual describes the facilities available in the 32K version as configured for the Chemical Engineering Department PDP11/03 computer and will be updated as the system is changed. The detailed summary on page 128 quotes extensively from A.G. Pink's User Guide Issue 3, an internal publication of the C.E.G.B. Introductory notes are by C.P. Jeffreson.

A number of facilities provided on the original C.E.G.B. systems are not available on this reduced version because appropriate hardware is not installed. They may be conditionally assembled into the system later. These include the INCS, DRIVE, DRIVEH and PULSE blocks (used with incremental actuators) the WDOG or watchdog timer block and the TTY block which allows terminal output to be directed to a second terminal. The facility to store schemes on a separate disc and graphics capability is available on other versions.

2. GENERAL FACILITIES AND PHILOSOPHY

2.1 "SECURITY" AND "FAULT TOLERANCE"

An over-riding requirement of process plant operation is safety. The student will already be aware that control valves and other actuators must fail safe in the event of an air failure; a computer system is complex and must also fail safe and ensure that the hardware associated with it does the same. The system must hence be both "secure" and "fault tolerant".

Software security implies that the functioning existing "real time" control and logging programmes may not be disrupted by new programme development or by the uncontrolled expansion of data storage arrays. The student will be aware how easily his FORTRAN programmes "crash". The consequences of this are much more serious in a computer control language.

Flexibility, i.e. the ability to re-configure control loop programmes and "schemes" is also important but such reconfiguration must not affect the security of existing programmes. In a wider context, security includes control hardware and is concerned with the overall plant. The overall system [2] must "ensure that control will continue uninterrupted in the event of equipment failure". Some of these considerations arise in the lecture course.

The provision of "fault tolerance" extends to hardware as well as software; here we are concerned with the ways in which the computer software may be made to respond to (say) the failure of a measurement transmitter or to unacceptable data. This should be illustrated by control laboratory experiments. We consider how to handle computer malfunction in the lecture course.

The following particular requirements of process control are reflected in the rules of the DDACS language, in the "modes" of operation allowed and in the word structure and type provided.

1. Timing

The system must "keep time", initiating sampling operations on a strict "real time" scale. This is necessary also to ensure that integral and derivative times are those required by the control engineer. Fast and slow loops must be sampled at the rates specified, for example, sampling a supervisory concentration control loop every 10 seconds and a flow loop every 50 msec. TIME statements allow LOOP timing in DDACS.

2. Control and Filter Functions

The usual three term, (proportional integral derivative or PID) controller is required for feedback control but sufficient arithmetic facilities need to be provided to allow the engineer to configure his own control algorithms easily. Since PID is used frequently, a 'standard' form is provided as a function or "BLOCK" as well as integrator and differentiator blocks which could be linked in different ways if a non-standard controller algorithm is required.

In addition filters may be required, e.g. a first order lag (to reduce the effects of noise) or to provide a lead/lag transfer function for a feedforward system.

3. Flexibility in Configuring Loops

It should be possible to change control schemes readily on site without interrupting existing schemes. For this reason, the operating system can compile, copy and edit new schemes in the time left over from servicing "foreground" tasks requested by the scheduler. This flexibility is not usually available in conventional analogue control systems. This implies a considerable change in the way a process control system may be specified and commissioned, since the computer control system vendor need only be given the specification of the number and type of analogue and digital inputs and outputs rather than a detailed specification of the loops. Changes in the configuration of loops may often be made after startup by software rather than by complex re-wiring operations.

4. Fault Tolerance and Alarm Signalling

If an attempt is made to extract the square root of a negative number in a conventional system an error is flagged and the system halts or "crashes". The control engineer requires the system to continue operation so far as it can but to signal the fault. For this reason, a further data type "bad" is defined in addition to the usual ones of real, integer and logical. For example, the variable Y representing the result of the assignment statement

Y:=SQRT(X)

will be assigned the value "bad" if X is negative, but the system will continue operating. Obviously, facilities need also to be provided to signal the condition of Y to an operator. The system should respond appropriately to (say) an open circuit in a measuring element. The "bad data" flag is also useful in detecting whether a measured variable is outside the range specified. Digital I/O allows greater flexibility in signalling fault conditions to an operator control panel.

5. Software Security

In a conventional multiuser real time system, a number of FORTRAN programmes may execute apparently simultaneously. Each programme will provide with a separate copy of such commonly used functions as SQRT and their location in core may vary from job to job. In the DDACS system, each block remains in the same location and is accessed by each SCHEME in turn. Some commercial system vendors call this "softwiring" of blocks, as distinct from the "hardwiring" necessary for conventional analogue systems.

The function of each block is clearly defined and TABLES and parameters used by the schemes are also strictly defined and localised. Hence the structure of the operating sytem helps provide software and ultimately system security. Checks are also built into the system to prevent accidental interference with existing control schemes during programme development. For example, it is not possible to DELETE any scheme which is running or to delete any table used by any scheme whether running or not. "Tables" are used to convert measured quantities to problem units such as degC and lb/min. The ability to linearise thermocouple readings or control valves is also assisted by the use of such interpolation tables. Tables may also be written into and read by schemes. However unlike conventional FORTRAN arrays, such data storage areas cannot expand beyond the defined areas in memory.

For obvious reasons, a real time SCHEME cannot be allowed to become "hung up" in a repeated loop. Hence branching by IF and GOTO statements is allowed only in the forward direction. This allows simpler line by line compilations or translation since earlier labels are not referenced by later programme branching statements. Such line by line translation is usually only available with much slower "interpretive" languages such as BASIC or FOCAL.

Because of the need to reconfigure the system without disturbing existing schemes, variables are not accessible between schemes, i.e., are not "global". However parameters may be passed between schemes through common tables. Improper modification of the data in these tables is not allowed; for example a WRITA block cannot change the data stored in a TABLE used for interpolating A/D and D/A conversions.

6. "Background" Computations

The control engineer may also expect a computer control system to periodically compute, for example, the thermal efficiency of a fired heater as a "background" scheme or to carry out a material or energy balance over the plant. Such a facility would only be available with great difficulty in a conventional analogue system but would reasonably be expected from a digital system. DDACS does allow one "background" scheme to run while the system is not attending to real time loops but extensive and complex background tasks are not possible with the present version of DDACS. A separate "higher level" computer would be more appropriate if extensive offline computations are required. The reader will note that a branch back to an earlier segment of programme is allowed in background schemes since the consequence of a repeated loop is not serious. However nesting of GOTO's is not allowed in the present version of DDACS.

It should be noted that for on-line control the computer must always respond to the clock; it must always provide an output and sample a loop variable at the time required. All control functions are hence in the foreground. Elaborate graphic display and logging facilities are given low priority in a computer system designed for on line control. There is a case for providing such facilities in a separate computer and display station. The present system was designed to interact with an existing operator's panel with computer auto/local manual switching and would be used by an engineer for background development rather than for on line control.

7. Sequencing Functions

In starting up the burner system of a fired heater, a complex set of operations involving logic and timing is necessary. For example, the combustion chamber must be purged for a set time and then a pilot burner lit. The next stage of the operation can only be proceeded with IF the flame detector "proves" the pilot, otherwise a "flame failure" indicator must come on and the purging operation repeated IF the operator manually "resets" the sequence. A similar combination of logic and timing is required for BATCH processes involved in filling and heating up a batch chemical reactor. Such operations could be carried out by real time or foreground DDACS schemes, provided IF statements are used to branch to the appropriate stage in the sequence every time the scheme is entered. However, programming is complex; what is really required is the facility to start a suitable background scheme once automatically from another SCHEME. Timing then becomes difficult since, by definition, a background scheme only runs when the computer "clock scheduler" sees that time is available not required to service real time operations. The present (Adelaide) version of DDACS described in this manual is hence not really suitable for such batch sequencing operations.

8. Auto/Manual and Remote/Local Transfer

The control engineer requires a smooth transition between manual (or direct operator manipulation of the process) and automatic operation. This involves interaction with hardware and may be achieved in a number of ways by using mode (i.e. "initialise" or "normal") control via, for example, the AMS block. See laboratory notes for examples. A similar requirement arises when changing over from single loop or "local" operation of a cascade system to two loop operation when the set point of the inner, or secondary loop must be adjusted by an outer or primary controller.

Modes:

There are three "modes" of operation in DDACS and the BLOCKS (described from page 129 onwards) respond to these modes in different ways:

START mode is not usually visible to the operator. It is a transient state applicable on the first cycle through a SCHEME immediately after entering a START command.

INITIALISE mode is used by a number of special purpose blocks such as PID, INT and FIRST to set up an initial condition. For example, if the auto manual block AMS indicates that a scheme is in 'MANUAL' then that SCHEME will be set to INITIALISED and PID, INT and FIRST block outputs will simply track their initial condition.

If the SCHEME is on AUTO, then integrators will function in NORMAL mode, see below, until a CONSTRAINT occurs, at which stage the loop in which the constraint has occurred will alternate between initialise and normal modes.

NORMAL mode allows all functions to proceed normally. For example an INT block will simply integrate its input subject to its initial condition which was set last time the integrator was in initialise mode.

In summary, a Loop which is part of a Scheme on Auto can function in "initialise" or "Normal" mode depending on the existence of a CONSTRAINT whilst a all loops of a SCHEME on Manual will be in Initialise Mode. The use of mode control to prevent "Reset Windup" and ensure "bumpless" transfer is most easily understood after performing the laboratory experiments.

9 Operator Communication and Display

In addition to alarm indication, conventional analogue instruments usually provide analogue displays in the form of chart recorders and so on. This assists the operator to visualise trends and hence to take appropriate action. Although DDACS is able to drive a graphics terminal, the version used in the department does not have this capability and all communication (except from panel-mounted switches etc.) is through the V.D.U. at present for historical reasons already mentioned under 6. above.

2.2 REFERENCES AND FURTHER READING

[1] Johnstone, L.R., Marsland, C.R. and Pringle, S.T. "A Distributed Computer Control System for a 120 MW Boiler", I.E.E. Conf. Pub., 153, 1977, pp 114-119.

[2] Marks, H., "An Evolutionary look at Centralized Operation/2". Honeywell, Process Control Division, Washington, 1977.

[3] Smith, C.L., "Digital Computer Process Control" Intext, 1972.

[4] Bartlett, L.A., Marsland, C.R. and Smith, C.D. "A Guide to the Use of DDACS-MCS (Modulating Control System) C.E.G.B. (N.E. Region, U.K.) Report SSD/NE/N138, November, 1976.

3. DETAILED INSTRUCTIONS

The following sections give detailed descriptions of procedures and monitor commands.

3.1 TO START AND END A DDACS SESSION

Load system floppy disc in left hand drive (DXO), close cover, turn all three switches off (down), turn on "ENABLE/HALT" and "DC ON/OFF" switches in that order repeating if necessary until a REV11 prompt ($) appears on the VDU. Enter DXO followed by a carriage return. There will be some considerable delay (about 45 seconds) as disc is read in. When fully read, the system will respond with:-

```
        ADEL DDACS VER 80:4
!>
```

This is the Monitor "prompt" and indicates that the system monitor is available.

Turn the real-time clock ON (RH switch up)

At the End of a Session -

Preserve memory images on the floppy DXO including any SCHEMES entered during the session by typing:

```
DUMPD<CNTRL/X>
```

followed by a carriage return. When the entire system has been written on the disc (40-60 seconds) the DDACS monitor prompt above will occur. Power may then be turned off.

3.2 CONVENTIONS USED IN THIS GUIDE

Underlined characters represent user responses to computer prompts, Computer outputs not underlined.

O is a letter, 0 is a numeral

Control characters entered by the user are enclosed in angled brackets

e.g. <CNTRL/U> denotes press control key and U down simultaneously (first CNTRL then U).

Spaces and their absence are significant and should not be inserted unless shown.

3.3 SPECIAL CONTROL KEYS AND COMPUTER PROMPTS

Once the "return" key has been pressed, a line of keyboard input will be read by the computer. Until return is pressed there are two error correction facilities:-

(1) The Delete key deletes the last character on the screen, the correct character may then be entered.

(2) <CNTRL/D> deletes any line, and responds with a "new line" prompt:>

MONITOR LEVEL MAY ALWAYS BE RE-ENTERED AT ANY STAGE BY TYPING <CNTRL/U>.

Other Special Keys include:

<CNTRL/P>	Stops real time clock and re turns to Monitor.
<CNTRL/B>	Gives time and date.

The above do not require return key for execution

NOTE The compiler (scanner) will not recognise a leading decimal point or zero i.e. .5 or 0.5 must be entered as 5E-1. Similarly, large numbers with more digits than the capacity of the 16-bit mantissa should be entered in exponent form. For example, to set the integral time TI to a large number (to remove integral action) enter 1E5 or 1E10 instead of 100000. Trailing decimal points and zeros e.g. 1, or 1.0 will not be recognised.

Prompts

At most stages of DDACS when a prompt has been issued, the possible options will be listed in a ? is entered in reply. For example, in LIST, a ? will result in a listing of all the user's scheme names before continuing with a detailed listing. In monitor mode indicated by:

the list of all monitor facilities shown below will results e.g.

TIME	Enter time and date
START	Start all ENABLED foreground or real time schemes
STOP	Stop all enabled foreground schemes
EDITOR	Display and/or modify scheme parameters
LIBRARY	List available library of block names
SCHEME	Create new scheme
TABLE	Create new table
LIST	List user-created schemes and tables
COPYED	Copy existing scheme or table line by line allowing Editorial additions and deletions.
DELETE	Delete existing schemes and tables not in active use
ENABLE	Allows any real time scheme to be started or executed
DISABLE	Allows any real time scheme to be prevented from execution
EXECUTE	Starts one background (non real time) scheme
ABORT	Stops execution of background schemes, provided the SCHEME is running
RESUME	Continue execution of background scheme
DEVICES	Displays locations of device registers, for example the floppy disc control-status register is at 177170
TESTWD	also KILL, RUN EDIT: used in detailed software debugging
TABSWP	Used to change table references SEE PAGE
DUMPD<CTR/X>	Dumps DDACS system to disc See page

4. MONITOR FACILITIES AND COMMANDS

4.1 TIME

At any level of DDACS if a <CTRL/B> command is entered the terminal will printout the current time and date. Following that printout, the reader handler acts as if a <CTRL/D> command (i.e. delete current line entry) and causes the prompt to be output again.

To set the correct time and date, enter TIME and correct entry if necessary. Otherwise enter a carriage return e.g.

```
!> TIME
1:2:3 1/2/78
SECS 3>0
MINS 2>30
HRS >11   Sets new time and date, 11:30:00
          7/6/80
DATE >7
MNTH 2>6
YEAR 78>80
```

New data can be entered in reply to the prompt, or if no new data is entered, the previous value is retained. The real time clock schedulaer for schemes is started by a START command and stopped by a STOP command or <CNTRL/P>.

Enter library to display all blocks available on this system. At present, these cannot be held on the screen (use printing terminal).

4.3 TABLE

Before creating a new control scheme, all tables to be referenced must be set up. For example, the analogue to digital converter block AAV at present is hardware configured to convert an internal variable (say OP, ranging from -10 to 340) to an output voltage ranging from a minimum of -5.12 volts to +5.12 volts. Hence the table:

```
        -10 0   (min voltage, -5.12)
        340 1   (max voltage, +5.12)
```

will produce a (linearly interpolated) voltage of 2.56 volts

when OP is 257.5 (Figure 1)

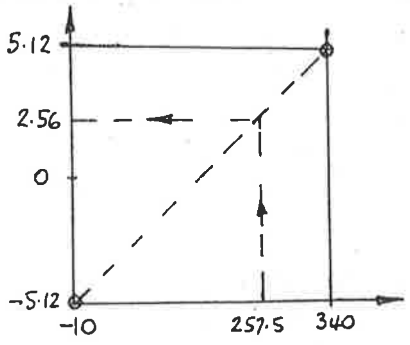


Figure 1

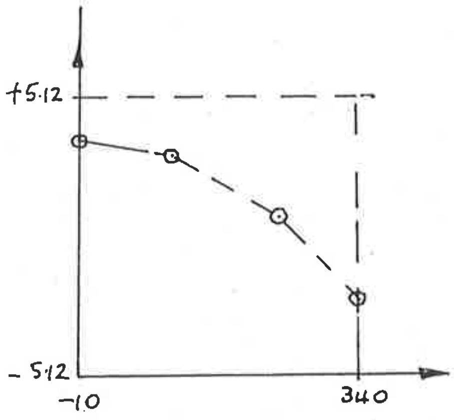


Figure 2

Further break points for table pairs may be entered (Figure 2) if, for example, it is desired to linearise a final control element. However L.H. entries must be in ascending order, they must be monotonic increasing or decreasing. As in figure 2, the full range of the output (or input) need not be used, but variables outside that range will be flagged as "bad", see AAV and ADV blocks for further details (see p18 also). The monitor will ask for the block which is to use the table. In the example below, an ADV block (i.e. analogue to digital converter) is to reference a table called ADV2 which converts a 0 to +5.12 volt input to internal (perhaps engineering) units ranging from 35.2 down to 22.3:

```
!>TABLE

NAME?>ADV2 ; Name can be any length

BLOCK?>ADV ; ADV11 will be used

        >5E-1 36.2 ; The user enters succes-
                     ive coordinate pairs
                     until the TABLE is

        >1 22.3    ; terminated by
           **      ; two asterisks
```

Spaces between entries may be used for clarity and the table terminated with two asterisks. Trailing zeroes and decimals will generate a compiler error. TABLES can also be typed in with all the elements of the TABLE on the same line, still separated from each other by a space. The terminating asterisks must be on another line.

TABLE can also be used to prepare arrays for subsequent use by blocks WRITA and READA See page 22

```
!>TABLE

NAME?>TABLE 1

BLOCK>SIZE          ; Provide space for 2x3
                      array called TABLE 1
COLS>2
ROWS3

**                  ; Terminate TABLE
```

The value of individual elements of the array will be set (and read) by blocks WRITA and READA respectively on execution.

ALL TABLES MAY BE ACCESSED BY MORE THAN ONE SCHEME if desired.

4.4 Scheme

SCHEME is used to create a new SCHEME which will be subsequently ENABLED and STARTED (real time, foreground SCHEMES) or EXECUTED (background scheme).

For example, the following schene called EXAMPLE 1 will read a thermocouple transmitter output connected to channel 1 of the ADV11 analogue to digital converter converting, through TABLE ADVMV, the value to deg C by linear interpolation. Setpoint, SP and Regulator Output RG are also read from voltage regulators and an error ER calculated. (NOTE that variable names in DDACS may only be a maximum of two characters in length).The output, OP of this single loop scheme is obtained from the PID (Controller) block, see page 25. Output is then sent to channel 0 of the AAV11 Digital to Analogue Converter.

```
NAME?>EXAMPLE1                              ;These Comments are not allowed in DDACS
  ! >TIME(4)                                ;Real time SCHEME to run every (2**4)*10=160mSec
  ! >BREAD(0,ST)                            ;Read State, ST, of SCHEME from DRV11 Digital I/O
  ! >AMS(ST)                                ;Uses ST to set state to Auto or Manual
  ! >ADV($ADVMV,1,MV)                       ;Read Measured variable (Thermocouple TX)
  ! >ADV($ADVSP,O,SP,2,IC)                  ;Also Setpoint and PID Initial Condition
  ! >ER:=SP-MV                              ;Arithmetic Assignment statement calculates error
  ! >OP:=PID(ER,IC,K,TI,TD)                 ;PID controller block with usual parameters
  ! >AAV($DACOP,OP,O,O)                     ;Output to (Analogue) Final Control Element
  ! >MONITOR(ST,SP,MV,IC,ER,K,TI,TD,OP); Gathers SCHEME variables
  ! >TIME(0)                                ;Terminates Loop and SCHEME
```

A SCHEME must start with a TIME(n) statement, and end with a TIME(0) statement, where TIME(1) refers to a loop rate of 20mS, TIME(2) refers to a loop rate of 40mS, TIME(3) refers to a loop rate of 80mS etc. TIME statements may not appear successively and no two TIME statements within the same SCHEME may be the same. Although there is only one LOOP in SCHEME EXAMPLE 1 above, a Cascade loop would probably use two with the faster, inner loop set by the outer. TIME statements define LOOP boundaries. All labels specified after a TIME statement must be satisfied before the next TIME statement i.e. no referencing of labels in other LOOPS is permitted. TIME statements must be entered in order of decreasing period. A background SCHEME may be created by starting with a TIME(@) and ending with TIME(0). Such a background scheme will be EXECUTED when time is available (not required for real-time SCHEMES). COMPILER (scanner or parser) errors are listed on pages 29 and 30.

4.5 Copyed

COPYED is used to copy or modify an existing SCHEME or TABLE. For example, in the following dialogue, we use COPYED to modify EXAMPLE 1, adding an IF Block to branch to the Label L2: when MV becomes "Bad" (perhaps because of a broken thermocouple lead):

```
NAME?>EXAMPLE1            ;Old name
NAME?>EXAMPLE2            ;New Name
      TIME(4)             ;Accept this Statement?
 *>A                      ;Yes
      BREAD(O,ST)
 *>A
    :                     ;And so on
 *>A
      ER:=(SP-MV)         ;This one too?
 *>I                      ;Maybe, insert a few lines
                           first though
     >IF(MV,L1:,L2:)      ;Branch to L2: if MV "Bad"
     >L1:ER:=SP-MV
     >                    ;"Return" Key Terminates
                           Insertion
      ER:=(SP-MV)         ;Don't need this now so
 *>R                      ;Enter R for "Reject".
    :                     ;and so on until last
                           statement
      TIME(0)
 *>A
```

The reply A causes the line previously printed out to be accepted, whilst R causes it to be rejected. I is used to insert one or more lines. The reply E causes the input to be accepted to the end of the SCHEME.

4.6 List

LIST is used to create a print out of SCHEMES or TABLES. As noted before, a ? will result in a listing of all schemes and tables:

```
NAME?> ?
   ! EXAMPLE1                 ;Indicates  a SCHEME
                               (Not ENABLED)
   $ ADVSP                    ;Indicates a TABLE
   $ DACOP
   @ RESET                    ;Indicates a BACKGROUND
                               Scheme
   $ ADVMV
 # ! EXAMPLE2                 ;Indicates an Enabled
                               SCHEME
NAME?>EXAMPLE2
      TIME(4)
      BREAD(0,ST)
      AMS(ST)
      ADV($ADVMV,1,MV)
      ADV($ADVSP,O,SP,2,IC)
      IF(MV,L10:,L11:)        ;Compiler Changed these
                               from L1: and L2:
L10   ER:=)SP-MV)
   :
   etc.
NAME?>DACOP                   ;Can Also List Tables:
                  0.0000E+00  1.0000E+00
                  1.0000E+02  6.3700E-01

                  **

  !>
```

Real time SCHEMES will only be executed if they are enabled andaSTART command has been given e.g.

```
!> ENABLE
NAME?> EXAMPLE2           ;Enable EXAMPLE2

NAME?> <CNTRL/U>          ;That's all so
                           return to Monitor

!> START                  ;Starts all schemes
                           currently enabled

!>
```

A real time SCHEME can be disabled using the DISABLE command so that it will not be executed. The START command begins execution of all enabled SCHEMES. The STOP command stops execution of all enabled SCHEMES.

The EXECUTE command is used to start the execution of one background SCHEME. Background SCHEMES are executed once only unless they include a backward branch, in which case the part of the program between the branch point and the label to which it refers will be executed repeatedly until the program is halted at its current position and the computer is returned to the monitor level via a < CTRL/U > command. (N.B. Backward branching is not permitted in real time SCHEMES). If RESUME is then entered, the background program begins execution at the position it had previously reached. If ABORT is entered, the position which the background program had reached is disgarded, and using EXECUTE to start the program again will cause execution to commence at the start of the program.

For both background and real time SCHEMES, the command DELETE results in the deletion of the name SCHEME e.g.

```
! DELETE

NAME? FRED

NAME?  CTRL/U

!
```

DELETE may only be used when a STOP command has been entered (i.e. no SCHEMES are being executed). Any attempt to DELETE a TABLE being used by a SCHEME, enabled or disabled, the same error message is produced.

4.8 Editor

EDITOR is used to list and modify the values of variables and control parameters within a block. The variables of a new SCHEME created using the SCHEME command or the COPYED scheme will be flagged and displayed as "bad" (i.e. will not have a value) even if the scheme has been enabled and started unless the user sets that variable in EDITOR. This applies particularly to the parameters of block such as FIRST (i.e. the time constant, TC) or to the PID controller block (gain, K, integral time TI and derivative time TD). DDACS will, however, assign values to variables which it can so that it will be unnecessary to initialize or preset these variables prior to START. For example, if a scheme with the code listed on pages 10 and 11 is ENABLED and STARTED then the PID block called by the assignment statement:

```
OP:=PID(ER,IC,K,TI,TD)
```

will assign a value of "bad" to OP even though ER and IC have "good" or valid values until K,TI and TD have been assigned values using EDITOR.

EDITOR has four levels: SCHEME, LOOP, BLOCK and PARAMETER. At each level, EDITOR prints the appropriate prompt, e.g.

```
     !> EDITOR
NAME?> EXAMPLE1
LOOP?> 4
BLOCK?> MONITOR                ;See listing pagell
PAR?> 10                       User requests
                               display of current
                               value

  ST= 0.0000E+00
  SP= 3.6380E+01

 :
 :K =**********                ;K IS "BAD" (has
                               not been assigned
                               yet)

  TI=**********                ;So is TI
  TD=**********
  OP=**********                ;And, as a result,
                               so is OP.
PAR?> DV< CNTRL/A>             ;Enter to CHANGE
                               allowable parameters
    K=**********
VAL?> 1                        ;User changes K to
                               1.0
   TI=**********
VAL?> 2                        ;and so on
   TD=**********
VAL?> 0
PAR?> Entering 10 will verify that K,TI,TD
      and OP now have values,:Typing
      < CNTRL/U> will return to monitor level,
BLOCK?> < RTN>                 ;Return key returns
                               to monitor through
                               various
LOOP?> < RTN>                  ;Levels of EDITOR
NAME?> < RTN>
!>                             ;Back to Monitor
                                level
```

In summary, the reply 10 causes the present value of all Inputs and Outputs to the previously specified BLOCK to be printed. In the case where more than one BLOCK of the same type is included in a LOOP, successive BLOCKS of that type can be accessed by entering + in reply to the PAR?> prompt. If a variable has not been allocated a value it is printed out as a series of asterisks. A variable may be assigned a value by means of the DV<CTRL/A> command.

Some BLOCKS such as the PID BLOCK, have special control parameters such as GAIN, TI and TD and these may also be modified by entering the parameter name followed by < CTRL/A> .

If the user is in doubt as to what replies are permitted to the PAR?. prompt a? will cause all permitted replies to be listed.

4.9 Devices

This command allows the user to discover the addresses which are allocated in the range 000000-177776. Since it uses the hardware trap facility, the command is only available when the machine is in a STOPped state.

4.10 TABSWP

This command allows the user to change all references to a particular Table to another Table. Hence if two tables exist $A and $B then the effect is as follows

```
    !>TABSWP
    NAME>A       ;User enters TABLES to be
                  interchanged.
    NAME>B
    !>
```

If the block previously referencing $A is now listed it will read:

AAV($B,....)

4.11 Less Frequently Used Commands

Students are not allowed to use these commands under any circumstances.

4.11.1 TESTWD

This command has the effect of an implicit background job which tests a particular word in store for a change in value. It is used for testing DDACS software and should not normally be needed by the DDACS user.

In addition to the above mentioned monitor level commands, there are three others available.

4.11.2 KILL

This isolates the terminal from the computer. Re-entry to the monitor level is via <CTRL/U >.

4.11.3 RUN

```
    157600 program at address 157600 is
    executed
```

4.11.4 EDIT

This allows the examination and modification of the contents of any location in the computer memory. It is used for finding and correcting faults in the DDACS software and should not normally be needed by the DDACS user e.g.:

```
    !>EDIT
    >042432 displays contents at location
            042432 which may be changed
            in response to the prompt >
            or left unchanged by a <RETN>.
```

DESCRIPTION OF BLOCKS

5.1 General Remarks

The following description of the DDACS BLOCKS should be used in conjunction with the examples provided in the introductory section and the notes to accompany third and forth year practical experiments.

Conventions:

The abbreviation var means a variable of any type real, bad, integer or logical. The abbreviation varnum means that either a literal (actual number) or a variable may be substituted.

Variables may be of one or, at most, two alphanumeric characters, the first of which must be alphabetically [A,B,C,...,Z]. A variable is flagged 'bad' and will be printed as ******** if it has not been assigned a value (e.g. on the first pass through the block after START if not previously assigned a value) or if it would be outside the range of TABLE's scope.

Arithmetic Assignment statements use the compound symbol := e.g.:

A:=A+1

Logical assignment statements are written thus:

L==(OP >100)!(OP< 0)

where L is assigned true if the logical statement on the RHS is true. Variables may be used in arithmetic or logical statements interchangeably however:

A variable is considered true if it is greater than 0 and false if equal to or less than 0.

5.2 Arithmetic and Logical Operator Blocks

The usual operations are provided i.e.:

+ - / *

and parenthesis may be freely used. The compiler (parser) will accept the usual FORTRAN-like expressions but will insert additional parenthesis when LISTed back e.g.

IC:=RG-ER*(K+T/TI)

becomes IC:=RG-(ER*(K+(T/TI)))

nonetheless the first statement will be accepted as unambiguous. The usual FORTRAN rules of precedence apply i.e. *and / are evaluated first and + and - next and left to right for equal status operators.

In addition, logical operators are provided viz.:

>	(Greater than)
<	(Less than)
	(NOT operator, negation)
.	(Logical AND)
!	(INCLUSIVE OR)

Logical and arithmetic operators and functions (see 5.3 below) may be used in logical statements e.g.:

A==(B>C) ! (D> (ABS(X)*SQRT(Y)))

although care should obviously be exercised.

Non permissible arithmetic expressions such as A/B or SQRT(A) where A is negative and B zero are assigned as "bad" and execution will continue although such conditions may be detected in Editor.

5.3 Arithmetic Function Blocks

The present 32KW version provides SQRT, FINT, ABS, FINT and a selection of trigonometric and exponential functions. They have the form:

FUNCTION(varnum)

e.g.,

```
        A:=SQRT(X)*ABS(Y)
```

FINT truncates a floating point number to an integer

```
e.g.    X:FINT(Y)

     If  Y=1.23, then X = 1.
```

SQRT calculates square roots

```
e.g.    X :SQRT(Y)

     If  Y is negative, then X is bad
```

ABS calculates the absolute value of the varnum input.

SIGN outputs +1 if the varnum is positive, 0 if the varnum is zero, and -1 if the varnum is negative.

Trignometric Blocks are SIN, COS, TAN, ASIN, ACOS, ATAN.
EXP and LOG (natural) and LOG 10 and LOG 2 are also provided.
Further special purpose arithmetic functions are listed on page 24.

5.4 Timing Function Blocks

TIME(num) where num is an integer, sets the period at which a LOOP will be executed, e.g.:

TIME(3)
requires the clock handler or scheduler to execute all statements down to the next TIME statement every (2**3)*10 or 80 msec. Values of num of 1,2,3, 4, ... hence result in periods of 20, 40, 80, 160, ... msec up to a maximum of 327680 msec (or 9.10 hours) i.e. the maximum value of num is 15.

All schemes must start with a time statement and end with TIME(0). TIME(@) denotes a background scheme.

READTIME enables a real-time program to read the time rate of the LOOP which the block is running in:-

```
    READTIME(A)
```

Will only be accepted in a real time program - otherwise compiler error 207 is generated.

Set bad data on a START in A, otherwise A is a floating point number representing the LOOP rate,

i.e. 5.120 etc.

RTIM

```
Format:-
     var:=RTIM(num)
where num can be:-
     0-seconds
     1-minutes
     2-hours
     3-day
     4-month
     5-year
```

This BLOCK allows the user to access the current time and refer to it as a floating point number e.g.

```
     X:=RTIM(3)
```

In this example, X is the day of the month.

STIM

```
Format:-
     STIM(num,var)
```

where num can be as in RTIM.
The STIM BLOCK, which can only be used in a background SCHEME, allows the user to set the current time e.g.

```
     STIM(2,Y)
```

In this example, the hour is set by the value of the variable Y.

PAUSE

```
Format:-
     PAUSE(varnum)
e.g. PAUSE(N)
```

The PAUSE BLOCK causes a delay in the execution of a background scheme. The argument, N, is considered to be unsigned hence negative numbers give long delays. The delay length is approximately N*1mS (time taken to execute real time SCHEMES). The PAUSE(0) is the DDACS "no-operation" instruction.

5.5 Input-Output

AAV (Digital/Analogue Converter)

```
     Format:-
     AAV (tablename, var, channel
     number, constraint enable)

e.g.
     AAV($TI< A< 0,1)
```

This block outputs analogue values via the AAV11 digital to analogue converter accessing channels 0 to 4 [via buffers at the following locations:

```
     channel 0 : 170440
             1 : 170442
             2 : 170444
             3 : 170446]
```

Hence the channel number must be in the range 0 to 3.

Table name is the name of a table which converts the value in the buffer to the required range for the analogue output, e.g. the value of the variable A above is to be converted via table T1 and output through channel 0.

The Constraint enable varnum (which must be integer) indicates whether or not a constraint condition is to be imposed on the loop when the input variable is bad or out of range of the table. If true, the constraint is set (true>0), if false (<0) no constraint is set.

Note that whether the constraint varnum is set or not, a bad or out of range value will not be written to the converter, i.e. the effect is to 'freeze' the analogue output voltage at its last value. Values of input A which would cause the converted value to exceed the maximum voltage or fall below the minimum voltage (e.g. if an incorrect table is used) will result in a voltage of +5.12 or -5.12 respectively.

READBACK

It is remotely possible that because data is held in two words the result of a calculation could be half formed when another program interrupts it to read the data. This is automatically protected when a result is written out by raising or dropping the priority. However, if a slower loop is reading data from a faster loop then potentially the faster loop itself could interrupt causing a change of its own data. If the system is heavily loaded it is recommended that the READBACK block be used. This enables protected feedback of variables which are derived from faster loops.

```
READBACK(input variable, output
variable, input variable, output
variable ... etc.)

READBACK (A,A1, B,B1 C,C1...)
```

causes the value of A to be put in A1 etc. A would have been calculated in a faster loop. All the output variables are filled with bad data on a START.

ADV (Analogue/Digital Converter)

This BLOCK reads any number of the 16 analogue inputs channels to the ADV11 analogue to digital converter.

Format:-

```
ADV(tablename, channel no.,
variable name, channel no,
variable name, etc.)
```

Example:

```
ADV($T1,0,A,1,B,E)
```

reads channels 0, 1 and 3 of the ADV converting the voltages through tab T1 and assigning the values to variables A, B and E.

Channel numbers must be between 0 and 15 and must be integers (not variables). There may be, up to the length of the line, any number of channel number variable pairs and the block will read the next channel immediately it has read the present one.

The outputs will be bad if the input voltage is out of range of the table and also in start mode.

BREAD ("Bits READ", 16 bit digital 1/0)

This BLOCK reads any number of the 16 individual bits (numbered 0 to 15) of the DRV11 parallel interface.

Format:-

```
BREAD(bit no.,var,bit no.,var, etc)

e.g.BREAD(0,A,3,B,14,C)
```

where bit number must be an integer (not a variable) and a bit number/ variable pairs may be listed to the end of the line. Each digital input state is assigned to the logical variable named. If the input bit is set then the variable is made true, if clear it is false corresponding to 1.0 and 0.respectively.

In 'start' mode all output variables are set bad.

BWRITE ("Bits WRITE"; 16 bit digital 1/0)

This BLOCK converts a logical variable to a set or clear (or alternating set and clear) at the DRV11 16 bit digital parallel 1/0 terminals.

Format:

```
BWRITE(channel no.,var,inum1,inum2,inum3,
      inum4)

e.g. BWRITE(3,X,-1,1,0,-1)
```

where the inum are four integer numbers or 'output state flags" used as described below, the channel number represents the bit position to be set as output (0 to 15) and is an integer not a variable,

and var is the name of the variable (X in the example) which is to determine the state of the output in conjunction with the output state flags.

If variable is NEGATIVE (<0) the first output state flag controls the value of the digital output as below i.e. if

```
inum1 = 1 : set digital output
        0 : clear digital output
       -1 : set and clear (i.e. flash if
            connected to an LED) at the loop
            rate
```

if var is ZERO (=0) inum2 controls as the output state,
if var is POSITIVE (>0) inum3 controls
and if var is BAD, inum4 controls the state of the output.

In the example, when X is negative or bad a light connected to channel 3 will flash, when X is zero the output will be set (steady light)

and

when X is positive the output will be clear.

The greatest use of BWRITE is in signalling alarm and fault conditions to an operator panel.

WRITNO

Format:-
WRITNO (format number, varnum)
where the format number is coded as follows:-
0 free format integer
1 fixed format integer
2 fixed format floating point
3 free format floating point

WRITNO is only used in background SCHEMES. It causes the varnum to be printed on the terminal e.g.

WRITNO(3,X) causes the value of X to be printed e.g. 15.4

TEXT

Format:-

TEXT(string)
where string is a non-zero sequence of characters surrounded by double quotes (")

e.g.

TEXT("THIS MESSAGE IS PRINTED")

TEXT is only used in background SCHEMES and causes the string to be printed on the terminal.

READNO

Format:-

READNO(string,var)

e.g.

READNO("ENTER VALUE OF A :-",A)

READNO is only used in background SCHEMES and causes the string to be output to the terminal and assign the value of the typed reply to the variable.

PRINTCH
Format:-
PRINTCH(varnum,varnum.....)
PRINTCH is only used in background SCHEMES. It causes one or more characters to be printed on the terminal. The value of the variable or number is converted to octal and truncated to a byte which is treated as ASC11 by the terminal e.g.
PRINTCH(13,10) causes a <CR>, <LF> to be printed.

READA (Foreground or background schemes)

Format:-
READA(table name, rows,columns,var,flag)
e.g.READ($FRED,24,X,FL)
The element of the second row of the fourth column is read and given the name X. If X is true FL is 1, if X is bad FL is bad.

WRITA (Foreground or background)

Format:
WRITA(tablename,rows,columns,varnum,flag)
e.g. WRITA(FRED,2,4,Y,FL)
The element of the second row of the fourth column of the array FRED is assigned to the value of Y. If Y is true, FL is 1, if Y is false, FL
is
0 and if Y is bad FL is bad.

MONITOR

Format:-
MONITOR(var,var,var......)

e.g. MONITOR(A,B,C......)

The purpose of this block is to allow the user to access (from the console through EDITOR) several different variables without having to specify the BLOCK in which each of the occurs. In the example A,B,C,.... may occur in different blocks scattered through a scheme. They are all conveniently gathered for access in the MONITOR block.

5.6 MODE AND PROGRAM FLOW CONTROL

AMS

Format:-

AMS(varnum)

This BLOCK indicates to the SCHEME containing a PID, INT or FIRST block that the block should be in "auto" if the varnum is true, and in "manual" if it is false. If there is no AMS block in a SCHEME, the SCHEME will function in normal mode i.e. "Auto" (P.5) unless a constraint occurs. Such a Constraint can be set from the CONSTRAINT block (see below) or from and AAV block with "Constraint Enable" flag set. A scheme which is in "manual" sets PID,INT and DELTA blocks to initialise mode such that their outputs are continually equal to their initial condition input. In "auto", the normal functions of these blocks occurs although they may be forced to initialise mode by a CONSTRAINT block in the loop.

CONSTRAINT

Format:-
CONSTRAINT(varnum)
This BLOCK sets a constraint in the LOOP in which it appears if the varnum is true. Its effect is to set the LOOP to INITIALISE mode then to NORMAL, continuing this until the constraint condition has been removed (see p).

GETMODE

Format:-
GETMODE(var)
e.g. GETMODE(A)
This BLOCK assigns a value to the variable depending on what mode the LOOP in which it occurs is. If the LOOP is in start mode, the variable becomes -1. If it is in initialise mode, the variable becomes 0, and if it is in normal mode, the variable becomes 1.

SETMODE

SETMODE(varnum)
SETMODE is only used in background SCHEMES. Its purpose is to tell the background SCHEME what mode it is in. If the varnum is negative the SCHEME runs in start mode. If the varnum is 0 the SCHEME runs in initialise mode, and if the varnum is positive the SCHEME runs in normal mode.

GOTO

Format:-
GOTO(label)

e.g. GOTO(X1:)
This causes an unconditional branch to the line of the program which starts with the label name. In realtime SCHEMES, only branching forward is allowed. When the SCHEME is compiled, the actual name used (or the label is not remembered). When the SCHEME is printed out by a LIST command, the first label to appear is referred to as L10:, the second one L11: etc.

TIMESET

Format:-
TIMESET(n)
Where n=1 to 15
TIMESET is only used in background SCHEMES. Its purpose is to enable BLOCKS which utilise the loop rate in their calculations to have a "pseudo loop rate" given to them.

SWITCH

FORMAT:-
SWITCH(VAR,LABEL-LABEL)
E.G. SWITCH(A,L1:L2:)
If A is true, go to L1:
If A is false, carry on
If A is bad, go to L2:

The SWITCH BLOCK maintains within it a flag to indicate to itself which branch it took on the previous operation. If there is no change in the branch then it operates normally. However, upon any change the SWITCH BLOCK causes the remaining BLOCKS in that LOOP to initialise, and then on the next timestep initialises all the BLOCKS in the LOOP.

IF

Format:-
IF(var,label,label)

The IF BLOCK is identical to the SWITCH BLOCK, except that when a new branch is taken it does not cause initialisations.

LOOP

Label:
LOOP(var,label:)
e.g. L:A=A+1
LOOP(X,L:)
LOOP is only used in background SCHEMES. It executes the part of the SCHEME starting at L: a given number of times X. It should be noted that the loop is executed onece even if the variable X is bad, negative or zero.

INT (Integrator Block)

Format:-
var:=INT(var,var)
e.g. Y:=INT(X.IC)

In NORMAL mode this BLOCK produces an output approximating to an integrator:-
Y(t) = T*X(t) + Y(t-1)
where t is the sampling instant,
where t is the last sample instant
and T is the sampling interval
(reciprocal of loop rate).
In INITIALISE mode, the output is set equal to the initial condition.

DELTA (Differentiator Block)

Format:-

var:=DELTA(var)
e.g. Y:=DELTA(X)
In NORMAL mode, this BLOCK produces a Euler approximation to a differentiator

Y(t) = [X(t)-X(t-1)]/T

In "initialise" mode, the "history", X(t-1) of the input is set to the present input, X(t) and thus the output is zero.

[Note: by combining INT, DELTA and the arithmetic blocks in various ways special purpose PID controllers may be designed if the PID controller provided is not satisfactory].

FIRST (First Order Lag Block)

Format:-

var:=FIRST(var,varnum)
e.g. Y:=FIRST(X,TC)
In NORMAL mode the BLOCK produces an approximation to a continuous first order lag or filter:-

Y(t) = (1.-T/TC) * Y(t-1) + T*X(t)/TC

In INITIALISE mode, the output Y(t) equals the input X(t). If the time constant TC is less than the loop rate the BLOCK functions in initialise mode.

AHYS (Antihysteresis)

FORMAT:-
var:=AHYS(var,varnum)
E.G. Y:=AHYS(X,H)
In normal mode the BLOCK provides an output which is designed to eliminate hysteresis by detecting changes in sign of the rate of change input.

Y(t)=X(t) if sign[X(t)-X(t-1)]*sign [X(t-1)-X(t-2)]
Y(t)=X(t)+H if sign[X(t)-X(t-1)] is not equal to sign[X(t-1)-X(t-2)]
and sign of[X(t)-X(t-1)]is positive;
Y(t)=X(t)-H if sign X(t)-X(T-1) is not equal to sign [X(t-1)-X(t-2)]
and sign of [X(t)-X(t-1)] is negative.

In initialise mode:
Y(t)=X(t).

STIC (Stiction)

Format:-
var:-STIC(var,varnum)
e.g. Y:=STIC(X,DB)
In normal mode the BLOCK's output follows a staircase function in response to variations in the input, the step size being equal to the stiction DB.

Y(t)=X(t) if X(t)-X(tr) is greater than or equal to DB

Y(t)=Y(tr) if X(t)-X(tr)is less than DB

where tr is the time of the previous successful output

In initialise mode it sets its output Y(t)=inputX(t) and sets its stored value of the previous successful input X(tr)=X(t)

<u>DBAND</u>

This BLOCK allows introduction of a dead-band into the output to an actuator to eliminate hunting.

Format:-
var:=DBAND(var,varnum)
e.g. Y:=DBAND(X,DB)

The BLOCK functions is the same in all modes.

Y(t)=0 if modulus of X(t) is less than DB

Y(t)=X(t)-DB if modulus of X(t) is greater than or equal to DB and X(t) is positive

Y(t)=X(t)+DB if modulus of X(t) is greater than or equal to DB and X(t) is negative.

<u>RAMP</u>

Format:-

VAR:=RAMP(var,var,varnum)
e.g. Y:=RAMP(X,IC,RT)

The building BLOCK limits the rate of change of the output variable in response to changes in the input variable. In normal mode the BLOCK outputs

Y(t)=X(t) if (X(t)-X(t-1))/T is less or equal to RT

Y(t)=X(t-1)+RT if (X(t)-X(t-1)/T is greater than RT.

In initialise mode Y (t)=IC(t)

5.8 SPECIAL PURPOSE ARITHMETIC AND LOGICAL FUNCTIONS

<u>TRACK</u>

This block provides an output which tracks an input or holds it last value depending on the value of a switch.

Format:-
var:=TRACK(var,var)
e.g. A:=TRACK(B,C)
A=B if C is true (greater than zero)
A=A if C is false (less than or equal to zero)
A is bad if C is bad or C is true and B is bad.

<u>AVE</u>

This BLOCK provides an instantaneous average of a number of input variables, for example when averaging a number of transducer outputs ignoring Bad inputs.

Format:-
var:=AVE(var,var,var.....)
e.g. A:=AVE(B<C<D)
A becomes equal to the average of all the good variables. Thus if B, C and D are all good, A=(B+C+D)/3. However if say C is bad then A=(B+D)/2

<u>EAVE</u>

As for AVE Block except that all imputs must be good.

Format:-
var:=EAVE(var,var,var......)
The LHS becomes equal to the average of all good variables. If any are bad, then the LHS is bad.

<u>MIN</u>

Format:-
var:=MIN(var,var,var......)
e.g. A:=MIN(B,C,D)
A becomes equal to the minimum of all the good variables. Thus if B=2,C is Bad and D=-2, then A=2.

<u>AMIN</u>

Format:-
var:=AMIN(var,var,var......)
e.g. A:=AMIN(B,C,D)
A becomes equal to the minimum of the absolute values of all the good variables. Thus if B=2, C=1 and D=2, then A=1.

<u>EMIN</u>

Format:-
var:=EMIN(var,var,var.....)
The LHS becomes equal to the minimum of the variables. If any variable is bad then the LHS is bad.

<u>EAMIN</u>

Format:-
var:=EAMIN(var,var,var......)
The LHS becomes equal to the minimum of the absolute values of the variables. If any variable is bad, then the LHS is bad.

<u>MAX</u>

Format:-
var:=MAX(var,var,var.......)
The LHS becomes equal to the maximum of all the good variables. Thus if B=2, C=1 and D=-3, A=2.

<u>AMAX</u>

Format:-
var:=AMAX(var,var,var......)
e.g. A=AMAX(B,C,D,)
A becomes equal to the maximum of the absolute values of all the good variables. Thus if B=2, C=1 and D=3, A=-3.

<u>EMAX</u>

Format:-
var:=EMAX(var,var,var.....)
The LHS becomes equal to the maximum of the variables. If any variable is bad.

6. DDACS COMPILER ERROR CODES

If the rules of the language are violated then the computer outputs an error message. The input typed by the user passes through two phases. Scanning and parsing. The scanner checks that the input is composed of legal characters and forms in the DDACS control language. Errors coded with the letter 'S' indicate that the scanner has rejected the input and hence it is likely to be an obvious error.

S 000002	argument string not terminated by a bracket
S 000003	illegal argument in an argument string number when it should be)
S 000005	illegal character
S 000401	Table named which doesn't exist
S 000404	Not a library word(i.e., building BLOCK not a system)
S 000405	Integer too large
S 000406	Incorrect number format

If an error code has the letter 'P' in front then this indicates a syntatically correct statement of DDACS -MCS has been typed which the parser has decided is meaningless or ambiguous.

P 000002	Extra characters following a statement
P 000003	Illegal character string
P 000005	Not a statement following a TIME statement
P 000007	Not a statement
P 000010	Not on assignment (i.e.:=)
P 000011	Not a valid arithmetic expression
P 000020	Mismatched brackets
P 000200	Not a variable when expected in BLOCK arguments.
P 000202	Not a number or variable when expected in BLOCK arguments
P 000203	Not an integer when expected in BLOCK arguments

<u>PARSER TABLE CREATION ERRORS</u>

P 000300 Co-ordinates not terminated correctly

P 000301 Not a number where X co-ordinate expected

P 000302 Not a number when a Y co-ordinate expected

P 000401 TIME BLOCK SPECIFICATION ERRORS

a) No closing bracket
b) Not a +ve argument
c) Not an integer
d) Not a legal LOOP period
e) Greater than or equal to previous TIME statement

P 000402 LABEL ERROR

a) Unsatisfied label when TIME(0) statement reached.
b) Attempt to take a backwards label

P 000403 Attempt to write to a variable in a slower LOOP

P 000404 Attempt to insert a label before a reference is made to it

P 000405 Multiple label

P 000406 Floating point typed where integer expected

P 000500 Table Creation Error: Successive X values the same

OTHER "SYSTEM" ERROR CODES:

E 077777	No room left in DIRECTORY for new SCHEME or TABLE
F 177775	No room left for Compilation

7. FORMATTING DDACS FLOPPY DISCS

This section describes the procedure for formatting DDACS floppy discs, such that they can be used with the DUMPDX command.

1. Load a floppy disc which is already formatted.

2. Make the following alterations by typing EDIT followed by the address e.g. for a 16 KW memory where the bootstrap loader is located at 077600 (at top of core - 200):

```
!>EDIT
>077636
      077636  000002>000000<RET>
      <CTRL/D>
```

Continue with the other locations, re-entering EDIT each time:

```
077746 005007>000000
077776  XXXX007>000400
```

For a 32 KW machine the first three digits of each address will be 157 instead of 077.

3. Run the dump read program at 077600. (N.B Short execution time - finishes successfully at 077750 this reads the dump program from the formatted disk.

4. Load the floppy disk which requires the formatting information into DXO.

5. Run the dump write program at 077606. (N.B. Also short run-time, same finish address). Note that the ODT instruction 0776066 will be used since 3. results in program leaving DDACS.

6. Process complete, the disk may now be used to DUMP the DDACS system at the end of a session.

APPENDIX B

B.2 SDK-86 SOFTWARE LISTING - STAGE 1

```
ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE FLOWLOOP
OBJECT MODULE PLACED IN :F1:FLOW.OBJ
COMPILER INVOKED BY:  :F2:PLM86 :F1:FLOW.SRC SMALL


              /**********************************************
               *  FLOW CONTROL LOOP PROGRAM.  RUN          *
               *  ON THE SDK-86 UNDER KEYBOARD CONTROL     *
               **********************************************/
  1           FLOWLOOP:   DO;

              /* INTER-PROCESSOR PARAMATERS */
  2   1           DECLARE
                      (STATUS,SETPOINT,FANSPEED,FLOW,
                       INITIAL$CONDITION) WORD;

              /* KEYBOARD AND DISPLAY DECLARATIONS */
  3   1           DECLARE KB$DIGIT$BUFFER(4) BYTE;
  4   1           DECLARE KB$B$PTR BYTE;
  5   1           DECLARE
                      KB$STATUS$PORT      LITERALLY '0FFEAH',
                      KB$DATA$PORT        LITERALLY '0FFE8H';
  6   1           DECLARE
                      KB$PERIOD       LITERALLY '10H',
                      KB$COMMA        LITERALLY '11H',
                      KB$MINUS        LITERALLY '12H',
                      KB$PLUS         LITERALLY '13H',
                      KB$COLON        LITERALLY '14H';
  7   1           DECLARE
                      DISPLAY$DIGIT(10) BYTE DATA
                      (3FH,06H,5BH,4FH,66H,6DH,7DH,07H,        /* '0,1..7' */
                      7FH,6FH),                        /* '8,9' */
                      DISPLAY$A       LITERALLY '0DFH',   /* AUTO */
                      DISPLAY$M       LITERALLY '0D5H',   /* MANUAL */
                      DISPLAY$S       LITERALLY '0EDH',   /* SETPOINT */
                      DISPLAY$R       LITERALLY '0D0H',   /* REG INPUT */
                      DISPLAY$C       LITERALLY '0D8H';   /* CHANGE */

              /* ADC AND DAC DECLARATIONS */
  8   1           DECLARE
                      ADC$$PORT$0     LITERALLY '0FF00H', /* A/D BASE */
                      DAC$PORT$0      LITERALLY '0FF20H'; /* D/A BASE */

              /* IPI (INTER-PROCESSOR INTERFACE) DECLARATIONS */
  9   1           DECLARE
                      CSR$ADDR        LITERALLY '0FF40H',
                      DBR$ADDR        LITERALLY '0FF40H',
                      DOWN$TR$CHO     LITERALLY '0FF42H';

              /* KEYBOARD PARSER DECLARATIONS */
 10   1           DECLARE
                      PARSER$TABLE(5) STRUCTURE (ACTION(6)
                      BYTE,NEXT$STATE(6) BYTE,FUNCTION(6)
                      BYTE) DATA
                      (1,0,0,0,0,0,   1,0,0,0,0,0,
                       3,0,0,0,0,0,                    /* STATE 0 */
                       0,2,3,7,0,0,   0,1,2,1,0,0,
                       2,6,3,0,0,0,                  /* STATE 1 */
                       0,1,4,5,6,7,   0,1,4,4,3,2,
                       2,3,4,5,1,0,                  /* STATE 2 */
                       0,4,5,6,3,0,   0,4,4,3,2,0,
                       2,4,5,1,0,0,                  /* STATE 3 */
                       0,4,5,3,0,0,   0,4,4,2,0,0,
                       2,4,5,0,0,0),                 /* STATE 4 */

                      (ACTION,STATE,FUNCTION,NUMBER) BYTE;

               /* UTILITY DECLARATIONS */
  11   1             DECLARE
                       (CHAR,NOMATCH,I,MAN$AUTO$SWITCHED) BYTE,
                       FOREVER             LITERALLY 'OFFH';

               /**************************************
                *   BASIC I/O PROCEDURES FOR ADC,   *
                *   KEYBOARD AND DISPLAY.          *
                **************************************/

  12   1       ADC:
               /* THIS PROCEDURE PERFORMS AN A/D CONVERSION AND RETURNS
                  THIS VALUE;
                       DIFF = 0    SINGLE ENDED INPUT
                       DIFF = 1    DIFFERENTIAL INPUT. */

                     PROCEDURE (CHANNEL$NO,DIFF) WORD;
  13   2             DECLARE (CHANNEL$NO,DIFF) BYTE;

               /* START CONVERSION OF CHOSEN CHANNEL */
  14   2             CHANNEL$NO = SHL(CHANNEL$NO,1) OR DIFF;
  15   2             OUTPUT(ADC$PORT$0 + CHANNEL$NO) = 0;

               /* FETCH CONVERTED VALUE */
  16   2             DO WHILE INWORD(ADC$PORT$0);
  17   3             END;
  18   2             RETURN SHR(INWORD(ADC$PORT$0),1) AND OFFFH;
  19   2       END ADC;


  20   1       DAC:
               /* THIS PROCEDURE SENDS A WORD TO A SPECIFIED D/A CHANNEL */

                     PROCEDURE (CHANNEL$NO,VALUE);
  21   2             DECLARE CHANNEL$NO BYTE, VALUE WORD;

  22   2             CHANNEL$NO = SHL(CHANNEL$NO,1);
  23   2             OUTWORD(DAC$PORT$0 + CHANNEL$NO) = -1-VALUE;
  24   2       END DAC;


  25   1       KB$GET$CHAR:
               /* THIS PROCEDURE RETURNS THE NEXT KEYBOARD CHARACTER */

                     PROCEDURE BYTE;
```

```
/* WAIT TILL CHARACTER IS PRESENT */
    DO WHILE
        (INPUT(KB$STATUS$PORT) AND 0FH) = 0;
    END;

/*FETCH CHARACTER */
    OUTPUT(KB$STATUS$PORT) = 40H;
    RETURN INPUT(KB$DATA$PORT);
END KB$GET$CHAR;


DISPLAY$CHAR:
/* THIS PROCEDURE DISPLAYS A CHARACTER ON A SPECIFIED
   7 SEGMENT LED */

    PROCEDURE (CODE,POSITION);
    DECLARE (CODE,POSITION) BYTE;

/* WAIT TILL DISPLAY IS AVAILABLE */
    DO WHILE
        (INPUT(KB$STATUS$PORT) AND 80H) <> 0;
    END;

/* DISPLAY CHARACTER */
    OUTPUT(KB$STATUS$PORT) = POSITION OR 80H;
    OUTPUT(KB$DATA$PORT) = CODE;
    RETURN;
END DISPLAY$CHAR;


DISPLAY$NUM:
/* THIS PROCEDURE DISPLAYS A 4 BYTE BCD LIST AS A
   4 DIGIT DECIMAL NUMBER (XX.XX) ON THE RIGHT
   HALF OF THE DISPLAY */

    PROCEDURE (PTR);
    DECLARE PTR POINTER,
        LIST BASED PTR(1) BYTE;
    CALL DISPLAY$CHAR(DISPLAY$DIGIT(LIST(0)),0);
    CALL DISPLAY$CHAR(DISPLAY$DIGIT(LIST(1)),1);
    CALL DISPLAY$CHAR(DISPLAY$DIGIT(LIST(2)) OR 80H,2); /* DEC PT */
    CALL DISPLAY$CHAR(DISPLAY$DIGIT(LIST(3)),3);
    RETURN;
END DISPLAY$NUM;


/*********************************************
 *  PROCEDURES TO PERFORM ACTIONS 0 TO     *
 *  6 REFERENCED BY THE KEYBOARD PARSER    *
 *********************************************/


ACTION$0:
/* THIS PROCEDURE BLANKS THE SDK-86 DISPLAY */

    PROCEDURE;
 48   2          DECLARE I BYTE;

             /* WAIT TILL DISPLAY READY */
 49   2          DO WHILE
                     (INPUT(KB$STATUS$PORT) AND 80H) <> 0;
 50   3          END;

             /* BLANK */
 51   2          DO I = 0 TO 7;
 52   3           CALL DISPLAY$CHAR(0,I);
 53   3          END;
 54   2          RETURN;
 55   2      END ACTION$0;



 56   1      PRCNT:
             /* THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. IT MULTIPLIES
                THE INPUT VALUE BY 2.71H    */
                 PROCEDURE (VALUE) WORD EXTERNAL;
 57   2          DECLARE VALUE WORD;
 58   2      END PRCNT;


 59   1      SPAN:
             /* THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. IT DIVIDES
                THE INPUT VALUE BY 2.71H   */
                 PROCEDURE (VALUE) WORD EXTERNAL;
 60   2          DECLARE VALUE WORD;
 61   2      END SPAN;


 62   1      ACTION$1:
             /* THIS PROCEDURE RESPONDS TO THE VALUE OF STATUS.
                 IF AUTO...   THE SETPOINT IS DISPLAYED AS
                         A PERCENTAGE OF SPAN.
                 IF MANUAL...    THE FAN SPEED IS DISPLAYED AS
                         A PERCENTAGE OF SPAN.

                IN EACH CASE THE MODE IS DISPLAYED.    */

                 PROCEDURE;
 63   2          DECLARE VALUE WORD,BCD(4) BYTE;

             /* DETERMINE IF AUTO OR MANUAL AND DISPLAY MODE */
 64   2          IF STATUS THEN
 65   2          DO;                            /* AUTO */
 66   3           CALL DISPLAY$CHAR(DISPLAY$A,7);
 67   3           CALL DISPLAY$CHAR(DISPLAY$S,4);
 68   3           VALUE = SETPOINT;
 69   3          END;
                 ELSE
 70   2          DO;                            /* MANUAL */
 71   3           CALL DISPLAY$CHAR(DISPLAY$M,7);
 72   3           CALL DISPLAY$CHAR(DISPLAY$R,4);
 73   3           VALUE = FAN$SPEED;
 74   3          END;
```

```
                /* SCALE VALUE TO GIVE PERCENTAGE SPAN */
 75   2             VALUE = PRCNT(VALUE);

                /* PERFORM BCD CONVERSION */
 76   2             BCD(3) = VALUE/1000;
 77   2             VALUE = VALUE MOD 1000;
 78   2             BCD(2) = VALUE/100;
 79   2             VALUE = VALUE MOD 100;
 80   2             BCD(1) = VALUE/10;
 81   2             BCD(0) = VALUE MOD 10;

                /* OUTPUT TO DISPLAY */
 82   2             CALL DISPLAY$NUM(@BCD);
 83   2             RETURN;
 84   2         END ACTION$1;



 85   1         ACTION$2:
                /* THIS PROCEDURE CHANGES THE AUTO/MANUAL STATUS AND
                   DISPLAYS AS IN ACTION 1.  BUMPLESS TRANSFER IS ACHEIVED.    */

                    PROCEDURE;
 86   2             IF STATUS THEN

                /* AUTO TO MANUAL TRANSFER */
 87   2             DO;
 88   3              STATUS = 0;
 89   3              CALL ACTION$1;                      /* DISPLAY */
 90   3             END;

                /* MANUAL TO AUTO TRANSFER */
                    ELSE
 91   2             DO;
 92   3              DISABLE;
 93   3              INITIAL$CONDITION = FAN$SPEED;
 94   3              SET$POINT = ADC(0,0);
 95   3              STATUS = 0FFFFH;
 96   3              MAN$AUTO$SWITCHED = 1;
 97   3              ENABLE;
 98   3              CALL ACTION$1;                      /* DISPLAY */
 99   3             END;
100   2             RETURN;
101   2         END ACTION$2;



102   1         ACTION$3:
                /* THIS PROCEDURE DISPLAYS THE KEYBOARD DIGIT
                   BUFFER (BCD) AND RESETS THE BUFFER POINTER */

                    PROCEDURE;

                /* RESET BUFFER POINTER */
103   2             KB$B$PTR = 3;
```

```
              /* DISPLAY BUFFER CONTENTS */
 104   2          CALL DISPLAY$NUM(@KB$DIGIT$BUFFER);

              /* INDICATE DISPLAY MODE */
 105   2          CALL DISPLAY$CHAR(DISPLAY$C,4);
 106   2          RETURN;
 107   2      END ACTION$3;



 108   1      SCALED$BCD$BIN:
              /* THIS PROCEDURE CONVERTS THE CONTENTS OF A 4 BYTE
                 BCD LIST INTO A BINARY WORD.  THIS IS SCALED SO
                 THAT PERCENTAGE SPAN BECOMES ACTUAL SPAN.   */

                  PROCEDURE (PTR) WORD;
 109   2          DECLARE PTR POINTER,
                      LIST BASED PTR(1) BYTE,
                      VALUE WORD;

              /* CONVERSION */
 110   2          VALUE = 1000*LIST(3) + 100*LIST(2)
                      + 10*LIST(1) + LIST(0);

              /* SCALE */
 111   2          VALUE = SPAN(VALUE);

 112   2      RETURN VALUE;
 113   2      END SCALED$BCD$BIN;



 114   1      ACTION$4:
              /* THIS PROCEDURE CONVERTS THE KEYBOARD DIGIT BUFFER
                 TO A BINARY, SCALED WORD.  THIS IS THEN SUBTRACTED FROM,
                  1. SETPOINT IF AUTO STATUS,
                  2. FAN$SPEED IF MANUAL STATUS.

                 IN EITHER CASE THE MINIMUM SETPOINT/FAN$SPEED = 0.
                 THE RESULT IS DISPLAYED AS IN ACTION 1 AND UNDER
                 MANUAL STATUS THE NEW FAN$SPEED IS SENT TO
                 D/A CHANNEL 0.     */

                  PROCEDURE;
 115   2          DECLARE ALTER WORD;

              /* CONVERT TO SCALED BINARY */
 116   2          ALTER = SCALED$BCD$BIN(@KB$DIGIT$BUFFER);

              /* AUTO STATUS */
 117   2          IF STATUS THEN
 118   2          DO;
 119   3           IF ALTER < SETPOINT THEN
 120   3           SETPOINT = SETPOINT - ALTER;
 121   3           ELSE SETPOINT = 0;
 122   3          END;
```

```
                /* MANUAL STATUS */
                    ELSE
 123   2            DO;
 124   3             IF ALTER < FAN$SPEED THEN
 125   3             FAN$SPEED = FAN$SPEED - ALTER;
 126   3             ELSE FAN$SPEED = 0;
 127   3             CALL DAC(0,FAN$SPEED);
 128   3            END;

                /* DISPLAY NEW SETPOINT/FAN$SPEED */
 129   2            CALL ACTION$1;
 130   2            RETURN;
 131   2        END ACTION$4;



 132   1        ACTION$5:
                /* THIS PROCEDURE IS THE SAME AS ACTION 4 EXCEPT THAT
                   ADDITION IS PERFORMED RATHER THAN SUBTRACTION.
                   THE MAXIMUM SETPOINT/FAN$SPEED = 0FFFH.    */

                    PROCEDURE;
 133   2            DECLARE ALTER WORD;

                /* CONVERT TO SCALED BINARY */
 134   2            ALTER = SCALED$BCD$BIN(@KB$DIGIT$BUFFER);

                /* AUTO STATUS */
 135   2            IF STATUS THEN
 136   2            DO;
 137   3             IF (0FFFH - SETPOINT) > ALTER THEN
 138   3             SETPOINT = SETPOINT + ALTER;
 139   3             ELSE SETPOINT = 0FFFH;
 140   3            END;

                /* MANUAL STATUS */
                    ELSE
 141   2            DO;
 142   3             IF (0FFFH - FAN$SPEED) > ALTER THEN
 143   3             FAN$SPEED = FAN$SPEED + ALTER;
 144   3             ELSE FAN$SPEED = 0FFFH;
 145   3             CALL DAC(0,FAN$SPEED);
 146   3            END;

                /* DISPLAY NEW SETPOINT/FAN$SPEED */
 147   2            CALL ACTION$1;
 148   2            RETURN;
 149   2        END ACTION$5;



 150   1        ACTION$6:
                /* THIS PROCEDURE SAVES A DIGIT IN THE KEYBOARD
                   DIGIT BUFFER AND DISPLAYS THIS BUFFER.  THE BUFFER
                   POINTER IS MOVED READY FOR THE NEXT DIGIT.    */

                    PROCEDURE;
```

```
151   2          DECLARE I BYTE;

152   2          IF KB$B$PTR = 3 THEN                   /* IF RESET */
153   2          DO I = 0 TO 3;
154   3           KB$DIGIT$BUFFER(I) = 0;               /* CLEAR */
155   3          END;

            /* SAVE NEXT KEYBOARD DIGIT */
156   2          KB$DIGIT$BUFFER(KB$B$PTR) = NUMBER;

            /* DISPLAY BUFFER */
157   2          CALL DISPLAY$NUM(@KB$DIGIT$BUFFER);

            /* MOVE POINTER */
158   2          KB$B$PTR = (KB$B$PTR - 1) AND 3;
159   2          RETURN;
160   2     END ACTION$6;



            /* SCANS INTER-PROCESSOR INTERFACE AND ACTS
               UPON COMMANDS RECEIVED BY THE PDP */

161   1     PDP$INT:

                PROCEDURE;
162   2         DECLARE CSR BYTE;

            /* DETERMINE CHANNEL NUMBER OF UP TRANSFER */
163   2         CSR = INPUT(CSR$ADDR);
164   2         CSR = SHR(CSR,1) AND 0FH;

            /* ENSURE THAT CHANNEL NUMBER IS WITHIN RANGE */
165   2         IF CSR > 2 THEN RETURN;

167   2         DO CASE CSR;

            /* CSR = 0...FLOW REQUIRED */
168   3          DO;
169   4           FLOW = ADC(0,0);
170   4           OUTWORD(DBR$ADDR) = FLOW;

            /* ALSO UPDATE FAN$SPEED */
171   4           IF MAN$AUTO$SWITCHED THEN
172   4           MAN$AUTO$SWITCHED = 0;
173   4           ELSE FAN$SPEED = INWORD(DOWN$TR$CH0);
174   4           CALL DAC(0,FAN$SPEED);
175   4          END;

            /* CSR = 1...STATUS REQUIRED */
176   3          OUTWORD(DBR$ADDR) = STATUS;

            /* CSR = 2...INITIAL CONDITION REQUIRED */
177   3          OUTWORD(DBR$ADDR) = INITIAL$CONDITION;

178   3         END;
179   2         RETURN;
```

```
 180   2      END PDP$INT;



              /*********************************************
               *   MAIN PROGRAM LOOP.  INITIALISATION    *
               *   PARSER CONTROL.                       *
               *********************************************/


              /* INITIALISE INTER-PROCESSOR PARAMATERS AND D/A OUTPUTS */
 181   1         DISABLE;
 182   1         STATUS = 0;
 183   1         SETPOINT = 0;
 184   1         FLOW = 0;
 185   1         INITIAL$CONDITION = 0;
 186   1         FAN$SPEED = 0;
 187   1         CALL DAC(0,FAN$SPEED);

              /* INITIALISE KEYBOARD/DISPLAY.  THE KEYBOARD MODE IS
                 ENCODE 2-KEY LOCKOUT.  DISPLAY MODE IS 8 - 8 BIT
                 LEFT ENTRY. */
 188   1         OUTPUT(KB$STATUS$PORT) = 0;
 189   1         OUTPUT(KB$STATUS$PORT) = 39H;
 190   1         CALL ACTION$0;                      /* BLNK DISP */

              /* INITIALISE KEYBOARD DIGIT BUFFER */
 191   1         DO I = 0 TO 3;
 192   2          KB$DIGIT$BUFFER(I) = 0;
 193   2         END;

              /* INITIALISE PARSER PARAMATERS */
 194   1         STATE = 0;
 195   1         ENABLE;


              /******************
               *  PARSER LOOP  *
               ******************/
 196   1         DO WHILE FOREVER;

              /* TEST KEYBOARD STATUS...SCAN IPI IF NO KEY INPUT */
 197   2         DO WHILE (INPUT(KB$STATUS$PORT) AND 0FH) <> 0;
 198   3          CALL PDP$INT;
 199   3         END;

              /* FETCH KEYBOARD CHARACTER AND DETERMINE THE
                 CORRESPONDING VALUES OF FUNCTION AND NUMBER */
 200   2         CHAR = KB$GET$CHAR;
 201   2         IF CHAR <= 9 THEN
 202   2         DO;
 203   3          FUNCTION = 1;
 204   3          NUMBER = CHAR;
 205   3         END;
 206   2         ELSE IF CHAR = KB$PERIOD
                 THEN FUNCTION = 2;
 208   2         ELSE IF CHAR = KB$COMMA
```

```
            THEN FUNCTION = 3;
            ELSE IF CHAR = KB$MINUS
            THEN FUNCTION = 4;
            ELSE IF CHAR = KB$PLUS
            THEN FUNCTION = 5;
            ELSE IF CHAR = KB$COLON
            THEN FUNCTION = 6;
            ELSE FUNCTION = 0;

        /* FIND CORRECT PARSER TABLE ENTERY */
            I = 0;
            NOMATCH = 1;
            DO WHILE NOMATCH;
             IF (PARSER$TABLE(STATE).FUNCTION(I) = 0) OR
                (PARSER$TABLE(STATE).FUNCTION(I) = FUNCTION)
             THEN
             DO;
              ACTION = PARSER$TABLE(STATE).ACTION(I);
              STATE = PARSER$TABLE(STATE).NEXT$STATE(I);
              NOMATCH = 0;
             END;
             ELSE I = I + 1;
            END;

        /* INITIATE REQUIRED ACTION */
            DO CASE ACTION;
             CALL ACTION$0;
             CALL ACTION$1;
             CALL ACTION$2;
             CALL ACTION$3;
             CALL ACTION$4;
             CALL ACTION$5;
             CALL ACTION$6;
             ;
            END;

        END;    /* END PARSER LOOP */


            END FLOWLOOP;

        /***************************************************************************/


MODULE INFORMATION:

     CODE AREA SIZE     = 05B6H   1462D
     CONSTANT AREA SIZE = 0064H    100D
     VARIABLE AREA SIZE = 0026H     38D
     MAXIMUM STACK SIZE = 0016H     22D
     549 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

```
ISIS-II MCS-86 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE UTIL
OBJECT MODULE PLACED IN :F1:UTIL.OBJ
ASSEMBLER INVOKED BY: :F2:ASM86 :F1:UTIL.ASM DEBUG PRINT(:LP:)

LOC  OBJ             LINE     SOURCE

----                    1     DSA     STRUC
0000                    2     OLDBP   DW        ?
0002                    3     RETURN  DW        ?
0004                    4     VALUE   DW        ?
----                    5     DSA     ENDS
                        6     ;
                        7     CGROUP  GROUP CODE
                        8             ASSUME CS:CGROUP
                        9     ;
----                   10     CODE    SEGMENT PUBLIC 'CODE'
                       11     ;
                       12             PUBLIC PRCNT,SPAN
                       13     ;
                       14     ;THESE ROUTINES INTERFACE TO PLM86 FLO
                       15     ;DECLARATIONS.
                       16     ;
                       17
0000                   18     PRCNT   PROC      NEAR
0000 55                19             PUSH      BP
0001 8BEC              20             MOV       BP,SP
                       21     ;
                       22     ;MULTIPLY BY 2.71H.
0003 8B4604            23             MOV       AX,[BP].VALUE
0006 B97102            24             MOV       CX,271H
0009 F7E1              25             MUL       CX
000B B90001            26             MOV       CX,100H
000E F7F1              27             DIV       CX
0010 03D2              28             ADD       DX,DX
0012 2BCA              29             SUB       CX,DX
0014 7A01              30             JP        PR1
0016 40                31             INC       AX
0017 5D                32     PR1:    POP       BP
0018 C20200            33             RET       2H
                       34     ;
                       35     PRCNT   ENDP
                       36     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
001B                   37     SPAN    PROC      NEAR
001B 55                38             PUSH      BP
001C 8BEC              39             MOV       BP,SP
                       40     ;
                       41     ;DIVIDE BY 2.71H.
001E 8B4604            42             MOV       AX,[BP].VALUE
0021 B90001            43             MOV       CX,100H
0024 F7E1              44             MUL       CX
0026 B97102            45             MOV       CX,271H
0029 F7F1              46             DIV       CX
002B 03D2              47             ADD       DX,DX
002D 2BCA              48             SUB       CX,DX
002F 7A01              49             JP        SP1
```

```
LOC  OBJ                 LINE     SOURCE

0031 40                    50             INC      AX
0032 5D                    51     SP1:    POP      BP
0033 C20200                52             RET      2H
                           53     ;
                           54     SPAN    ENDP
                           55     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
----                       56     CODE    ENDS
                           57     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

B.3 KEYBOARD PARSER - STAGE 1

The program 'FLOWLOOP' (B.1) is designed using a 'state machine' approach. The keyboard inputs control the state transitions and actions to be initiated, with operator feedback being provided by eight 7-segment LEDs (light emitting diodes). These display information as follows (LEDs are numbered from left to right);

(1) LED 2 is the auto/manual indicator ('a' for auto, 'm' for manual),

(2) LED 4 describes the type of value being displayed in the following field ('s' for setpoint when in auto, 'r' for manual output, 'c' for decrement/increment value),

(3) LEDS 5 to 8 display the percentage span value as defined in (2) above (two decimal places).

The above are defined as fields 1, 2 and 3 respectively.

The table below describes the operation of the keyboard parser (note that the input '*' refers to any input excluding those already defined in the current state);

APPENDIX B

STATE	INPUT	NEXT STATE	ACTION
0	,	1	1
	*	0	0
1	.	0	0
	:	1	2
	,	2	3
	*	1	NIL
2	.	0	0
	,	1	1
	-	4	4
	+	4	5
	DIGIT	3	6
	*	2	NIL
3	.	0	0
	-	4	4
	+	4	5
	DIGIT	3	6
	*	2	3
4	.	0	0
	-	4	4
	+	4	5
	*	2	3

The 'action' numbers refer to the following actions:

(1) Action 0. The LED display is cleared.

(2) Action 1. If in auto mode the setpoint is displayed as percentage of span. If in manual mode the controlling output is displayed as percentage of span.

(3) Action 2. The mode is toggled between auto and manual. The display is effected as in 'action 1'.

(4) Action 3. Displays the current 'ramp number'.

(5) Action 4. If in auto mode the 'ramp number' is subtracted from the setpoint. If in manual mode it is subtracted from the controlling output value (essentially this is a ramp down function). The display is effected as in 'action 1'.

(6) Action 5. As in 'action 4', except addition is performed instead of subtraction (ramp up function).

(7) Action 6. Displays additional digits as the 'ramp number' is entered.

As an example, consider that the system is in manual mode (on initialisation) and in state 0. Then if "," is pressed the state table shows that state 1 will be entered and 'action 1' performed (i.e. controller output is displayed as percentage of span). If this is followed by "." then state 0 is re-entered and 'action 0' performed (i.e. the display is cleared).

B.4 SDK-86 SOFTWARE DESCRIPTION - STAGE 2

Stage 2 development was not done by the author, and as the software descriptions were minimal, the following description is brief.

The software tasks fall into two categories; foreground and background. The background software is essentially the main program loop, which implements the channel and loop mode features.

The foreground tasks (interrupt procedures 64, 65 and 66) handle the real time data transfer. The ADC requests from the LSI-11/03 are queued on an 8 word circular buffer by interrupt procedure 65, whenever an A/D conversion is already in progress. This prevents the loss of multiple requests. If no A/D conversion is in progress, interrupt procedure 65 will immediately process the request. Pending ADC requests on the queue are serviced by interrupt procedure 64.

The DAC requests by the LSI-11/03 are serviced by interrupt procedure 66. Whenever data is 'written' to a DAC register by the LSI-11/03, this procedure transfers the data in each DAC register to the corresponding D/A convertor. This approach was adopted since there is no way of identifying the DAC register that has just received data.

B.5 SDK-86 SOFTWARE LISTING - STAGE 2

```
ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE PLO
OBJECT MODULE PLACED IN :F1:PLO.OBJ
COMPILER INVOKED BY:  :F1:PLM86 :F1:PLO.PLM ROM COMPACT OPTIMIZE(2) DATE(16-MAY-82) &
                      NOINTVECTOR PRINT(:F1:PLO.LST) PAGELENGTH(45)



   1           PLO: DO;                                                   /* Ver 1.2 */

                 /************************************************************************


                       AVUE Consulants                                  Copyright Jan 1982
                       Another MAGENTA Production
                           Written by B.W. Korbel

                       This program allows the SDK-86 to simulate DEC ADV-11's and
                          AAV-11's.
                       It also allows the setting of 'switches' for loop control
                          and allows 'loop' monitoring.


                 ************************************************************************/


                     /****** hardware change section *****/
   2   1             Declare
                        NO$OF$DACS       literally '06',   /* no of DAC's */
                        NO$OF$LOOPS      literally '08';   /* no of loops */

                     /****** A/D section *****/
   3   1             DECLARE
                        ADC$ADDR         literally '0FF00H',   /* adc base address */
                        ADC(16)          WORD,                 /* store of last value */
                        ADC$PTR          BYTE,                 /* ptr to above store */
                        ADC$QUEUE(8)     BYTE,                 /* queue for conversion */
                        ADC$Q$IN         BYTE,
                        ADC$Q$OUT        BYTE,
                        ADC$BUSY         BYTE;

                     /***** D/A section *****/
   4   1             DECLARE
```

```
                         DAC$ADDR          literally 'OFF20H',   /* dac base address */
                         DAC(12)           WORD;

                      /***** IPI section *****/
  5   1               DECLARE
                         ADV$CSR           literally 'OFF40H',   /* ADC csr address  */
                         ADV$DBR           literally 'OFF40H',   /* ADC dbr address  */
                         AAV$ADDR          literally 'OFF42H';   /* DAC address      */

                      /***** Interrupt section *****/
  6   1               DECLARE
                         INT$REG$0         literally 'OFF60H',   /* P8259A reg 0 */
                         INT$REG$1         literally 'OFF62H',
                         ICW1$I            literally '1FH',     /* see specs sheets */
                         ICW2              literally '40H',     /* for more info    */
                         ICW4$D            literally '01H',
                         OCW1              literally '0F8H',
                         OCW2$E            literally '20H',
                         ADC$EOC$FF        literally 'OFF64H',  /* IR0 F/F reset */
                         ADC$SOC$FF        literally 'OFF68H',  /* IR1 F/F reset */
                         DAC$REQUEST$FF    literally 'OFF6CH',  /* IR2 F/F reset */
                         SHOW$INTERRUPTS   BYTE;

                      /****** key-board and display section ******/
  7   1               DECLARE
                         KB$STATUS$PT      literally 'OFFEAH',
                         KB$DATA$PT        literally 'OFFE8H',
                         KB$MODE           literally '00H',     /* 8 8-bit left entry */
                         KB$SCAN$RATE      literally '39H',     /* 10 mSec scan rate  */
                         KB$INTRDY         literally '07H',

                         DIGIT(*)          BYTE DATA
                                           (3FH,06H,5BH,4FH,66H,6DH,7DH,07H,    /* 0..7 */
                                            7FH,6FH,77H,7CH,39H,5EH,79H,71H),   /* 8..F */

                         DIS$A             literally '077H',
                         DIS$M             literally '054H',
                         DIS$S             literally '0EDH',
                         DIS$R             literally '050H',
                         DIS$C             literally '039H',
                         DIS$D             literally '05EH',
                         DIS$H             literally '076H',
        DIS$P           literally '073H',
        DIS$L           literally '038H',
        DIS$U           literally '03EH',
        DIS$V           literally '01CH',
        DIS$E           literally '079H',
        DIS$O           literally '03FH',
        DIS$T           literally '078H',
        DIS$F           literally '071H',
        BLANK           literally '000H',
        DASH            literally '040H',
        DOT             literally '080H',

        AVUE(*)         BYTE DATA
                        (8,DOT,DOT,DOT,DOT,DIS$E,DIS$U,DIS$V,DIS$A),
        IDENT(*)        BYTE DATA
                        (8,DIS$O,DIS$L,DIS$P,BLANK,BLANK,5BH,86H,BLANK),
        LOOP(*)         BYTE DATA
                        (8,DOT,BLANK,BLANK,BLANK,DIS$P,DIS$O,DIS$O,DIS$L),
        CHAN(*)         BYTE DATA
                        (8,DOT,BLANK,BLANK,BLANK,DIS$M,DIS$A,DIS$H,DIS$C),
        SELECT$DAC(*)   BYTE DATA
                        (4,BLANK,BLANK,DIS$A,DIS$D),
        SELECT$ADC(*)   BYTE DATA
                        (4,BLANK,BLANK,DIS$D,DIS$A),
        LPS(*)          BYTE DATA
                        (6,BLANK,BLANK,BLANK,BLANK,DIS$P,DIS$L),
        CLEAR(*)        BYTE DATA
                        (8,00H,00H,00H,00H,00H,00H,00H,00H),
        PSCF(*)         BYTE DATA
                        (8,7FH,06H,3FH,86H,DIS$F,DIS$C,DIS$S,DIS$P),
        HALF$BLANK(*)   BYTE DATA
                        (4,BLANK,BLANK,BLANK,BLANK),
        DOTS(*)         BYTE DATA
                        (4,DOT,DOT,DOT,DOT),
        KB$COMMA        literally '11H',
        KB$PERIOD       literally '10H',
        KB$MINUS        literally '12H',
        KB$PLUS         literally '13H',
        KB$COLON        literally '14H',
        KB$REG          literally '15H';

      /***** loop section *****/
```

```
 8   1          DECLARE
                   STATUS            WORD,                     /* status given to LSI */
                   INDEX             WORD,
                   PROCESSING$LOOP   BYTE,
                   LOOP$STATUS(8)    BYTE,
                   LOOP$DAC(8)       BYTE,
                   LOOP$ADC(8)       BYTE,
                   CNTRL$PT$HI       literally 'OFFFEH',   /* 8255 cntrl pt */
                   CNTRL$PT$LO       literally 'OFFFFH',   /* 8255 cntrl pt */
                   STATUS$PT         literally 'OFFF8H',   /* 16 lines to LSI */
                   CNTRL$BYTE        literally '08BH',     /* PA output, PB,PC input */

                   AUTO              literally '00H',
                   RAISE             literally '01H',
                   LOWER             literally '02H',
                   HOLD              literally '03H',
                   TOGGLE            literally '00H';


                /***** main program variables *****/
 9   1          DECLARE
                   FOREVER           literally 'WHILE OFFH',
                   UNTIL             literally 'WHILE CHAR$NOT$PRESENT',
                   PRESENT           literally '',

                   TRUE              literally 'OFFH',
                   FALSE             literally '00H',

                   NUM               literally 'TRUE',
                   HEX               literally 'FALSE',

                  (VALUE,START$UP$FLAG)        WORD,
                  (I64,I65,I66)                BYTE,    /* interrupt counters */
                  (SHOW$ADC,DISPLAY$TYPE)      BYTE,    /* display options */
                  (SNEEK$CHAR,STATE,UP$DOWN)   BYTE,    /* loop position markers */
                  (CHANNEL,ACTIVE)             BYTE,
                  (COLD,SIGN$TIME)             BYTE;

              /********** interrupt error variables **********/
10   1          DECLARE
                VECTOR$PTR        POINTER,
                VECTOR            BASED VECTOR$PTR
```

```
                         (255) STRUCTURE (OFF WORD, SEG WORD),
        JUNK$ERROR        WORD,
        PIC$ERROR         WORD,
        JUNK$WORD         WORD;
```

```
            $ SUBTITLE('Main support routines')


               /* ***************     MAIN SUPPORT ROUTINES     *************** */


            /************/
 11   1        READ$KB:PROCEDURE BYTE;   /* *****   READ$KB   ***** */

               /* WAIT TIIL CHAR PRESENT */
 12   2        DO WHILE (INPUT(KB$STATUS$PT) AND KB$INTRDY)=0; END;
 14   2        OUTPUT(KB$STATUS$PT) = 040H;   /* ENABLE INPUT DATA */
 15   2        RETURN INPUT(KB$DATA$PT);

 16   2        END READ$KB;


            /***********/
 17   1        DISPLAY:PROCEDURE(CHAR,POSITION) REENTRANT;   /* *****   DISPLAY   ***** */
 18   2        DECLARE (CHAR,POSITION) BYTE;

               /* WAIT TILL DISPLAY READY */
 19   2        DO WHILE ( INPUT(KB$STATUS$PT) AND 80H ) <> 0; END;
 21   2        OUTPUT(KB$STATUS$PT)=POSITION OR 80H;  /* ENABLE OUTPUT DATA */
 22   2        OUTPUT(KB$DATA$PT)=CHAR;

 23   2        END DISPLAY;


            /***********/
 24   1        DISPLAY$DIGIT:PROCEDURE (CHAR,POSITION) REENTRANT;
 25   2        DECLARE (CHAR,POSITION) BYTE;

 26   2        CALL DISPLAY(DIGIT(CHAR),POSITION);
 27   2        END;


            /***********/
 28   1        DISPLAY$MESSAGE: PROCEDURE (MESS$PTR);
```

```
                         MESSAGE BASED MESS$PTR (1) BYTE,
                         I BYTE;

 30   2            DO I=0 TO LENGTH - 1;
 31   3               CALL DISPLAY(MESSAGE(I+1),I);
 32   3            END;

 33   2            END DISPLAY$MESSAGE;


                /***********/
 34   1            DELAY: PROCEDURE (I);
 35   2            DECLARE (J,I) BYTE;

 36   2            DO J = 1 TO I*25; CALL TIME(250) ; END;
 39   2            END DELAY;


                /***********/
 40   1            DISPLAY$NUM: PROCEDURE(VALUE);
 41   2            DECLARE VALUE WORD,
                           BCD(5) BYTE;

 42   2            BCD(0)=4;
 43   2            VALUE = (VALUE*12)/5;              /* scale OFFFH ==> 10000D */

 44   2            BCD(4) = DIGIT(VALUE/1000);        /* extract msb */
 45   2            VALUE =  VALUE MOD 1000;
 46   2            BCD(3) = DIGIT(VALUE/100) OR DOT;  /* extract next digit */
 47   2            VALUE =  VALUE MOD 100;
 48   2            BCD(2) = DIGIT(VALUE/10);          /* extract next digit */
 49   2            BCD(1) = DIGIT(VALUE MOD 10);      /* extract lsb */

 50   2            CALL DISPLAY$MESSAGE(@BCD);        /* display the BCD equivalent */

 51   2            END DISPLAY$NUM;


                /***********/
```

```
  54   2          DO I=0 TO 3;
  55   3             CALL DISPLAY$DIGIT( (SHR(VALUE,4*I) AND 0FH) , I);
  56   3          END;

  57   2          END DISPLAY$HEX;
```

```
              $ SUBTITLE('Interrupt and 'change' flow routines')


              /***********/
 58   1         DOPE: PROCEDURE INTERRUPT 2 REENTRANT;
 59   2         DECLARE CAL(*) BYTE DATA
                               (0EAH,1CH,00H,0C0H,0FFH),
                        MON(*) BYTE DATA
                               (0EAH,00H,00H,00H,0FEH),
                        CODE POINTER;

 60   2         ENABLE;
 61   2         CALL DISPLAY$MESSAGE(@HALF$BLANK);

 62   2         CALL DISPLAY(DIS$M,0);
 63   2         IF READ$KB=KB$PERIOD THEN  DO; CODE=@MON; CALL CODE; END;

 68   2         CALL DISPLAY(DIS$C,0);
 69   2         IF READ$KB=KB$PERIOD THEN DO; CODE=@CAL; CALL CODE; END;


 74   2         END DOPE;


              /***********/
 75   1         JUNK: PROCEDURE INTERRUPT 0;

 76   2         JUNK$ERROR=(JUNK$ERROR+1) AND 0FFH;

 77   2         END JUNK;


              /***********/
 78   1         PIC: PROCEDURE INTERRUPT 67;

 79   2         PIC$ERROR=(PIC$ERROR+1) AND 0FFH;

 80   2         END PIC;
 81   1         ADC$EOC: PROCEDURE INTERRUPT 64;   /* a ADC request has been processed */
 82   2         DECLARE CHANNEL BYTE;

 83   2         OUTPUT(ADC$EOC$FF)=0;
 84   2         ADC(ADC$PTR) , OUTWORD(ADV$DBR) = SHR(INWORD(ADC$ADDR),1) AND 0FFFH;
 85   2         IF ADC$Q$OUT=ADC$Q$IN
                   THEN
 86   2               ADC$BUSY=FALSE;
                   ELSE
 87   2               DO;
 88   3                  ADC$Q$OUT=(ADC$Q$OUT+1) AND 07H;    /* increment queue pointer */
 89   3                  CHANNEL=ADC$QUEUE(ADC$Q$OUT);       /* extract channel no.     */
 90   3                  OUTPUT(ADC$ADDR+CHANNEL)=0;         /* start a conversion      */
 91   3                  ADC$PTR=SHL(CHANNEL,1);             /* senrate buffer pointer  */
 92   3               END;
 93   2         IF SHOW$INTERRUPTS THEN
 94   2             DO; I64=(I64+1) AND 0FH; CALL DISPLAY$DIGIT((I64),2); END;
 98   2         OUTPUT(INT$REG$0)=OCW2$E;

 99   2         END ADC$EOC;


         /**********/
100   1         ADC$SOC: PROCEDURE INTERRUPT 65;  /* Start an ADC on prescribed channel */
101   2         DECLARE CHANNEL BYTE;

102   2         OUTPUT(DAC$REQUEST$FF)=0;              /* clear DAC interrupt generated */
103   2         OUTPUT(ADC$SOC$FF)=0;

104   2         IF INPUT(ADV$CSR) THEN                 /* ie. if 'Go bit' set, continue */
105   2            DO;
106   3            CHANNEL = INPUT(ADV$CSR) AND 1EH;   /* read CSR just written by LSI */
107   3            IF ADC$BUSY
                      THEN
108   3                  DO;
109   4                     ADC$Q$IN=(ADC$Q$IN + 1) AND 07H;
110   4                     ADC$QUEUE(ADC$Q$IN)=CHANNEL;
111   4                  END;
                      ELSE
 114   4                  OUTPUT(ADC$ADDR + CHANNEL) = 0H;  /* start an conversion */
 115   4                  ADC$PTR = SHR(CHANNEL,1);
 116   4               END;
 117   3            END;

 118   2         OUTPUT(INT$REG$0)=OCW2$E;

 119   2         IF SHOW$INTERRUPTS THEN
 120   2            DO; I65=(I65+1) AND 0FH;  CALL DISPLAY$DIGIT((I65), 1);  END;
 124   2         END ADC$SOC;


               /**********/
 125   1         DAC$REQUEST: PROCEDURE INTERRUPT 66;
 126   2         DECLARE VALUE WORD,
                         (I,J) BYTE;

 127   2         OUTPUT(DAC$REQUEST$FF)=0;
 128   2         ENABLE;

 129   2         DO I=0 TO NO$OF$DACS-1;
 130   3            J=SHL(I,1);
 131   3            VALUE , DAC(I) = INWORD(AAV$ADDR + J) AND 0FFFH ;
 132   3            OUTWORD(DAC$ADDR+J)= NOT VALUE;
 133   3         END;

 134   2         IF SHOW$INTERRUPTS THEN
 135   2            DO;  I66=(I66+1) AND 0FH;  CALL DISPLAY$DIGIT((I66),0); END;

 139   2         OUTPUT(INT$REG$0)=OCW2$E;

 140   2         END DAC$REQUEST;


               /**********/
 141   1         CHAR$NOT$PRESENT:PROCEDURE (CHAR) BYTE;
 142   2         DECLARE (CHAR,FLAG) BYTE;

 143   2         FLAG , SNEEK$CHAR = TRUE;
 146   3            OUTPUT(KB$STATUS$PT) = 40H;    /* enable input data */
 147   3            SNEEK$CHAR = INPUT(KB$DATA$PT);      /* read char */
 148   3            IF CHAR = SNEEK$CHAR THEN FLAG=FALSE;
 150   3            END;
 151   2         RETURN FLAG;   /* ie. 'true' if 'char' not present */
                                /*     'false' if 'char' present */

 152   2         END CHAR$NOT$PRESENT;
```

```
             $ SUBTITLE('Loop process routines')


             /**********/
 153   1        CURRENT$STATUS:PROCEDURE (CMMD) BYTE;
 154   2        DECLARE CMMD BYTE;

 155   2        RETURN SHR(LOOP$STATUS(CHANNEL),CMMD);
 156   2        END CURRENT$STATUS;


             /**********/
 157   1        SET$LOOP$DEVICE: PROCEDURE(MESS$PTR,DEVICE$PTR);
 158   2        DECLARE (DEVICE$PTR,MESS$PTR)  POINTER,
                        DEVICE BASED DEVICE$PTR BYTE,
                        I BYTE;

 159   2        CALL DISPLAY$MESSAGE(MESS$PTR);
 160   2        I=DEVICE;

 161   2        DO WHILE I <> KB$PERIOD;
 162   3           DEVICE=I AND 0FH;                      /* generate new device pointer */
 163   3           CALL DISPLAY(DIGIT(DEVICE) OR DOT,0);  /* display it */
 164   3           I=READ$KB;                             /* see if new one present */
 165   3        END;

 166   2        END SET$LOOP$DEVICE;


             /**********/
 167   1        LOOP$DEVICES: PROCEDURE;

 168   2        CALL SET$LOOP$DEVICE(@SELECT$DAC,@LOOP$DAC(CHANNEL));   /*    DAC no.   */
 169   2        CALL SET$LOOP$DEVICE(@SELECT$ADC,@LOOP$ADC(CHANNEL));   /*    ADC no.   */

 170   2        CALL DISPLAY$MESSAGE(@LPS);
 171   2        CALL DISPLAY$HEX(INDEX OR LOOP$STATUS(CHANNEL));
 172   2        CALL DELAY(2);
```

```
              /**********/
 174   1         SET$STATUS:PROCEDURE (CMMD);
 175   2         DECLARE CMMD BYTE, STATUS BYTE;

 176   2         STATUS = LOOP$STATUS(CHANNEL);

 177   2         DO CASE CMMD;

                    /* toggle auto/manual (cmmd=0) if 'auto' clear LSB, if 'manual' set LSB */
 178   3            IF STATUS THEN STATUS = STATUS AND 1111$1110B;
 180   3                     ELSE STATUS = STATUS OR  0000$0001B;

 181   3            DO; /* raise (cmmd=1) raise=1, lower=0 */
 182   4               STATUS = STATUS OR  0000$0010B;   /* ie. set bit 1 */
 183   4               STATUS = STATUS AND 1111$1011B;   /* ie. clear bit 2 */
 184   4            END;

 185   3            DO; /* lower (cmmd=2) raise=0, lower=1 */
 186   4               STATUS = STATUS AND 1111$1101B;   /* ie. clear bit 1 */
 187   4               STATUS = STATUS OR  0000$0100B;   /* ie. set bit 2 */
 188   4            END;

                        /* hold (cmmd=3) raise=0, lower=0 */
 189   3            STATUS = STATUS AND 1111$1001B;      /* ie. clear bits 1&2 */

 190   3         END; /* end case */

 191   2         LOOP$STATUS(CHANNEL)=STATUS;
 192   2         OUTWORD(STATUS$PT) = INDEX OR STATUS;  /* shuve status out to LSI */

 193   2         END SET$STATUS;


              /**********/
 194   1         ACTIVITY: PROCEDURE;
 195   2         DECLARE I BYTE;

 196   2         I=(I+1) AND 0FH;
 199   3              ACTIVE=NOT ACTIVE;
 200   3              IF ACTIVE THEN CALL DISPLAY(DOT,6);
 202   3                        ELSE CALL DISPLAY(BLANK,6);
 203   3           END;

 204   2        END ACTIVITY;
```

```
$SUBTITLE('Key porcessing routines')


/**********/
   PROCESS$LOOP: PROCEDURE;

   INDEX=1;
   INDEX=SHL(INDEX,CHANNEL+8);    /* generate ptr for status word */
   STATUS = INDEX OR LOOP$STATUS(CHANNEL);
   OUTWORD(STATUS$PT) = STATUS;
   CALL DISPLAY$HEX(STATUS);
   CALL DELAY(2);

   DO UNTIL (KB$PERIOD) PRESENT;    /*** main processing loop ***/

      CALL ACTIVITY;

      UP$DOWN=BLANK;
      IF CURRENT$STATUS(RAISE) THEN UP$DOWN=DIS$R;
         ELSE IF CURRENT$STATUS(LOWER) THEN UP$DOWN=DIS$L;
      CALL DISPLAY(UP$DOWN,4);

      IF CURRENT$STATUS(AUTO)
          THEN  DO; UP$DOWN=DIS$A; VALUE=ADC(LOOP$ADC(CHANNEL)); END;
          ELSE  DO; UP$DOWN=DIS$M; VALUE=DAC(LOOP$DAC(CHANNEL)); END;
      CALL DISPLAY(UP$DOWN,5);
      CALL DISPLAY$NUM(VALUE);

      IF SNEEK$CHAR <> TRUE THEN
         DO;   /* a key has been pressed */
         STATE = TRUE;
         IF SNEEK$CHAR = KB$COMMA  THEN STATE=TOGGLE;
            ELSE IF SNEEK$CHAR = KB$PLUS THEN STATE = RAISE;
               ELSE IF SNEEK$CHAR = KB$MINUS THEN STATE = LOWER;
                  ELSE IF SNEEK$CHAR = KB$COLON THEN STATE = HOLD;
                     ELSE IF SNEEK$CHAR = KB$REG THEN CALL LOOP$DEVICES;
         IF STATE <> TRUE THEN CALL SET$STATUS(STATE);
         END;   /* set state */
   END;   /*** until(period) ***/
```

```
 248   2          END PROCESS$LOOP;


               /**********/
 249   1          DISPLAY$CHANNEL: PROCEDURE;
 250   2          DECLARE VALUE$OK BYTE;

 251   2          VALUE$OK=TRUE;
 252   2          IF SHOW$ADC
                     THEN
 253   2                DO;              /* a ADC value has been requested */
 254   3                UP$DOWN=DIS$A;
 255   3                VALUE=ADC(CHANNEL);
 256   3                END;
                     ELSE
 257   2                DO;              /* a DAC value has been asked for */
 258   3                UP$DOWN=DIS$D;
 259   3                IF CHANNEL<NO$OF$DACS THEN VALUE=DAC(CHANNEL);
 261   3                                      ELSE VALUE$OK=FALSE;
 262   3                END;

 263   2          CALL DISPLAY(UP$DOWN,5);
 264   2          IF VALUE$OK
                      THEN
 265   2                IF DISPLAY$TYPE=NUM THEN CALL DISPLAY$NUM(VALUE);  /* D$TYPE = NUM */
 267   2                                     ELSE CALL DISPLAY$HEX(VALUE);  /* D$TYPE = HEX */
                      ELSE
 268   2                CALL DISPLAY$MESSAGE(@DOTS);

 269   2          END DISPLAY$CHANNEL;


               /**********/
 270   1          PROCESS$CHANNEL: PROCEDURE;

 271   2          DO UNTIL (KB$PERIOD) PRESENT;
 272   3             CALL ACTIVITY;

 273   3             IF SNEEK$CHAR = KB$COLON THEN DISPLAY$TYPE = NOT DISPLAY$TYPE;
 279   3                    ELSE IF SNEEK$CHAR = KB$COMMA THEN SHOW$ADC = NOT SHOW$ADC;

                  CHANNEL = CHANNEL AND 0FH;
 282   3          CALL DISPLAY$DIGIT((CHANNEL),7);

 283   3          CALL DISPLAY$CHANNEL;
 284   3        END; /* do until(period) */

 285   2        END PROCESS$CHANNEL;


              /**********/
 286   1        PRE$PROCESS: PROCEDURE BYTE;
 287   2        DECLARE (STATE,I) BYTE;

 288   2        I=0;                  /* intialize states */
 289   2        STATE=FALSE;

 290   2        DO WHILE (I<>KB$PERIOD) AND (I<>KB$COMMA); /* loop till period or comma */
 291   3           I=READ$KB;
 292   3           IF I=KB$COLON THEN SHOW$INTERRUPTS=NOT SHOW$INTERRUPTS;
 294   3           IF I=KB$REG THEN
 295   3              DO;
 296   4              JUNK$WORD=SHL(JUNK$ERROR,8) OR PIC$ERROR;
 297   4              CALL DISPLAY$HEX(JUNK$WORD);
 298   4              END;
 299   3           IF I < 10H THEN
 300   3              DO;
 301   4                 IF PROCESSING$LOOP THEN I=I AND 07H;
 303   4                 CHANNEL=I;
 304   4                 CALL DISPLAY(DIGIT(CHANNEL) OR DOT,0);
 305   4              END;
 306   3           IF I = KB$PERIOD THEN STATE = TRUE;
 308   3        END;

 309   2        CALL DISPLAY$MESSAGE(@CLEAR);
 310   2        CALL DISPLAY$DIGIT((CHANNEL),7);

 311   2        RETURN STATE;
```

```
            $SUBTITLE('Main control program')

             /*************************  MAIN PROGRAM *********************/

 313   1        DISABLE;

 314   1        OUTPUT(KB$STATUS$PT)=KB$MODE;          /* Set mode */
 315   1        OUTPUT(KB$STATUS$PT)=KB$SCAN$RATE;     /* Set scan rate */

 316   1        IF START$UP$FLAG <> 5555H THEN
 317   1          DO;                        /*****    COLD START ONLY STUFF *****/
 318   2            START$UP$FLAG=5555H;
 319   2            COLD=TRUE;

 320   2            SIGN$TIME=6;

 321   2            INDEX=0;                            /* initialize 'loop' conditions */
 322   2            DO CHANNEL=0 TO 7;
 323   3              LOOP$STATUS(CHANNEL)=0;
 324   3              LOOP$DAC(CHANNEL) , LOOP$ADC(CHANNEL) = CHANNEL;
 325   3              END;

 326   2            DO CHANNEL=0 TO 15; ADC(CHANNEL)=0000H; END;
 329   2            ADC$BUSY=FALSE;                     /* initialize the ADC values */
 330   2            ADC$Q$IN , ADC$Q$OUT = 0;

 331   2            SHOW$ADC=FALSE;
 332   2            DISPLAY$TYPE=HEX;

 333   2            DO CHANNEL=0 TO NO$OF$DACS-1;       /* initialize the DAC values */
 334   3              DAC(CHANNEL)=0000H;
 335   3              OUTWORD(DAC$ADDR + SHL(CHANNEL,1))= 0FFFH;
 336   3            END;

 337   2            SHOW$INTERRUPTS=FALSE;
 338   2            OUTPUT(ADC$EOC$FF)=0;               /* clear the interrupt F/F's */
 339   2            OUTPUT(ADC$SOC$FF)=0;
 340   2            OUTPUT(DAC$REQUEST$FF)=0;
 341   2            JUNK$ERROR , PIC$ERROR =0;
```

```
                                       /*****    WARM START   *****/

 343   1         IF COLD=FALSE THEN SIGN$TIME=2;
 345   1         COLD=FALSE;

 346   1         CALL DISPLAY$MESSAGE(@AVUE);  CALL DELAY(SIGN$TIME);
 348   1         CALL DISPLAY$MESSAGE(@IDENT); CALL DELAY(SIGN$TIME/2);
 350   1         CALL DISPLAY$MESSAGE(@PSCF);  CALL DELAY(SIGN$TIME/2);

 352   1         CALL SET$INTERRUPT(0,JUNK);

 353   1         VECTOR$PTR=0;
 354   1         DO CHANNEL=1 TO 255;
 355   2            VECTOR(CHANNEL).OFF=VECTOR(0).OFF;
 356   2            VECTOR(CHANNEL).SEG=VECTOR(0).SEG;
 357   2            END;

 358   1         CALL SET$INTERRUPT(67,PIC);
 359   1         CALL SET$INTERRUPT(68,PIC);
 360   1         CALL SET$INTERRUPT(69,PIC);
 361   1         CALL SET$INTERRUPT(70,PIC);
 362   1         CALL SET$INTERRUPT(71,PIC);

 363   1         CALL SET$INTERRUPT(64,ADC$EOC);          /* set interrupt vectors */
 364   1         CALL SET$INTERRUPT(65,ADC$SOC);
 365   1         CALL SET$INTERRUPT(66,DAC$REQUEST);
 366   1         CALL SET$INTERRUPT(2,DOPE);

 367   1         I64,I65,I66 =0;                          /* initialize interrupt counters */
 368   1         OUTPUT(CNTRL$PT$HI) = CNTRL$BYTE;        /* set up 8255's */
 369   1         OUTPUT(CNTRL$PT$LO) = CNTRL$BYTE;

 370   1         OUTWORD(STATUS$PT)= INDEX OR LOOP$STATUS(CHANNEL);

 371   1         OUTPUT(INT$REG$0) = ICW1$I;              /*  set up PIC (ie. P8259A ) */
 372   1         OUTPUT(INT$REG$1) = ICW2;                /* see spec sheets for details */
 373   1         OUTPUT(INT$REG$1) = ICW4$D;
 374   1         OUTPUT(INT$REG$1) = OCW1;
 377   1          ENABLE;

 378   1          DO FOREVER;      /********** MAIN LOOP *********/

 379   2             CALL DISPLAY$MESSAGE(@LOOP);
 380   2             PROCESSING$LOOP=TRUE;
 381   2             IF PRE$PROCESS THEN CALL PROCESS$LOOP;

 383   2             CALL DISPLAY$MESSAGE(@CHAN);
 384   2             PROCESSING$LOOP=FALSE;
 385   2             IF PRE$PROCESS THEN CALL PROCESS$CHANNEL;

 387   2          END;   /*** FOREVER ***/

 388   1       END PLO;


MODULE INFORMATION:

     CODE AREA SIZE     = 0AF4H   2804D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0091H    145D
     MAXIMUM STACK SIZE = 0032H     50D
     702 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```