PHD

The improvement of an automatic procedure for the digital simulation of hydraulic systems

Hull, Stephen Robert

Award date:
1986

Awarding institution:
University of Bath

THE IMPROVEMENT OF AN AUTOMATIC PROCEDURE FOR THE DIGITAL SIMULATION OF HYDRAULIC SYSTEMS

Submitted by

STEPHEN ROBERT HULL

for the degree of Ph.D.

at the University of Bath

1986

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

SUMMARY

The work described in this thesis is concerned with the development of a computer software package to simulate the behaviour of hydraulic systems and their associated electronic and mechanical components. The primary requirement was that the package should be suitable for use by engineers with little or no computing background. It should attract users who would not otherwise come into contact with computers, but who often stand to gain the most from their use. The second requirement was that it should be able to simulate the steady state and dynamic behaviour of as wide a range of circuits as possible.

The resulting CAD package is known as HASP (the Hydraulic Automatic Simulation Package). The user of the package need not produce any computer code. All that is necessary is to define the components in the circuit and indicate the manner in which they are connected. This differs from simulation languages which require the user to become familiar with a vocabulary of mnemonics representing the mathematical components of an algorithm rather than the hydraulic components of a circuit.

A significant amount of programming had already been carried out at the commencement of the current work. However, at that time, the software failed to meet the fundamental requirements. It was the author's intention to examine the package from two different viewpoints:

1. from the point of view of the user and
2. from that of the program developer.

To the target user, the original package appeared unfriendly and complex, and required a detailed knowledge of the computer operating system. The author has erradicated these problems by developing a new command interpreter, together with other enhancements such as simultaneous simulation and graphical display of results.

The structure of the package is such that it can continue to expand and broaden its areas of application. However, this growth was hindered by cumbersome modelling methods, which lacked any classification. These vague methods have been critically examined and developed. The author has defined a classification scheme for modelling methods with firm guidelines for future modellers. In addition, modelling tools such as valve port area calculations and polynomial regression algorithms have been developed.

On completion of this work, the simulation package presents itself to the user as a syntactically simple system, but provides sufficient growth paths for the program developer.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

A,B	Coefficients of a differential equation
A_A	Actuator annulus area in m
A_P	Actuator piston area in m
a,b,c,d	Coefficients of a least squares analysis
F_D	Coulomb friction in N
F_{EXT}	External force applied to actuator rod in N
f_V	Viscous friction coefficient in Ns/m
f_ω	Windage coefficient in Ns^2/m^2
J	Moment of inertia in kgm^2
m_e	Effective mass in kg
P_A	Actuator annulus pressure in N/m
P_P	Actuator piston pressure in N/m
Q	Flow in l/s
s	Sum of the squares of the errors
t	Time in s
v	Velocity in m/s
x	Displacement
ε_T	Out of balance torque in Nm
ω_E	Engine speed in rpm

CHAPTER 1

INTRODUCTION

THE SIMULATION OF HYDRAULIC SYSTEMS

100. The design of fluid power systems, along with the requirements of most other technologies, has become more demanding. The systems must be both efficient and reliable in order to reduce the running and maintenance costs to a minimum. Also, the systems must provide the often demanding level of control required. Nowadays, designs must be practicable and competitive if the contracter is to stay in business.

101. The designer of hydraulic systems may be required to perform dynamic as well as steady state analyses on the systems he considers to be feasible. The options open to the designer are to perform classical paper calculations or to employ a computer to perform a simulation of the response of the circuit in either the time or the frequency domain. In the past, the use of digital computers has been neglected for two main reasons, these being firstly, the fact that computers were expensive items of equipment and secondly, that a great deal of specialist knowledge and familiaristaion was required in order to use them.

102. With the advent of the micro-electronic chip, the cost of computers reduced dramatically. Conversely, their power has become

significantly greater. However, even with inexpensive computing facilities, it is still difficult to use the computer efficiently due to the experience and time required in the production of a simulation program for a particular hydraulic circuit.

103. The purpose of this work is to develop some aspects of a computer software package which allows both experienced and inexperienced computer users to produce pertinent simulation programs economically in both time and effort.

THE HISTORY OF COMPUTER SIMULATION

104. At this point, it is useful to examine the wealth of experience that has been gained over the years in the use of the computer in simulating physical systems. Computer simulation began over thirty five years ago following the advent of the analogue computer. When using such a device, the mathematical model describing the system to be simulated is depicted as a block diagram. The diagram consists of symbols which represent mathematical functions to be simulated by electrical components. For example, rectangles represent arithmetic functions and the circles represent generators.

105. In 1955, following the introduction of the digital computer, R.G. Selfridge produced the first of a large number of so-called digital analogue simulators. These were programs written for digital computers, or more specifically, they were digital representations of sets of analogue elements. The elements normally appeared as

subroutines or functions. In the case of an analogue simulation, the elements of the circuit were normally connected using a patch-board. Similarly, users of a digital analogue simulator would be expected to write main segments linking the subroutines and functions in the required manner. The simulators that followed developed in terms of the complexity of systems that could be simulated and also in terms of the integration method employed. For example, the first simulator used Simpson's rule to perform the integration, whilst by the early 1960s, fourth-order Runge-Kutta and fifth-order predictor-corrector methods were being employed by packages such as DYSAC (DigitalIY Simulated Analog Computer) and HYBLOC [1].

106. In 1965, a compiler called MIMIC was written which was closely related to its predecessor MIDAS (Modified Integration Digital Analog Simulation). They differed in two respects. Firstly, MIDAS was an interpreter which translated the user-defined commands into a language called FAP (an IBM assembly language), whereas MIMIC was a compiler which translated the commands directly into machine code. Secondly and more importantly, MIMIC allowed its own commands to be interspersed with Fortran-like algebraic statements. MIMIC was closely followed by DSL/90 (Digital Simulation Language) which not only allowed mixing of DSL and Fortran IV statements but also provided a facility for automatically sorting the statements. The DSL compiler translated the statements into Fortran IV subroutines which are then compiled.

107. The general philosophy of simulation languages has remained

largely unaltered from this time, more recent effort being placed in ease of use, scope of application and the development of efficient integration routines with particular emphasis on step-control. However, since the appearance of SIMULA in 1967, simulation languages have been split into two broad but distinct groups: methods to simulate continuous systems and methods to simulate discrete event systems.

108. A continuous system consists of components which have the ability to interact in a continuous fashion due to an internal or an external excitation. The continuity is generally with respect to time but may also be a function of other system features. Mathematical representation of continuous systems is done with the aid of time (and other) dependent differential equations. Examples of the simulation of continuous systems are the representation of the time (and possibly displacement) dependent pressure transients of an hydraulic circuit as an actuator encounters a sudden increase in load, and the changing stability of an aircraft as it passes through its stalling condition. Since the digital simulation of continuous systems inevitably involves numerical integration and lengthy algebraic manipulation, simulation languages and packages such as those mentioned above tend to be based upon the most popular high level scientific language, Fortran.

109. A discrete event system, as the name suggests, consists of components which interact in a discontinuous or discrete fashion. If an event occurs, the operation of the system may be altered in some

respect. However, the transient behaviour, i.e. the manner in the system moves between states is of no interest. The simulation of discrete event systems generally involves logical decisions and optimisation based upon statistical data. Examples of the simulation of these systems are the response of say a traffic system to a bus breaking down in the rush hour, and the analysis of the cause of an hydraulic actuator sticking using failure mode analysis. Since the digital simulation of discrete event systems often involves logical decisions, simulation programs and packages tend to be based upon high level languages which support rich Boolean algebra features such as Algol and more recently Pascal (e.g. the simulation language Simula [2]). It has been found that the literature concerned with discrete event simulation has little relevance to the development of a dynamic simulation package and an extensive appraisal of past work in this field has not been carried out.

110. The basic problem with simulation languages is that the user must still perform a large amount of mathematical modelling in order to adequately describe his system. This is inevitable if such a general purpose tool is to be used. However, if the package is to be used in limited applications, then it is possible to produce simulation packages which reduce the need for mathematical modelling.

111. The McDonnel Aircraft Corporation released an important package for simulating hydraulic systems in 1977. The package consists of several programs which simulate different aspects of several generalised hydraulic systems. For example, the program HYTRAN [3]

analyses hydraulic transients and the program SSFAN [4] analyses the steady state behaviour of a system. The programs are supplied with additional modules which may be added by the user with few amendments to the coding of the main program. Two problems with these programs have been identified. Firstly, it is necessary to understand the complex structure of the program in order to include additional blocks, a necessary task if the program is to be used to simulate a wide range of systems. Secondly, the data the user must define in order to run the programs is often difficult to obtain. McDonnel Douglas suggest that this is not a burden since the components which require the attention are common to many systems, e.g. DC-10 pumps are used on the Boeing 747, the Lockheed L-1011 and the Airbus A300 [5]. However, the dimensions and characteristics of this pump are unlikely to be applicable to pumps in fields of interest outside aeronautics.

112. In West Germany, a package called DSH (Digital Simulation of Hydraulics) has been developed by Backe et al [6]. This package is intended to have the versatility and a degree of user-friendliness which allows an inexperienced computer user to simulate any hydraulic system. The set of programs which form DSH can be run on relatively small computers (less than 64k bytes of core). The user of the package defines his circuit in terms of 'macro' or 'micro' words. The macro word defines a mathematical model which already exists in the package. A micro word defines a single mathematical operation. By defining a sequence of micro words, it is possible to represent a

model not catered for by the basic package. The package consists of five programs controlled by a coordinating program.

113. The major drawbacks of this package lie in two distinct areas. Firstly, the static nature of the programs which constitute DSH tends to limit the types of models which one can write. Secondly, the user interface is extremely basic; users having to define information in terms of data fields. To quote Gordon and Riesenfeld [7], 'In order to be successful, a (CAD) system must have appeal to the designer - it must be simple, intuitive and easy to use. Ideally, an interactive design system makes no demands on the user other than those to which he has been formerly accustomed through ... design experience'. It would be unreasonable to suggest that a system which requires a user to define a great deal of information in terms of data fields can ever be considered as simple, intuitive and easy to use. The current work aims to provide a system which satisfies the requirements quoted above.

THE HYDRAULIC AUTOMATIC SIMULATION PACKAGE

The aims

114. A package known as the Hydraulic Automatic Simulation Package has been developed at the University of Bath over a period of years. Its aim is to allow a user to produce computer programs which will perform a dynamic simulation of any hydraulic system of his choice. Furthermore, it is desirable that the user need not learn a great deal of the skills normally associated with the use of digital

computers. In particular it is intended that the package should:

(i) not require the user to have any previous knowledge of computing;

(ii) not require the user to spend a great deal of time learning the procedures to be adopted;

(iii) allow a user to produce a simulation program and obtain results in a matter of hours;

(iv) provide a library of components sufficient to build a large number of practical circuits;

(v) provide a standard method of modelling such that if a new component is required, the construction of that model is neither laborious nor error-prone.

A description of the package

115. The manner in which the first three aims were achieved was to produce a program generator together with a comprehensive libarary of models. The program generator creates a set of controlling routines for a simulation program and is subsequently attached to the necessary routines from the model library. Having used the program generator, the user is left with a unique simulation program corresponding to his unique hydraulic circuit. The simulation programs are considered as temporary whereas the program generator is permanent and is the most important program of the package.

116. There is one principal difference between this package and the multitude of other packages that are available. In general, the user

of simulation packages must perform a large amount of mathematical modelling in order to describe his system, a task which is normally unnecessary when using HASP. Accordingly, almost all other simulation packages are actually simulation languages [8,9]. It should be emphasised that HASP is not what one would normally consider a language: it is a complete package of programs, data-bases, libraries and command procedures and requires nothing of the user other than a definition of the layout of his circuit. Therefore, the user need never learn a language and need never directly produce a file; all necessary files being produced by the programs following an interactive session.

117. Since HASP relies upon a fixed library of hydraulic components, an inexperienced user is restricted to simulating circuits described by the components that have already been modelled. (However, the structure of the package and the standardisation of modelling techniques would allow more advanced users to include their own models into the library). Therefore, at the user level, HASP is not as versatile as simulation languages which require the user to develop his own complete mathematical algorithms. However, that is not to say that HASP cannot be adapted to simulate other continuous systems. In fact, it could be adapted to simulate any physical system which can be described in terms of discrete mathematical models. A proviso is of course that these models should be of a form where they may be considered constituent parts of many different configurations of the overall system.

118. It is useful to consider the tasks the user of HASP must carry out, from the initial conception of a hydraulic circuit configuration to being able to examine the behaviour of that circuit. At the same time, it is worthwhile introducing some of the software that has been designed to achieve these aims.

119. Initially, the user must produce a proposal for the hydraulic circuit to be designed. His use of HASP begins with a search through the component model library to find models which suitably represent the components which constitute his circuit. This may be done by consulting the documentation which exists or perhaps more easily by using the online help facility provided by the package. To use this facility, the user must be operating within the package. To enter the package, the user simply types HASP. The online help facility is offered when the user issues the command to generate a new simulation program.

120. Whichever method the user adopts in determining the required components (i.e. online information, written reports or experience), he must produce a description of his circuit in terms of HASP component models and interconnecting links. An example of a simple hydraulic circuit that one might wish to simulate is given in figure 1.1 and the corresponding linking diagram is shown in figure 1.2. The blocks represent component models, the HASP code name of the components being inserted in the blocks. The interconnecting links indicate information transfer between models and should be arbitrarily numbered for reference.

121. The user is now ready to generate the controlling segments of the simulation program. This is done by invoking the program generator (assuming it has not already been invoked to give advice on component model selection), then answering the questions posed. The questions issued by the generator are entirely concerned with the definition of the linking diagram produced by the user and the production of a file to store that information for possible retrieval, amending and regenerating in the future. The program generator employs an efficient method of user-error diagnosis.

122. Assuming the circuit configuration data defined by the user is acceptable, several routines (mostly written in Fortran) are generated. They are then compiled and linked with other necessary routines. These routines are the selected component models, the fluid properties definition routine and the standard integrator. The integrator employed by the simulation program is based on a method developed by Gear [10]. The method is particularly useful in the solution of the differential equations produced in the analysis of hydraulic systems due to its ability to select the most efficient combination of order and integration timestep.

123. The user is now ready to run the simulation program that has been produced. The simulation is best considered as being split into two. The first section is concerned with the definition of the parametric data required by the component models. Again, this process is carried out completely interactively, posing questions relevant to the size and performance characteristics of the hydraulic

components. The information is again stored in a data file which, on subsequent runs of the simulation program, may be retrieved and amended. The first section also calculates all the parameters that will remain constant throughout the simulation. The second section of the simulation program is concerned with the calculation of all time dependent variables. These results are collected in another data file.

124. Finally, the results are viewed graphically using one of the graphics programs associated with HASP. Based on these results, the user can then decide whether or not it is worth considering components with different dimensions. If so, he simply reruns the simulation program, retrieves and amends the parametric data then repeats the simulation process. Alternatively, he may wish to alter the basic configuration of his circuit in which case he returns to the program generator. He retrieves and amends the circuit configuration data and generates a new simulation program with which he can investigate further.

125. It should be emphasised that the time scale involved in carrying out the process above can vary greatly depending on the size of the circuit, the relative values of certain system parameters and of course, the experience of the user. However, at most, the whole process may be carried out in a matter of hours.

The user of the package

126. The hypothetical user that has been described in the preceeding

paragraphs is seen as being at best, inexperienced, and at worst, totally ignorant of computing. However, it must be emphasised that this ignorance should not extend to the design of hydraulic systems. The computer will not carry out the creative thought required in engineering. It is merely a tool to examine the possible solutions quickly and as accurately as the known parametric data will allow. With the aid of HASP, it should be possible to produce a good prototype design, but the prototype and the experimental work should not be cast aside. However, with diligent use, HASP has in the past, and will in the future prove to be a useful and potent tool in the design and trouble-shooting of hydraulic systems.

AN OVERVIEW OF THE AUTHOR'S CONTRIBUTION

127. The aim of the current work is to ensure that the package meets the aims set out in para.114. Put rather more simply, these aims are to present a package which can be used to both produce simulation programs by engineers with no knowledge of computing, and to ensure that modellers may produce new component models as quickly and effectively as possible. To this end, the author has produced several versions of the program generator and also written many component models and standard modelling utilities intended not only to enhance the component model library, but also to indicate new techniques and provide tools for modellers in the future. Also, a command interpreter has been written in order to provide an interface between the user and the computer operating system.

128. Chapter 2 includes a description of this command interpreter as it provides a useful vehicle with which to describe the overall structure of the package in more detail. Its primary purpose is, of course, to explain what the user of HASP will see. Its secondary purpose is to show the interaction of the different programs and files which collectively form the package.

129. Chapter 3 describes the structure of the program generator in more detail, this being the first part of HASP a user will experience. Chapter 4 logically continues with a description of the simulation program. This involves a description of all its constituent parts and therefore covers component models, modelling aids (the special utility routines) and the general structure of the program. Chapter 5 describes how the package was used to simulate a large practical hydraulic circuit. Chapter 6 looks at the management of such a large software package. This includes information on computer file and directory structure, documentation, portability and also mentions certain legal aspects. Finally, chapters 7 and 8 present a discussion and a conclusion of the work carried out.

130. The appendices present additional details on the command interpreter, a selection of component models, the special utility routines, integrator control from component models and several ideas for future work.

Computer operating systems

131. The package was initially developed on a DIGITAL PDP-11

computer operating under the RSX-11M, version 4.0 operating system. Originally, it was attempted to employ standard Fortran IV [11] throughout. However, more recently, the use of Fortran 77 [12] has been encouraged due to its wide acceptance. This has allowed the use of a more structured approach to the development of the code. Nevertheless, certain parts of the package are necessarily system dependent. One important part of the package that is totally system dependent is the routines which write the component selector file. Under RSX, this file takes the form of an overlay descriptor file written in DIGITAL's overlay descriptor language (ODL), obviously incompatible with other operating systems. Also, much of the directory organisation and file manipulation is carried out by command files which issue system commands.

132. Recently, the whole package has been transported to a DIGITAL VAX 11/750 operating under VMS. The transition from one system to the other was trouble free and although these systems are produced by the same manufacturer, it does indicate that transporting the package to any other system is feasible.

133. A special note is made if any part of the software described in the text is peculiar to a particular operating system.

134. The manuscript of the thesis has been prepared using the text formatters DSR operating under VMS and mm/nroff operating under UNIX. It was printed on a Hewlett-Packard ThinkJet printer using a courier 12 type-face.

CHAPTER 2

THE HASP ENVIRONMENT

INTRODUCTION

200. The acronym HASP has been used to describe the software designed to simulate the response of hydraulic systems. However, if we examine the construction of this package in terms of its constituent parts, a host of files are found, each with its own particular role. The software consists of files containing source, object code, executable task images, data and commands. Some of the files are accessible by all users, others are stored in secure directories. Some of the files must always exist, others are transient. Some of the programs must be used on specific terminals, others are terminal independent.

201. In the early development of HASP, it was necessary for the user to acquaint himself fully with this vast array of files. He would have to learn all the commands required by the operating system in order to control the creation and storage of files. Also, there would be no safeguard against him accidentally misusing the system.

202. It became obvious that, in order to produce a system which is inviting to the computer-layman, a user interface was highly desirable. It was decided that such an interface should remove the necessity of the user becoming directly involved in issuing system

commands. The commands required to operate HASP should be single words reminiscent of English. This interface has been designed, written and implemented by the author on the VAX-11/750 operating under VMS. It is an interpreter which takes a user-defined high level command such as "SIMULATE" or "DRAW" and translates the requirement into a series of commands native to the operating system. The interface is written in a language known as DCL (Digital Command Language). With this interface, the user is led through HASP and, if required, he is given guidance and advice. The command interpreter controls the constituent parts of HASP to present a structurally cohesive, but more importantly, a practicable package.

203. The general architecture of HASP is shown in figure 2.1. The figure is split into two distinct sections: one section shows the tasks a user must carry out in order to simulate a circuit, the other shows the corresponding tasks the computer must carry out. A description of the structure of the command interpreter is a convenient manner in which to introduce the various tasks and procedures which are collectively known as HASP. Figure 2.1 will be found of use throughout the following description.

THE HASP COMMAND INTERPRETER

204. The HASP command interpreter (HASP-CI) has been implemented using the command procedures known collectively as the Digital Command Language (DCL). Therefore, it must be understood that references to specific system commands apply only to operating

systems which support DCL (i.e. VMS and more recent versions of RSX-11). However, it is possible to emulate HASP-CI on most other modern operating systems.

205. Two levels of commands exist under HASP-CI. Having typed a level 1 command, it may be possible to type subsequent commands in order to complete the instruction. The commands available under the first level of the interpreter are as follows:

GENERATE	Generate the source of the simulation program
LINK	Produce the simulation task
SIMULATE	Run the simulation program
DRAW	View the simulation results
EXIT	Exit from the HASP command interpreter
BATCH	Submit a simulation program to run on batch
HELP	Obtain more information about these commands
VMS	Have the ability to type VMS commands

In all cases, it is merely necessary for the user to type sufficient characters to make the command unambiguous. For the commands shown above, this means that only the first character need be typed.

The GENERATE command

206. The command GENERATE invokes the current standard HASP program generator. (The term standard is used since more than one program generator exists. Non-standard generators have been written for specific applications, further details of which are given in Chapter 3.) In order to use the program generator, the user must have

available sufficient information to describe the hydraulic circuit he wishes to simulate. This data takes the form of a list of HASP component model names together with link numbers defining the connections between these models. The program generator produces several files which will form the basis of the simulation program. Both the program generator and the simulation program are described in detail in the following chapters.

The LINK command

207. In simple terms, the command LINK takes the generated segments and creates a simulation program from them. Specifically, the tasks carried out are as follows:

1. Files created by previous generations are deleted and parametric and results data files are maintained for the three most recent versions.
2. The generated source of the simulation program is compiled.
3. The compiled version of the generated source is linked with the required component models from the library and the standard integrator to form the executable task image of the simulation program.

The SIMULATE command

208. The command SIMULATE runs the most recently produced simulation program. If it is the first time a simulation program has been run, then the user will be required to interactively define the parametric data for every component in the simulation. On completion of the

first run, this data is stored in a file on disc. On subsequent runs, the data may be retrieved and amended by the user as required. Furtther information on the simulation program is given in Chapter 4.

The DRAW command

209. The command DRAW allows the user to run a graphics program in order to view the results of simulation runs. The user effectively enters another level of HASP-CI commands. He now has the option of running one of four different graphics programs. These programs, together with the commands associated with each, are displayed every time the user enters the command DRAW and are listed below.

1. ONE Plots any item of information from the latest results file against time.
2. XY Plots any item of information against any other item of information, both from the latest simulation.
3. TWO Plots any two items of information from the latest results file against time.
4. UPDATE Plots the same item of information from two consecutive simulations against time.

The EXIT command

210. The command EXIT returns the user to the normal operating system.

The BATCH command

211. The command BATCH invokes a further command procedure which allows the user to run a simulation program as a background task. By

doing so, the issuing terminal is left free. This command is useful when a simulation program has been tried and tested and one or more sets of results are required.

212. The procedure requires the user to define the job to be submitted to the batch queue. This entails the definition of the total time for which the simulation is to be carried out and the print interval. The user is also given the options to set a CPU time limit of one hour on the job (useful in the case of a logical error) and to request that the job be held until after 1700 hrs, i.e. outside the period of maximum usage of a multi-user operating system. (The latter option is obviously not given if it is already after 1700 hrs.)

The HELP command

213. The command HELP gives on-line assistance to the user by producing a list of possible commands and describing any selected command in detail. Each description consists of a general overview of what the command does and also details of the system commands which will be issued by that particular HASP command.

The VMS command

214. The command VMS allows the user to type standard operating system commands without leaving the HASP-CI. This is useful for more experienced users who wish to carry out functions not catered for under the interpreter.

CHAPTER 3

THE PROGRAM GENERATOR

AN INTRODUCTION AND AN EXAMPLE

300. The program generator is a program which takes a user defined hydraulic circuit and using this information, produces a corresponding simulation program. Specifically, it produces four Fortran source files which form the main segment and three controlling subroutines of the simulation program together with a selector file which, in effect, instructs the component library which models are to be attached to the controlling segments. The form of this selector file is totally dependent upon the computer operating system being used. It will be described in detail in paras. 315 to 342.

301. As an example, consider the simple open loop transmission system shown in figure 1.1. The first step the user must take is to select appropriate HASP component models. These models normally represent their hydraulic counterparts on a one-to-one basis. The schematic block diagram of the circuit in terms of these component models may then be constructed as shown in figure 1.2, the interconnecting links being numbered arbitrarily. It should be emphasised that the links in no way represent physical components such as pipes or shafts but merely indicate an exchange of information between two models.

302. Until recently, the development of this linking diagram was carried out manually i.e. the user would have to search for suitable components by reading the literature associated with the model library and then draw his own diagram. However, the author has added a "help" routine to the program generator where lists of components of a given type are presented in the form of a menu and additional information subsequently presented for any particular model should it be required (to be described in para.311).

303. The final task the user must complete in the manual method of linking, is to convert the linking diagram into a table of information in a format acceptable to the program generator. The table of data corresponding to figure 1.2 is shown below.

```
09
TK0001 01
PU0001 01 02 03
PM0001 03
PI0501 02 04 05
PC0101 04 06
TK0002 06
MO0001 05 07 08
LR0001*08
TK0003 07
```

304. The first line indicates that there are nine component models in the circuit. The remaining lines list the component models in an arbitrary order and define the links between them. Each line consists of the four character mnemonic for the component model, the two digit identifier to indicate multiple occurrences of the same model, and the external links in the form of two digit numbers separated by single blank spaces. The asterisk next to the LR00

entry indicates that this model is experimental and will be found in the user's own directory rather than in the standard component model library (directory structure is described in detail in paras.604 to 620).

305. At this stage, the user is ready to employ the program generator. The information outlined above is defined in a simple interactive manner, the user being given the chance to correct typing and logical errors. The generator employs a sophisticated algorithm to check the validity of defined data and displays diagnostic error messages as necessary. A file produced by the developers of component models, termed the component model attributes file, is interrogated at various stages of the generation procedure in order to aid in checking the validity of the data and also to set up the order and form of call statements to the component model subroutines. Provided the defined data is acceptable, four Fortran files and one component selector file are produced. At this point, the program generator has completed its task and it is now necessary to link the generated segments to selected segments from the component model library.

THE STRUCTURE OF THE PROGRAM GENERATOR

306. In total, the program generator alone consists of some three thousand lines. It is unnecessary to give a complete listing of the program generator in this thesis. However, it is worthwhile to list just one of the segments as an example. A routine known as PGHELP

has been chosen since it was written using the structured approach allowed by the language Fortran-77 and also due to its brevity (Table 3.1). It will be noticed that comments in the coding tend to be placed to the right of the statements rather than being interspersed with them. Although this is not allowed under ANSI X3.9-1978 [12], it is recommended that all future program generator software contain this type of commenting since it allows the graphic nature of the indented coding to show through. Should it be necessary to transfer the software in source form to a computer with a compiler which does not allow this form of commenting, then it can be readily removed with the aid of a screen editor or a simple specially written editing tool.

307. The program generator consists of three levels of segments (figure 3.1). The highest is the main segment, its only purpose being to call all five segments of the second level (figure 3.2). The second level of segments make up the primary logic of the generator. The lowest level of segments carry out single specific tasks such as character manipulation, interrogation of the component model attributes file and variable type conversion. Third level segments form a set of thirteen utilities which may be employed by one or more of the second level segments.

308. It is useful to restrict the description of segments to those in the second level since, as mentioned, it is here that the primary logic is based. The tasks of the third level segments are given in Appendix F.100. The second level can be broadly divided into three

functions:

1. The function of the routines known as PGIN and PGHELP is to allow the user to describe his circuit.
2. The function of the routine known as PGCOMP is to check the validity of the circuit so described and to set up an acceptable call sequence.
3. The function of the routines known as PGOUT and PGSEL (or PGODL under RSX) is to write the five files which constitute the control and selector segments of the program generator.

The specific tasks of each of the five second level segments are briefly described below.

Segment PGIN - Definition of the hydraulic circuit

309. The segment PGIN is the first routine to be employed when running the generator. PGIN is essentially the user/generator interface in that the majority of questions and replies are controlled in this segment. Its primary purpose is to obtain all the information necessary to produce the simulation program. The user is allowed to define his circuit interactively or to recall his circuit from an existing data file. If he chooses to define his circuit interactively, then he is given the option of storing the data in a file for further use or reference. If he chooses to retrieve his data from a file, then he is given the option of interactively editing his circuit data and storing in either a new data file or the initial file.

310. There is a great deal of interaction between this routine and a routine PGHELP (para.311) which gives the user aid in selection of appropriate component models. There may also be interaction between this routine and a sorting routine PGCOMP (para.312) should the user have defined an unacceptable circuit. A flow chart for PGIN is shown in figure 3.3 from which this interaction may be better appreciated.

Segment PGHELP - Aid in component selection

311. The segment PGHELP is employed if the user requires information about a single component or a group of components. The purpose of the file is to access the component model attributes file and the information text file called INFORM.DAT. The latter should contain a written introduction to every model in the component model library. It is suggested that this text should be the introduction given in the model report together with the relevant model assumptions. When a developer has completed a new model, he is expected to insert entries into both COMPON.DAT and INFORM.DAT. Figure 3.4 is a flow chart for segment PGHELP. At present, PGHELP is included in the program generator. However, it is feasible to extract this utility and merge it with a program which allows the graphical definition of the circuit (to be described in para.711 and Appendix E.200).

Segment PGCOMP - The sorting routine

312. The segment PGCOMP serves two purposes. Firstly, it examines the attributes of the models with the aid of a utility subroutine, LOOKUP. If the user has defined a circuit which is unacceptable, then it produces a diagnostic error message then returns control to

PGIN to allow the user to reconsider (possibly with the aid of PGHELP). Secondly, PGCOMP uses the information gained from COMPON.DAT to set up an acceptable component model call sequence. The algorithm of this routine has remained unchanged since the original development of HASP. Minor alterations have been carried out by the author in order to maintain compatibility with revised and new routines. A flow chart of PGCOMP is included for completeness and is shown in figure 3.5.

Segment PGOUT - Write the simulation program source files

313. Though the largest of all the generator routines, the segment PGOUT has perhaps the simplest and certainly the most mechanical task. Its sole purpose is to write the four Fortran routines which will control the simulation program ultimately produced, using the information already gained by PGIN and PGCOMP. The coding is completely sequential and, as such, does not require a flow chart. This routine has been substantially altered by the author in order to introduce the simulation program corrections and modifications mentioned in Chapter 4.

Segment PGODL/PGSEL - Write the component selector file

314. This is the part of the generator which is dependent upon the operating system being used. PGSEL is used in the generator which operates under VMS and PGODL is used in the generator which operates under RSX. Their primary objective is to create a selector file for the linker. Under VMS, this takes the form of a simple options file (called CAD.OPT) which merely lists the names of the object files to

be included in the simulation task. However, under RSX, the purpose of the selector (called CAD.ODL) is twofold. Firstly, it carries out the same function as the VMS options file. Additionally, it describes to the linker (called the taskbuilder under RSX) the method by which the simulation program is to be overlaid. CAD.ODL is written in a form of assembler called the Overlay Descriptor Language. Due to the rather complex nature of the routine PGODL and the fact that this routine is system dependent, a complete section is devoted to its structure (paras.315 to 342). This section also serves to introduce the structure of the simulation program (Chapter 4).

THE COMPONENT SELECTOR FILE (RSX)

315. This section describes the selector file written for RSX (i.e. the file written by the generator segment PGODL) since this description also covers the rather more trivial task of writing the selector file for VMS (i.e. the file written by segment PGSEL).

316. The primary addressing mechanism of the PDP 11 is the 16 bit word. The maximum physical address space that the PDP 11 can reference at any one time is 177777 bits (in octal) i.e. the maximum virtual address of a task must be less than 177777. This effectively means that the size of any task or any segments of a task in memory at any one time must be less than 32k words.

317. A simulation program produced by the program generation and

subsequent linking pocedures consists of a large number of subroutines and therefore requires a large amount of computer memory. The amount of memory required is evidently a function of the complexity of the hydraulic circuit to be simulated. However, the space required would almost inevitably exceed the 32k words of virtual address space available. This problem is overcome using the principle of overlays.

318. Using overlays saves memory space by reducing the size of the executing section of the task. The task must be carefully designed to have discrete sections which can execute independently of the other sections. These sections reside on disc until they are required thereby saving memory space.

319. An example of where overlaying may be used is given below. Consider a main segment which calls a subroutine which we shall call subroutine A. Subroutine A carries out its function then returns to the main segment. The main segment subsequently calls another subroutine, B. It is evident that subroutine A and subroutine B need not both be in memory at the same time. They are said to be logically independent. A task consisting of this main segment and the two subroutines could be overlaid so that either the main segment and subroutine A or the main segment and subroutine B is in memory. However, had the program been written such that the main segment called subroutine A which in turn called subroutine B, then the task could not have been overlaid.

320. For clarity, this section is divided into two. The first part describes the nature of the overlaying employed by the simulation program and the second describes the structure of the subroutine PGODL, i.e. that part of the program generator which writes the overlay descriptor file. A far more detailed description of the principle of overlaying is given in the RSX-11M task-builder manual [13].

Memory resident and disc resident overlays

321. A principle of overlaying is that any segment which calls other segments must be resident in memory whilst the other segments are being used. It follows that the main segment must always be in memory. This common part of the task is called the root.

322. The RSX-11M computer operating system provides two types of overlaying. One type of overlaid task reads in segments of the program from disc as and when required, overwriting segments previously in memory. This procedure is termed "disc-resident" overlaying. Segments which use the same memory address space must be logically independent. Because segments of a disc resident overlaid task can occupy the same memory space, the overall task size will be smaller than if the task was not overlaid. However, the task may take longer to execute than tasks which are not overlaid since more disc input/output transfers are required.

323. The other type of overlaid task reads in segments of the program from disc as and when required, appending them sequentially

to the root of the task. This procedure is termed "memory-resident" overlaying. A memory-resident overlaid task will take up more memory than a disc-resident overlaid task but will execute faster due to the fact that there are fewer disc input/output transfers.

324. We will now turn to the problem of overlaying the HASP simulation program. The simulation program may be divided into two primary sections. One section must ensure that all the parametric data has been defined either by reading this information from a data file or by asking the user to type in this data interactively. The other section must perform the mathematical simulation of the hydraulic circuit thus defined. These two sections are logically independent provided the parametric data may be transferred from the input segments to the calculation segments via a root. In the case of HASP, disc-resident overlaying is employed since the disc transfer will occur only twice during a standard simulation run (once following the completion of the input section, then once following the completion of the mathematical simulation). Therefore, a great deal of memory space has been saved without making the run time measurably greater.

325. Considering each section in detail, it is found that the root together with only the input routines are often collectively greater than 32k words. Therefore overlaying must also be incorporated within the input section of the simulation program. Again, disc-resident overlays are employed since the additionall time taken to carry out the disc transfers are of little significance when compared

to the reaction time of the user. On the other hand, due partly to efficient programming, the root together with the calculation segments have never been found to be greater than 32k words. Therefore, no overlaying seems to be required within this section.

The overlay tree and memory diagram

326. An overlay tree is a diagrammatic representation of the manner in which a task is overlaid. At the base of the tree is the root segment of the task. Each branch of the tree represents a segment of the task. Parallel branches denote segments which overlay one another and must therefore be logically independent. A path is a continuous vertical route from the root to the end of a series of branches. Any module may call any other modules that exist in the same path.

327. Figure 3.6 shows an overlay tree for a simulation program. The sections are arbitrarily called INPUT and CALCS. The segments contained in these two sections are listed below.

	Subroutine NAME	
(i) INPUT	CONTRL MESAGE FPROP	Collectively called IPERM
	****IN	Overlaid in groups of up to 3 (Groups called IA,IB,..etc - up to IJ)
	****IN General input utility	Single input subroutine with the required general utility (Collectively called IREG1,IREG2, etc)
(ii) CALCS	AUX	

OUT GEAR4 GEAR5 GEAR6 General calculation utility	Collectively called CGEAR	Collectively called CPERM
****	In groups of 6 but not overlaid (Groups called C1,C2,..etc - up C5)	

Note: **** represents the four character mnemonics of the model subroutines.

328. An alternative method of visualising the manner in which the overlaying is carried out is to construct a memory diagram. Figure 3.7 is a memory diagram of a particular simulation program. The rectangle represents the virtual address space available to a task, the base being address zero and the top of the rectangle being address 177777 (32k words). Segments shown vertically above each other reside in memory at the same time. Segments shown beside other segments are overlaid and are logically independent. It can be seen that for this particular simulation program which incorporates ten different component models, the maximum virtual address space required is 26790 words. If the task were not overlaid, over 42k of virtual address space would be required, i.e. a reduction in size of approximately 40% has been obtained using the principle of overlays.

The Overlay Descriptor Language (ODL)

329. The overlay descriptor language is a form of assembly code. Its purpose is to describe to the taskbuilder (the system software

which links segments of a program) the manner in which the overlaying is to be carried out.

330. The first line of the file must be a line termed .ROOT which informs the computer of the general structure of the overlay. It must start with a main section which is always in memory and must list the remaining sections of the task. These may be either program file names or arbitrary names (termed factors) which are defined in terms of file names later in the overlay descriptor file. The first line of the overlay descriptor file written by the program generator is always

```
.ROOT  MAIN-*(INPUT,CALCS)
```

A hyphen means that the following factors will be stacked above the preceding segment. The asterisk means that the following factors will be overlaid in a disc resident fashion. The comma means that the two factors INPUT and CALCS will overlay each other.

331. The files or factors which make up INPUT and CALCS are defined on named lines which must contain the term .FCTR following the name. Therefore, for the overlay descriptor file to define CALCS, the following lines must exist:

(i) The group of files which make up the integrator.

```
CGEAR: .FCTR GEAR4-GEAR5-GEAR6
```

(ii) The group of calculation subroutines whose names are independent of the particular hydraulic circuit to be simulated, including those named in the factor above.

```
CPERM: .FCTR AUX-OUT-CGEAR
```

If any of the models require the standard port area subroutines PTI/PTC (to be described in paras.437 to 438) then -PTC is appended to CPERM.

(iii) The groups of model calculation subroutines. Each factor contains up to six model file names and the overlay descriptor file will include only as many factors as are necessary.

e.g.

```
C1: .FCTR AB01-AB02-AB03-AB04-AB05-AB06
C2: .FCTR AB07-AB08-AB09...etc
```

where AB0n refers to a model mnemonic.

(iv) Finally,the line which describes the whole calculation section of the task.

```
CALCS: .FCTR CPERM-C1-C2...etc
```

332. Similarly in order to define the input section of the task, the following lines must exist:

(i) The subroutines which control the model input subroutines, i.e. CONTRL and MESAGE (to be described in paras.434 to 436)

```
IPERM: .FCTR CONTRL-MESAGE
```

(ii) The groups of model input subroutines. Each factor contains up to three model file names and the overlay descriptor file will include only as many factors as are necessary.

e.g.

```
IA: .FCTR AB01IN-AB02IN-AB03IN
IB: .FCTR AB04IN-AB05IN-AB06IN
IC: .FCTR AB07IN...etc
```

However, if any model requires the use of certain general purpose subroutines, then this model input file name would be extracted from its position shown above.

e.g. Model input subroutines AB05IN requires the use of the regression subroutines REGR and SIMUL (to be described in para.439). The factor IB above now becomes,

```
IB: .FCTR AB04IN-       AB06IN
```

The file name AB05IN together with the general purpose subroutines would be inserted below the model input factors as shown below.

```
IREG1: .FCTR AB05IN-REGR-SIMUL
```

At present, a maximum of four component models may use these general purpose subroutines in any one simulation.

The reason for extracting this routine from its position in the factor is that three input routines plus up to three general routines in the same factor may cause the task to attempt to address more than the 32k words allowed.

(iv) The following factor groups together the fluid properties

definition section and all the model input factors.

IMOD: .FCTR (FPROP,IA,IB,IC,...etc)

(v) Finally, the line which describes the whole input section of the task.

INPUT: .FCTR IPERM-(IMOD,IREG1...etc)

333. The end of the description must be defined by an end line (.END).

334. An example of a complete overlay descriptor file is shown below. The simulation for which this file was written has eight different component models, one of them requiring the general purpose regression subroutines. The simulation is in fact described in paras.516 to 518.

```
          .ROOT  MAIN-*(INPUT,CALCS)

CGEAR:    .FCTR  GEAR4-GEAR5-GEAR6
CPERM:    .FCTR  AUX-OUT-CGEAR
C1:       .FCTR  TK00-PI2Z-OD1Z-DE01-AL3Z-FC3Z
C2:       .FCTR  PI05-PI06
CALCS:    .FCTR  CPERM-C1-C2

IPERM:    .FCTR  CONTRL-MESAGE
IA:       .FCTR  TKOOIN-PI2ZIN-OD1ZIN
IB:       .FCTR  DE01IN       -FC3ZIN
IC:       .FCTR  PI05IN-PI06IN
IREG1:    .FCTR  AL3ZIN-REGR-SIMUL
IMOD:     .FCTR  (FPROP,IA,IB,IC      )
INPUT:    .FCTR  IPERM-(IMOD,IREG1    )

          .END
```

335. The names used and the layout incorporated is largely arbitrary, being chosen merely for clarity.

336. The format of the overlay descriptor file is slightly changed if the computer library version of the package is used (a version where all component model object code is contained in a single object module library). There are two features which must be revised.

(i) Previously, the terms subroutine name and file name have been interchangeable since, in general, they are the same. The names which appear in th overlay descriptor file shown above inform the taskbuilder which models must be found in the disc, i.e. They are file names. However, if an object module library is used to store the object code of component model subroutines, then the taskbuilder must be informed of the filename of the object model library and the names of the object modules required, i.e. The subroutine names.

(ii) For subroutines contained in an object module library, the hyphen separator is replaced by a colon (:).

The structure of the program generator segment PGODL

337. The subroutine PGODL described in this chapter automatically writes the overlay descriptor file outlined previously for any hydraulic circuit during the program generation stage. The component model subroutines required to form the simulation program are, in this case, assumed to be resident in their own files on disc and not in an object model library. The version of PGODL which writes overlay descriptor files for use with an object model library differs only in minor details from the version described here. Only a general explanation of the algorithm of the subroutine is necessary rather than a line by line account.

338. Careful inspection of the example of an overlay descriptor file given above shows that a substantial proportion of the file always exists, no matter what the configuration of the hydraulic circuit to be simulated may be. Therefore, this portion of the file may be written simply using Hollerith or literal strings. The remainder of the file is stored in a series of arrays of type LOGICAL*1 i.e. arrays containing short integers where one element of the array contains one byte (representing characters).

339. The general structure is best understood with the aid of the flow diagram shown in figure 3.8. An explanation of the flow diagram is given below.

(i) The number of different component models, NOCDS, must be calculated from COMP(N), which contains all the components required for the particular hydraulic circuit to be simulated.

(ii) If the total number of components is greater than 30, then write an error message on the terminal. Also, the flag NBIG initialised as zero in the main segment PGMN is set to 1. Control is then returned to the main segment and the program generation is aborted.

(iii) If any model requires the use of the general subroutines PTI/PTC (port area routines, to be described in paras.437 to 438), set IPT to 1.

(iv) If a model COMP(I) requires the use of the general subroutines REGR/SIMUL (regression routines, to be described

in para.439), set IREG(I) to 1.

(v) If more than four different component models require the use of the regression routines, then write an error message on the terminal. As in the case outlined in step 2 above, the flag NBIG is set to 1 and the program generation is aborted.

(vi) The LOGICAL arrays are cleared and counters set to zero.

(vii) Set the variable NFCTR (number of components still to be dealt with) equal to NOCDS.

(viii) Define the LOGICAL arrays for up to 6 components. (This operation is discussed more fully later).

(ix) Calculate the number of components still to be dealt with by subtracting 6 from NFCTR.

(x) If NFCTR is less than or equal to zero, then the overlay list is completed.

(xi) If NFCTR is positive, return to step 8. and repeat the process for the next set of up to 6 components.

(xii) When all components have been inserted into the LOGICAL arrays, then write literal strings and these arrays into logical unit 12 (CAD.ODL).

340. The majority of this subroutine is self-explanatory. However, step 8 above (definition of the LOGICAL arrays) is an important feature of the program and will therefore be explained in more depth. Figure 3.9 shows a flow diagram for this section of the subroutine. It must be emphasised that the diagram arbitrarily shows the definition of the LOGICAL arrays for the second set of six components. For the third set, for example, NLOOP2 becomes NLOOP3,

C2 becomes C3 and so on. The flow diagram is described in detail below.

(i) The variable NLOOP2 is set to NFCTR, the number of components left to be dealt with.

(ii) However, if NFCTR is greater than 6, then NLOOP2 equals 6. The excess components will be dealt with by the sections which define C3 etc.

(iii) Insert C2 into the array CALCS and insert IC into the array INPUT.

(iv) If NLOOP2 is greater than 3, then also insert, ID into the array INPUT.

(v) Call subroutine XTRACT(X). This subroutine extracts the name of a particular component from the array COMP defined elsewhere in the program generator. This name is assigned to the four element LOGICAL*1 array X.

(vi) Call subroutine CNAME(C2,X,I). This subroutine inserts the current array X into the correct position in array C2 together with the required hyphen.

(vii) Call subroutine INAME(IC,X,I). Similarly, this subroutine inserts the current array X into the correct position in array IC together with a hyphen preceding the entry and "IN" following the entry. (The characters "IN" indicate that the file thus named contains an input subroutine). However, if the current component is the fourth, fifth or sixth of the current set of 6, then call subroutine INAME (ID,X,J) where J=I-3. This will insert the array X into the correct

position in array ID.

(viii) Repeat the procedure from step 5. above for all components in the current group i.e. for I=1, to I=NLOOP2.

Limits on number of components

341. At present the maximum number of different components that can be dealt with by PGODL is thirty and the maximum number requiring the use of regression subroutines, four. These limits are completely arbitrary and either limit may be extended relatively simply. However, it should be remembered that the total number of components which make up the circuit being simulated cannot exceed fifty. Increasing this limit would require changes not only to subroutine PGODL, but also to the other subroutines of the program generator.

342. Whenever the overlaying procedure is altered for any reason, it is advisable to examine the structure of a simulation program with the aid of a map file (called CAD.MAP). The map file is created during the taskbuild procedure [13]. With the aid of a map file, one can construct a memory diagram for the simulation program and therefore ensure that the desired refinement has been successfully incorporated.

THE COMPONENT MODEL OBJECT MODULE LIBRARY

343. The reason for producing a version of the program generator which is compatible with a software library of component object modules (called COMPON.OLB) is concerned with the management of the

package. It is sufficient to note that this version of the program generator does exist and the difference between this and the standard generator is solely in the format of the overlay descriptor file (CAD.ODL). An example of an overlay descriptor file created by this non-standard version is given below. (This file would build an identical simulation task to that formed by the example given in para.334 above.)

```
            .ROOT     MAIN-*(INPUT,CALCS)

  CGEAR:    .FCTR     COMPON/LB:GEAR4:GEAR5:GEAR6
  CPERM:    .FCTR     AUX-OUT-CGEAR
  C1:       .FCTR     COMPON/LB:TK00:PI2Z:OD1Z:DE01:AL3Z:FC3Z
  C2:       .FCTR     COMPON/LB:PI05:PI06
  CALCS:    .FCTR     CPERM-C1-C2

  IPERM:    .FCTR     CONTRL-COMPON/LB:MESAGE
  IA:       .FCTR     COMPON/LB:TK00IN:PI2ZIN:OD1ZIN
  IB:       .FCTR     COMPON/LB:DE01IN       :FC3ZIN
  IC:       .FCTR     COMPON/LB:PI05IN:PI06IN
  IREG1:    .FCTR     COMPON/LB:AL3ZIN:REGR  :SIMUL
  IMOD:     .FCTR     (FPROP,IA,IB,IC                )
  INPUT     .FCTR     IPERM-(IMOD,IREG1              )

            .END
```

THE COMPONENT MODEL ATTRIBUTES FILE AND ITS EDITOR

The attributes file COMPON.DAT

344. The component model attributes file COMPON.DAT is the keystone to the program generator. It is a database which contains information about every model in the component model library. It describes the manner in which a model may be permissibly linked to other models, it notes the derivatives and state variables of the model (if applicable) and it describes the form of the argument list

of the model. Every time a new model is produced, its developer must insert a corresponding entry into COMPON.DAT.

The COMPON.DAT editor

345. During the development phase of a model, the component attributes file is continually being amended making this important file open to accidental corruption. In a multi-user system, such corruption can be catastrophic leading to incorrectly linked simulations (often affecting models other than the one whose entry contains the error). Therefore, it was seen as necessary to safeguard the database by ensuring that all entries are vetted by a utility. This was achieved by the development of an interactive editor.

346. The heirarchical structure of the COMPON.DAT editor (CDE) is similar to the program generator. There are three levels of segments as shown in figure 3.10. The first level is the main segment which simply determines the user requirements and calls the appropriate segment accordingly as shown in the flow diagram in figure 3.11. The second level controls all primary functions. The third level is a set of five utilities which are used by level two segments. Unlike the generator, it is useful to describe both level 2 and level 3 segments in some detail.

The segment CDEX

347. The routine CDEX examines an entry in the database COMPON.DAT and, writes this information to the screen (figure 3.12). Initially,

it enquires which component is required and checks the validity of the mnemonic and if necessary, calls the error diagnostic utility CDERR. Having set the locate flag, it moves the pointer to the required entry with the aid of the utility CDLOC. Finally, it displays the information contained in COMPON.DAT on the screen in a readable form with the aid of the utility CDOUT.

The segment CDED

348. The routine CDED edits an existing entry in COMPON.DAT (figure 3.13). Having determined the name of the component and checked its existence, CDOUT is employed to list the information on the screen. The user is invited to change specific items of data in an interactive manner, calling the utility CDQES to ask specific questions as necessary. Finally, the utility routine CDCOM is called in order to write the updated version of COMPON.DAT to disc.

The segment CDAD

349. The routine CDAD adds a new entry to COMPON.DAT (figure 3.14). It is similar in construction to CDED except that the utility CDQES asks all questions rather than a selection. Again, the utility CDCOM is called in order to write the updated version of COMPON.DAT to disc.

The segment CDDEL

350. The routine CDDEL simply deletes an existing entry in COMPON.DAT. The only utility routine used is CDLOC in order to locate the required entry. Due to its simplicity, a flow diagram for

CDDEL is not required.

The segment CDCOM

351. The routine CDCOM is a level three routine which writes an updated version of COMPON.DAT and it is employed by any level two routine that requires a change be made to the existing database. Again, a flow diagram is unnecessary.

The segment CDLOC

352. The routine CDLOC locates an entry in COMPON.DAT (figure 3.15) and is employed by every level two routine. If the required component is found, it sets the locate flag. Otherwise, the locate flag remains unchanged and the level two routine acts accordingly.

The segment CDOUT

353. The routine CDOUT writes an entry onto the screen in a readable form. It is employed by the level two routines CDEX, CDED and CDAD.

The segment CDQES

354. The routine CDQES carries out an interactive session with the user in order to determine the new or updated COMPON.DAT entry required. It is employed by the level two routines CDED and CDAD. If the user is editing an existing entry, then specific questions will be posed as dictated by the routine CDED. If a new entry is being added, CDQES poses all questions that are necessary (some being dictated by the user's response to previous questions).

The segment CDIN

355. The routine CDIN writes a complete entry into COMPON.DAT. This routine is only used by the level three utility CDCOM when creating a new version of COMPON.DAT.

The segment CDERR

356. The routine CDERR is used universally to display an error diagnostic. The utility is called from any routine where the user may have made an unacceptable reply. A flag is transferred to CDERR in order to point to the particular diagnostic message required.

The form of the file COMPON.DAT

357. The type of file used to store the component attributes has traditionally been a formatted sequential file since the file was constructed using the standard operating system editor. Now that access to the file has been automated by the development of CDE, a better form for COMPON.DAT would be the indexed type of file structure. This allows data to be stored in formatted "pockets" and also allows specific records of these pockets to be designated as keywords. A file of this type is obviously the most suitable for our application. Unfortunately, this type of file is not standard though it is available under VMS and also under later versions of RSX (provided the Record Management Services utility, RMS-11, has been built into the operating system during the system generation). Due to the non-standard nature of this file, its use has been regretfully avoided.

358. COMPON.DAT, at present, is still in the form of the simple formatted sequential file. This means that every time a change is made via CDE, a new version of COMPON.DAT must be created, an annoying and time consuming process.

359. A more efficient method of storing this information would be to use two direct access files where the information is stored in binary. The first file would contain the names of all the models in the library together with a pointer. This pointer is the record number of the second file where the attributes of the model may be found. This is effectively creating a simple indexed file using standard file structure. Figure 3.16 is a schematic of this proposed file structure.

A REVIEW OF THE WORK CARRIED OUT BY THE AUTHOR

360. In the case of the COMPON.DAT editor, the requirement was identified and the software developed solely by the author. The same is true for the object library software described in para.343. However, in the case of the standard program generator, matters are not that simple. It must be appreciated that when several people work on the same project (albeit different aspects of that project), it is difficult to accurately quantify the overall effect of any single developer. One could simply look at how much of the software can be contributed to a particular member of the team but that would be to ignore the influence, that and other work might have on the remainder of the package. Since the author is evidently unable to

give a totally objective view of the impact of the work, it will have to be sufficient to simply examine the coding developed.

Program generator level one segments

361. The main segment has been rewritten in order to accommodate modifications to the remainder of the generator. However, its basic function has remained (and should always remain) unchanged. That is, to simply arrange the calling of level two segments with the aid of error and user requirement flags.

Program generator level two segments

362. This is certainly the area where the most important work has been carried out. The segments are listed below together with a description of the amendments or reconstructions made.

The segment PGIN has been completely rewritten.

The segment PGHELP is a new addition.

The segment PGOUT has been significantly amended in order to introduce the modifications outlined in Chapter 4.

The segment PGCOMP has remained largely untouched, the only changes being to maintain compatibility between this routine and other rewritten routines or new additions.

The segment PGODL has been completely rewritten.

The segment PGSEL is a new addition and was written to allow HASP to be used under VMS. It is similar in structure to its RSX counterpart, PGODL.

Program generator level three segments

363. Like the main segment, the thirteen level three segments have remained unchanged, other than for minor amendments to utilites such as LOOKUP to cater for the changing form of COMPON.DAT. As stated previously, these routines are merely "tools" for the level two segments and their development should not be contributed to the author.

CHAPTER 4

THE SIMULATION PROGRAM

THE STRUCTURE OF THE SIMULATION PROGRAM

400. Having described the basic principles of the program generator, it is now useful to look in more depth at the simulation program it produces. This section looks at the program from two different angles. Firstly, it is examined from the user's viewpoint. Secondly, it is worthwhile to examine the different segments of the program.

401. Figure 4.1 shows the interaction between the user, the simulation program and the associated data files. When a user runs a particular simulation program for the first time, it is necessary for him to define a great deal of data relating to the construction and/or operation of each component model in his circuit. Whenever it is thought likely that a user would not be able to supply a particular item of important information, the simulation program gives advice and, in some cases, is able to retrieve parametric information from a database. Figure 4.2 shows a typical set of questions the user may be asked together with his response shown underlined. At the end of this first simulation run, the parametric data is stored in a sequential data file called PARAM.DAT (PARAMetric DATa). On subsequent runs of the simulation, this information is retrieved and the user given the option of changing

any item(s) of data.

402. The simulation also produces a great deal of numerical data with which the user may examine the response of any item of information which appears on the links. The information is stored in a direct access file called CADRES.DAT (CAD RESults DATa). The user may examine the information either in numerical form or with the aid of one of the graphics programs (see para.209).

403. The reason that CADRES.DAT is termed a direct access file, is that it resides in memory until completion of the simulation (the information being stored in binary form). This type of access is much more efficient than a that with a sequential file since the data is held in memory and no translation into ASCII code is required.

404. A further option developed by the author is to view the simulation results as they are being produced. This allows the user to gain some feedback by interrupting the simulation in order to investigate the results. He may then choose to continue or abandon the simulation. This graphics/simulation interaction facility is considered an important part of a simulation package. However, the structure of its software is complex. As such, a complete section is devoted to its description (paras.446 to 455).

405. From the structural point of view, the standard simulation program is not especially complicated. Figure 4.3 is a schematic of the standard simulation program. As can be seen, the program is broadly divided into two. One half of the program is concerned with

the definition of the parametric data, the other with the production of the results. The program has been structured in this manner in order to trim the calculation coding to a minimum. It is desirable that the whole of the calculation coding be resident in memory continuously, i.e. there should be no overlaying of the calculation segments. Therefore, all coding not directly concerned with the calculation must be overlaid. Under VMS, this structuring is less critical since overlaying is not required. However, this structure still minimises the degree of page swapping necessary.

406. For the reasons stated above, the parametric definition section is heavily overlaid. Unlike the calculation section where any particular routine may be called tens of thousands of times, routines in the parametric definition section are only called once or twice. Also, the time taken to perform a disc transfer (necessary in the case of disc resident overlays - paras.322 to 325) is of little significance when compared to the reaction of the user.

407. The heirarchy of the simulation program is not as clear cut as the programs described in Chapter 3. The main segment only calls two routines. Firstly it calls routine CONTRL which controls the parametric definition process. It then calls the integrator GEARKC which controls the calculation process. When the calculation is complete, CONTRL is again called in order to determine if there are any further user requirements. Figures 4.4 and 4.5 are flow diagrams for the parameter definition process and the calculation process of the simulation program respectively.

408. An important group of subroutines are the standard utilities. These are a set of routines which may be employed by any of the models and carry out a calculation of a general nature. These routines are described in more depth in paras.433 to 445.

COMPONENT MODELS

409. Once the structure of a simulation package has been finalised, then there still remains the mathematical models that simulate the real system. It could be argued that the mathematical models are the mainstay of the package. If they are considered incorrect or ineffiecient, then no matter how efficient the structure of the software which manipulates these models may be, the package as a whole is worthless. In the case of HASP, the mathematical models are subroutines which simulate the behaviour (dynamic or instantaneous) of the physical components. Each mathematical model consists of two subroutines; one concerned with the definition of parametric data and the calculation of associated constants, the other concerned with the actual time varying calculation.

410. The author has attempted to categorise the component models as shown in table 4.1. It is possible to model a component, say a pressure relief valve, in any of the four methods outlined. It is important to choose a model which adequately represents the operation of the valve in the particular case being simulated. For example, it may be realistic to assume that the dynamic response of the valve is such that it is unimportant compared to other dynamic effects within

the circuit. If this is the case, then the user may opt for an instantaneous model in order to reduce the computational overheads associated with simulations described by a large number of differential equations. On the other hand, the speed at which the valve opens may be of prime importance in which case the user will be forced to use a model which includes differential equations to describe the dynamic behaviour of the valve. Use of this type of model will undoubtedly cause the simulation program to take longer to execute. The other choice open to the user is between characteristic and first principle models. The main question here is "what information about the component is available?". The engineer using HASP to aid initial design is unlikely to know any more information than is available from a manufacturer's catalogue. In this case, a model requiring operating characteristics may be used to good effect. However, an engineer using HASP to aid in the trouble-shooting of an existing circuit may suspect a particular component as being the cause of a problem. In this case, he will require a component model which employs a highly sophisticated analysis of the behaviour of the component accepting the fact that this makes his simulation program relatively bulky and slow running.

411. Many reasearchers have created HASP models during its existence and Tomlinson [14] has produced information on the standard modelling techniques that have evolved. In this thesis, the description of component models is restricted to a selection of those developed by the author and these have been chosen to show important new modelling

techniques. These models are discussed below and a more complete mathematical description may be found in Appendix B.

Actuator/mechanical linkage models AL2Z and AL3Z

412. AL2Z models the behaviour of a double acting linear actuator which operates a mechanical linkage. This model has been included merely as an introduction to another actuator/linkage model, AL3Z. AL3Z models the behaviour of the actuator/linkage shown in figure 4.6. The mathematical model representing the motion of the actuator is identical to the standard HASP actuator model [14]. However, the interest in this model lies in the representation of the mechanical load.

413. The actuator model consists of two first order differential equations which describe the motion of the piston and effectively solve Newton's second law. These are

$$\frac{dx}{dt} = V$$

$$\frac{dV}{dt} = \frac{P_P A_P - P_A A_A - f_v V - f_w V^2 - F_D + F_{EXT}}{m_E}$$

Special care must be taken with the frictional terms in order to account for the difference between the level of stiction and that of coulomb friction. These equations require that the mechanical load be calculated in terms of force and mass reflected to the actuator piston. This calculation, though not complex as far as an engineering principle is concerned, is extremely lengthy in terms of

algebra. The corresponding Fortran coding requires the use of a large number of the standard trigonometric functions. When one considers that in a typical simulation, the calculation routines may be called in the order of 10^4 to 10^5 times, then it follows that the trigonometric functions will be called in the order of 10^5 to 10^6 times.

414. The manner in which this problem was overcome, is of general interest. The information required by the calculation routine was simply the variation of effective mass and external force with respect to piston rod displacement. Therefore, it was decided to calculate these functions before entering the calculation routine and represent them approximately by a polynomial. The function was calculated at one hundred equally spaced postions of rod displacement and these points used in a regression analysis to find a polynomial of a suitable order. The calculation was carried out in the calculation section of the parameter definition routine and a function segment called in order to carry out the regression analysis. The regression analysis is carried out ten times, for polynomial order one to ten inclusive. The two equations describing the applied force and effective mass now become simply

$$F_{EXT} = a + \sum_{i=1}^{n} b_i x^i$$

$$m_E = c + \sum_{j=1}^{m} d_j x^j$$

where n and m are the orders of the polynomials chosen by the user. The sum of the absolute values of the residuals is also calculated. These residuals are displayed in order that the user may select the order of the polynomials which best suit the functions.

415. It is considered that this technique of fitting polynomials to complex functions in order to simplify the simulation may be of use to other models which may be developed. Therefore, the regression function (REGR) and its associated function which solves the sets of normal equations by the Gauss-Jordan elimination method (SIMUL) have been incorporated into the general structure of HASP so that they may be easily employed by future model developers. The two routines are based on algorithms presented by Carnighan et al [15]. Further details of these functions may be found in Appendix C.300 and a complete description of AL3Z is given in Appendix B.100.

Directional control valve model DC4Z

416. DC4Z models the behaviour of a three way, four port directional control valve. This model has been chosen in order to illustrate the principle of generalising a model which may be required for a specific purpose. A model of a tandem centre directional control valve was required but at the time, such a model did not exist in the component model library. Models of a closed centre valve and an open centre valve do exist and normal practice had been to simply amend the coding of one of these models in order to produce the required central configuration. It was decided to adopt a rather more general approach and to write a model which accounts for any possible

configuration of the centre position.

417. The manner in which this generalisation is achieved is by always assuming that all six possible flow paths exist (ports: S to A, S to B, S to R, A to B, A to R and B to R). The flows are calculated for all paths but are subsequently factored by 0 or 1 depending upon whether that particular flow path actually exists in the configuration chosen by the user. This factor is set in the parametric definition routine of the model. The flows are then summed algebraically to give net port flows. This method allows the calculation routine to be completely free from decisions even though the model offers thirteen possible configurations. It does, of course, mean that unnecessary calculations are carried out. However, in this case, this is a small price to pay for the generality offered. A complete description of DC4Z is given in Appendix B.200.

Pressure compensated flow control valve model FC8Z

418. FC8Z models the dynamic behaviour of a pressure compensated flow control valve fitted with an orifice in parallel with the compensating orifice which eliminates the possibility of hunting at high differential pressures.

419. As stated earlier, four methods of modelling components exist. FC8Z is an example of perhaps the most unusual of these methods - the dynamic characteristic model. This model is based on FC3Z which is an instantaneous version of the pressure compensated flow control valve. FC8Z calculates this instantaneous flowrate in an identical

fashion to FC3Z. The differential pressure/flow characteristic is defined in the parametric definition routine leaving the calculation routine to merely compare the differential pressure defined by adjacent models to the characteristic. However, the dynamic model then calculates the "dynamic" flowrate by comparing the current condition of the valve to its previous condition then considering these variables as a demand and assuming the response is a first order lag. The differential equation used is

$$\frac{dQ}{dt} = -A \frac{(\Delta P_n - \Delta P_{n-1})}{(t_n - t_{n-1})} - BQ_T$$

where $\Delta P_n - \Delta P_{n-1}$ is the change in differential pressure during the time interval $t_n - t_{n-1}$.

420. Of course, the user is unlikely to know the values of the two coefficients A and B or even what their effect is in physical terms. Therefore, a chart has been developed which allows the user to read values of A and B given physical information concerning the transient response. A further step would be to code this chart such that the user need never know the existence of the coefficients A and B.

421. When developing a model of this kind, a great deal of care must be exercised in developing the algorithm and it is difficult to make rules that are water-tight. Evidently, this idea of calculating the intantaneous response of a component and then superimposing a first or second order transfer function, is sensible if the instantaneous variable being calculated is displacement (since it is the

differential equation with respect to displacement with which we are attempting to approximate). However, when the instantaneous variable is a co-ordinate of a pressure/flow characteristic, then it is difficult to know what to use as the forcing function of the differential equation.

422. In the case of the pressure compensated flow control valve, a change in differential pressure causes the compensating spool to change position. Therefore, differential pressure is taken as the disturbing parameter which causes a transient response of the flow rate. If one considers a pressure compensated flow control valve which has a perfectly flat characteristic, then an increase in differential pressure would cause the compensating orifice to close. Looking at the instantaneous characteristic, there is no change in flowrate. However, the dynamic response is likely to show an initial increase in flowrate, returning to the instantaneous characteristic as the compenstaing spool takes up its new position. Conversely, if there is a decrease in differential pressure, then the flowrate would show an initial decrease, again returning to the set value when the compensating spool takes up its new position. A more detailed description of FC8Z is given in Appendix B.300.

Synchronous electric motor model GE1Z

423. GE1Z models the dynamic behaviour of a synchronous electric motor. The reason for using this model as an example is to demonstrate a method of writing first principle dynamic models which are not defined by the simple second order equation of motion or the

first order fluid compressibility equation. The model is described in terms of a control block diagram employing the s-operator (figure 4.7). The model may be used as a motor or a generator since the algorithm is valid for a lead or lag condition of the rotor. The model also accounts for the effect of damper windings, friction and rotor and shaft inertia.

424. The algorithm in general terms is as follows:

- Determine the difference between the synchronous speed (the demand) and the current generator speed (the response).
- Integrate this error to get the lead/lag angle and thus the generated electrical torque due to rotor slip.
- Also calculate the effect of the damper windings. The damping torque produced is a linear function of speed error [16].
- Sum these two torques to give the total electrical torque.
- This electrical torque is then used, together with the applied torque (for example from an hydraulic motor) and the frictional torque, in the normal inertia equation to provide the generator speed at the next time step.

A complete description of GE1Z is given in Appendix B.400.

Meter In Pressure Compensator

425. PCDZ models the instantaneous behaviour of a meter in pressure compensator (for example, the Rexroth ZDC pressure compensator). This model calculates the flowrate through the valve algebraically. It does not employ differential equations. However, it does employ a

simple iteration process to calculate intermediate parameters. In many ways, this type of model is often the most complex algebraically.

426. It is intended that the component be used in conjunction with a 4-way proportional valve (such as the Rexroth 4WRZ16). The purpose of the compensator is to maintain a constant pressure drop across the proportional valve thus maintaining a constant flow for a given proportional valve position. Used in this manner, the combination forms a pressure compensated flow control valve with an electrically variable flow setting.

427. The interest in this model lies in the method for performing the force balance on the spool. Firstly, the spool position is calculated assuming static equilibrium and ignoring the effects of momentum force. This defines that the valve is acting in a certain mode. The momentum force can now be included into the force balance and a new position of spool displacement calculated. The valve may still be acting in the same mode (though with a different spool displacement), in which case, the latest spool displacement calculated is acceptable. However, alternatively, the valve may now be acting in a different mode, with the result that an incorrect equation for the equilibrium has been used. The momentum force as defined by this latest operating mode is then included into the force balance and a new spool position calculated. This process continues until the equation used and the result obtained both correspond to the same operational mode. PCDZ is described fully in

Appendix B.500.

Diesel engine model PM3Z

428. PM3Z models the dynamic behaviour of a diesel engine. Like GE1Z, the mathematical model is represented by a control block diagram. The model uses certain equations (such as the equation of motion) which would dictate that this reside in the dynamic first principle category. However, In order to calculate the applied torque, the algorithm requires the definition of a speed torque characteristic (accounting for normal speed droop, maximum torque and motoring). Therefore, PM3Z may be considered a "hybrid model". The differential equation employed is

$$\frac{d\omega_E}{dt} = \frac{60\varepsilon_T}{2\pi J}$$

where ε_T is the difference between the developed engine torque (from the torque/speed characteristic) and the applied pump torque.

429. The model is also included in order to illustrate the manner in which to produce the most versatile linking arrangement. The engine was intended for use in driving a hydrostatic transmission system which incorporated an electronic control system. The control system compared a demand signal to an engine feedback signal controlling the hydraulic motor swash accordingly. It was possible that the control circuit would require angular speed, angular acceleration or torque feedback (or any combination of these). Therefore, PM3Z was developed such that these three items of information were available,

but need not necessarily be connected to the electronic control models.

430. A further feature of PM3Z is that the engine governor setting is variable with respect to time. The governor setting is required as an external input and in general, the model would be connected to an external duty source. Further details of PM3Z may be found in Appendix B.600.

PU0Z Diesel engine/hydraulic pump

431. PU0Z models the instantaneous behaviour of a diesel engine and fixed displacement hydraulic pump combination. If an instantaneous model of a diesel engine were developed and connected to an instantaneous model of a hydrauluc pump, then an implicit relationship between the two models would exist. Therefore, to avoid introducing differential equations, the two components are combined into one model and the equations for hydraulic torque and angular velocity solved simultaneously.

432. However, a problem occurs in the attempt to reduce the equations. The torque speed characteristic incorporates extremely small smoothing regions in order to avoid a discontinuity between operating regions such as the speed droop and the maximum torque characteristics (This technique is applied generally and will be discussed in paras. 440 to 443 and Appendix C). These smoothing polynomials are cubics and, as such, are difficult to work with algebraically. Therefore, an iterative scheme is employed in order

to calculate the position on the speed/torque characteristic at which the engine is operating. PU0Z is described in more detail in Appendix B.700.

STANDARD UTILITIES

433. It was realised by the author that a set of routines was required in order to carry out procedures which occur in several models. These routines, for which the name standard utilities is suggested, eliminate the need for a modeller to develop coding which has been developed in a similar form previously. For example, many first principle valve models calculate the flow area of a port. If the standard routines did not exist, a modeller would be tempted to extract the necessary parts of coding from an existing model - an unnecessary and dangerous process. There are at present, four sets of utility routines available to the HASP modeller.

Unreasonable input data

434. A subroutine MESAGE has been developed to give a warning to users who define parametric data considered to be "unreasonable". The routine has since been employed by all parametric definition subroutines. Whenever a user defines a parameter, then its value should be checked at two levels. Firstly, it should be determined if the value of the parameter would cause a fatal error. This is normally achieved with a simple IF statement. However, a check should also be made to ensure that the value of the parameter is not unreasonable. This might be due to the user not reading the question

correctly or simply due to ignorance.

435. To cite an example, a user was once found entering a value of 500 for the stroke of an actuator thinking that the variable was required in mm. In fact, the question clearly stated that the actuator stroke should be in metres! (This would not be a fatal error but defining an actuator as being 500m long is certainly unreasonable.) In a case such as this, the user should be warned that the defined value is outside normal working limits and he should be given the chance to redefine. However, since the limits specified in the program are somewhat arbitrary, the user should be allowed to continue if he defines the same value twice. If the redefined value is different and still outside these limits, then the value should again be queried.

436. Due to its structured nature, the coding required to carry out this logic is best contained in a subroutine which may be called after each question. The upper and lower limits of the parameter may be defined explicitly in the argument list. A detailed description of MESAGE is given in Appendix C.100.

The flow area of valve ports

437. The subroutines PTI and PTC calculate the flow area of a valve port. The subroutine PTI is called from the parametric definition subroutine of a model and determines the configuration of the port i.e. whether it is an annular port, a circular port or a triangular port and its relevant dimensions. The subroutine PTC is subsequently

called from the calculation subroutine of a model and determines the actual flow area given the port dimensions (including the overlap or underlap of the valve) and also the spool displacement.

438. The use of these routines obviously reduces the time taken to develop, debug and test a model. However, the modeller should be aware of the fact that use of these subroutines (particularly PTC) often constitutes an inefficient use of cache memory (Cache memory is a small, high speed memory that maintains a copy of frequently selected portions of main memory for faster access to instructions and data). Therefore, a simulation program which incorporates a valve model using the logic of PTC will undoubtedly take longer to execute than the same simulation with the port area calculation included as inline coding. Furthermore, the whole principle of modular programming almost inevitably causes cache memory to be used inefficiently. It does, however, form a package which is manageable by users who do not have a large amount of programming experience and expertise. A complete description of these two subroutines is given in Appendix C.200.

The representation of functions by polynomial regression

439. The fuctions REGR and SIMUL perform a two-dimensional regression analysis on a set of data points. The regression analysis is completely standard and produces a polynomial

$$y = a + b_1x + b_2x^2 + \ldots + b_ix^i + \ldots + b_nx^n$$

by minimising the sum of the squares of the errors

$$S = \sum \left(y - a - b_1 x - b_2 x^2 - \ldots - b_n x^n\right)^2$$

It is envisaged that these functions could be used for two different purposes. Firstly, they could be employed by a parametric definition subroutine of a model which requires the definition of a characteristic available from test results. Secondly, they could be employed by a model where a complex calculation is required to define a function. This second case has been described in some detail in paragraph 413 (actuator/load model AL3Z). The arguments of efficient use of cache memory mentioned in paragraph 438 above does not really apply in this case since these functions are a part of the interactive input section of the simulation program where speed of execution is less critical. The functions REGR and SIMUL are described in Appendix C.300.

The smoothing of discontinuous functions

440. The integrator used in the simulation program is based on a method developed by Gear [10]. The method is particularly useful in the solution of the differential equations produced in the analysis of hydraulic systems due to its ability to select the most efficient combination of order and integration timestep. However, the drawback with this method is its inability to cope with derivatives which vary discontinuously with time. The derivative would vary in such a manner if any of the coefficients varied with time. As an example,

consider a simple pressure/flow characteristic of a single stage relief valve. When idealised to two straight lines, a discontinuity in the characteristic exists at the cracking pressure. Therefore, an adjacent pipe would find that the flowrate required in the differential equation which describes the compressibility of the fluid does not vary continuously with respect to time. The integrator logic becomes extremely inefficient at this discontinuity and may even fail [17].

441. The method used to overcome this problem consists of fitting an exact cubic polynomial through an extremely small region either side of the discontinuity. This allows the characteristic to be continuous in function and first derivative. The modeller was expected to calculate this cubic for every discontinuity in first derivative that existed in the characteristics of his models.

442. In order to simplify matters, a standard method of deriving the cubic in terms of a non-dimensional parameter, z, (rather than the independent variable, x) was produced (see Appendix D.100 for the derivation). The derivation of the coefficients of z is simpler than the derivation of the coefficients for a cubic expressed in terms of x. In order to simplify matters further, it follows that a standard routine for calculating these coefficients could be written. However, expressing the equation in terms z dictates that this non-dimensional parameter be calculated on every visit to the model calculation routine and for every discontinuity. Since it was decided to adopt a generalised routine called from the input section

to calculate the coefficients, the extra algebra involved in calculating coefficients of x was irrelevant.

443. The routine CUBIC is called from the calculation section of the parametric definition routine of a model and calculates coefficients of a cubic polynomial which is expressed in terms of an independent variable, x. The routine requires the values of the function and its first derivatives at the limits of the smoothing region. A detailed derivation of the cubic polynomials is given in Appendix D.100 and a description of the routine CUBIC is given in Appendix C.400.

The method of invoking standard utilities

444. It is almost certain that the routines MESAGE and CUBIC will be used in every simulation program produced. Therefore, it was considered unnecessary to expect the modeller to inform the generator whenever the routines are included in a model. These routines are built automatically into every simulation program produced.

445. On the other hand, the routines REGR/SIMUL and PTI/PTC will be used only occasionally and are substantial in length. Therefore, it is necessary for the generator to discriminate between simulation programs which require these routines and programs which do not. This is achieved by the user adding an entry to the model attributes file COMPON.DAT. At present, any one model may use only four standard utilities. This is an arbitrary limit dictated by the logic of the overlay descriptor writer PGODL (see para.341) which may be easily overridden. However, at present, no models in the HASP

library requires the use of more than one (set) of these optional utility routines.

GRAPHICS/SIMULATION INTERACTION

446. A good CAD system allows the user to exercise his potential to the full. His creativity should not be hindered. An important facet of this requirement is that he must get feedback from his simulations as quickly as possible. It follows that it is necessary to write models as efficiently as possible and also to employ an efficient numerical integrator. Another important method is to give the user useful feedback during the running of a simulation. Due to the fact that a great deal of information is required from the simulation of a hydraulic circuit, it is insufficient to list the output in numerical form. The output must be graphical.

447. It may be thought that the simplest manner in which to plot graphs during a simulation would be to add several plotting routines to the end of the output subroutine (OUT). This is not the simple answer it appears since it would certainly require overlaying to be carried out during the calculation section of the simulation. Therefore, it was decided to write a separate graphics program which would lie dormant (in dynamic memory if available) until required. The graphs that are produced are any item of information appearing on an external link against time.

448. It was decided that the most useful graphics program would

remain inactive until the user decided to investigate various system responses. This program was developed and called the "interrupt" graphics program. When the simulation starts, the graphics program writes a prompt to the screen inviting the user to interrupt the simulation. The simulation continues unhindered until the user types the interrupt. At this point the simulation is temporarily halted and the user is able to investigate any of the system parameters. When he has obtained sufficient information, he is able either to allow the simulation to continue or to abort the simulation should the results show unsatisfactory behaviour.

449. The graphics program described above obviously requires a great deal of interaction between the user and the simulation. Therefore, a further graphics program was developed which automatically plots a single graph at user defined intervals of simulation time. To the user, this form of graphics output is identical to that which would be achieved by appending graphics routines to the output routine (OUT). This program is termed the "refresh" graphics program. At the beginning of a simulation, the user is given the option of choosing one of these graphics modes.

450. This form of programming structure requires that two separate tasks should have the ability to communicate. This is achieved by using inter-task communication commands (often termed memory management directives). Important tools used in these programs are the event flags. Event flags are special addresses used by the system to determine whether or not a particular function should be

carried out. Different levels of event flags exist. The local event flags are known only to an individual task. The group global event flags are known to every task operating under user identification codes (UIC) of the same group and are the flags used in the graphics programs described in this section. The global event flags are known by every task in the system. The directives used in this application are those concerned with suspending and resuming tasks and reading, clearing and setting event flags. The logic of the two options are described below in more detail.

The interrupt graphics program

451. Figure 4.8 is a flow diagram of the logic of the interrupt graphics program and the relevant logic of the simulation program. At the start of a simulation, the group global event flags are cleared. This is required in the event of a previous simulation being aborted via a monitor console routine. Assuming the interrupt graphics program is required, the user defines the terminal. At this point, the graphics program writes the prompt onto the screen inviting the user to interrupt or abort the simulation. The graphics program halts at this point waiting for a response. In this state, the program will lie dormant either in dynamic memory, or checkpointed onto disc if many other tasks are active and the space in dynamic memory required.

452. Whilst the graphics program has been carrying out these functions, the simulation program has been active but has been sharing the processor. After every calculation step, the simulation

program reads event flag 72 in order to ascertain whether or not the simulation is to be aborted. If this flag is set, the simulation program closes the results data file then stops. If the flag is still clear, then event flag 70 is read. This ascertains whether or not the user has requested that a graph be drawn. If it is set, then the data file is closed the graphics program resumed and the simulation suspends itself.

453. Whilst the simulation program had been active as described above, the graphics program had obviously set event flag 70, then read event flag 71 to find out if the simulation had finished. If the simulation had not finished, then the graphics program suspends itself until the simulation program has had chance to read event flag 70 and close the data file. When resumed, the data file is opened and the required data read. The simulation is then resumed and time shares until the graphics program is again dormant. The required graphs are then plotted. If the simulation is complete, the graphics then stops. Otherwise, the interrupt prompt is again displayed to the user and the graphics program suspended.

The refresh graphics program

454. Figure 4.9 is a flow diagram of the logic of the refresh graphics program and the relevant logic of the simulation program. In fact, this simulation program is identical to the one described with the interrupt graphics in paragraph 451 above. The logic of the refresh program is simpler than that for the interrupt program since there is only interaction between the two programs rather than the

user creating the three way interaction described above. As for the interrupt program, at the start of a simulation, the group global event flags are cleared. Assuming the refresh graphics program is required, the user defines the terminal and also defines which particular parameter he requires. Whilst these functions are being carried out, the simulation has been suspended.

455. The simulation program is resumed and the graphics suspended. The simulation continues until it reaches a time where the graph is to be refreshed. At this point, the simulation closes the data file, resumes the graphics then suspends itself. The graphics program plots the required graph then reads event flag 71 to ascertain whether or not the simulation is complete. If it is not, the graphics program resumes and the simulation then suspends itself until the next refresh interval occurs. If flag 71 is set, then the graphics simply stops.

A REVIEW OF THE WORK CARRIED OUT BY THE AUTHOR

456. The most notable changes to the structure of the simulation program carried out by the author have been the inclusion of the special purpose utility routines (both the optional and compulsory variety) and the inclusion of the graphics interaction directives. As with any change to the structure of the simulation program, corresponding changes to the program generator must be made. As far as component models are concerned, over sixty have been written, some of which are included in this thesis (Appendix B). The models

included are intended to show new modelling techniques that have been developed during the course of this work, and also to show the use of the utility routines in practical applications.

CHAPTER 5

A PRACTICAL APPLICATION

INTRODUCTION

500. This chapter describes the simulation of a large hydraulic circuit intended to operate four actuators at different constant velocities. The circuit consists of a supply system and four sub-systems. The supply system is common to six operating circuits (i.e. a total of 24 actuators). In order to simplify the simulation, an initial assumption was made. This assumption was that the dynamic response of any one of the four sub-systems had a negligible effect on any of the other sub-systems. In fact, these sub-systems certainly have some effect since the pressure level of the supply is dependent upon how many of the actutors operate at once. However, provided the effect on the supply pressure could be accounted for, then it was considered feasible to simulate the four sub-systems separately. Each of the four simulation programs produced incorporates the supply system together with flow sinks to simulate the remaining three sub-systems. The assumption that the sub-systems did not interact dynamically was subsequently borne out by investigating the dynamic response of the pipes connecting the sub-systems.

501. The supply for the circuit is common to all the sub-systems and is therefore presented separately. In order to gain a complete

picture of a particular sub-system, it is necessary to combine the hydraulic diagram and the computer linking diagram for the supply (figures 5.1 and 5.2) with those for the sub-system in question.

502. The names given to the sub-circuits are derived from the mechanical devices that they operate. They are termed the slide sub-circuit, the cap sub-circuit, the shutter sub-circuit and the catch sub-circuit.

SUPPLY SYSTEM

503. Figure 5.1 shows the supply system. The external hydraulic supply consists of a fixed displacement hydraulic pump, a relief valve and two accumulators which may be supplying utilities other than the six circuits. Each particular operating circuit incorporates an accumulator which affects only the pressure levels in that circuit due to the inclusion of an anti-back flow check valve.

504. This complete supply delivers flow to the slide, cap, shutter and catch sub-circuits. In addition, it supplies other utilities within a particular circuit.

505. Figure 5.2 shows the computer linking diagram corresponding to the supply system hydraulic circuit in figure 5.1. Brief descriptions of the mathematical models are given below:

TK0B01 represents the two port hydraulic tank. It supplies a user-defined pressure to the adjacent models which is

independent of both flowrate and tank volume.

PM0001 represents the prime mover. It supplies a user-defined angular velocity to the pump. The effects of speed droop, maximum torque and motoring are ignored.

PU0001 represents the fixed displacement hydraulic pump. The analysis of the behaviour of the pump is based upon the Wilson model. Pump dynamics are ignored.

PC0101 represents the supply system relief valve. The model supplies flowrate to the adjacent models based upon a user-defined pressure/flow characteristic. The characteristic is assumed linear and leakage, saturation and valve dynamics are ignored.

AC2R models the dynamic and steady state response of a bladder accumulator and pipe combination. AC2R01 represents the two large accumulators of the external hydraulic supply. AC2R02 represents the accumulator serving the particular operating circuit being examined.

DF0A models a flow source with up to six time dependent constant flow stages. In general, this model will be used as a flow sink rather than a flow source. Therefore, the user must define the flow as being negative. DF0A01 represents the other utilities served by the external hydraulic supply, DF0A02 represents the five other operating

circuits and DF0A03 represents other sub-circuits of the same operating circuit. With correct use, these models will allow interaction between different hydraulic circuits to be examined.

OR3Z01 represents the flow limiting orifice.

PC0001 represents the anti-back flow check valve. Mathematically, PC00 is identical to the relief valve model PC01, the only difference being in the advice given concerning the user defined parameters.

PI0501 represents the length of pipe connecting the flow limiting orifice to the anti-back flow check valve. This dynamic model accounts for the effects of air release and cavitation.

506. Links 15 and 16 connect the supply circuit linking diagram to the sub-circuit linking diagram. The slide and cap circuits are both regenerative and as such, require both links 15 and 16. However, the shutter circuit requires only link 15. Therefore, link 16 is not present in the shutter program. The supply to the catch circuit is connected to the pipe which supplies the rod end of the slide actuator. However, the flow to the catch is extremely small and the supply is therefore assumed to remain at a constant level.

SLIDE SUB-CIRCUIT

507. Figure 5.3 shows the slide sub-circuit. The sub-circuit is regenerative on extension of the actuator and employs meter out flow control on both extension and retraction. Situated on the supply port of the directional control valve is a sharp edged orifice, the purpose of which is to create an additional pressure drop whilst the actuator is extending, thus limiting the actuator annulus pressure to an acceptable level. Orifices also exist on the actuator ports, the purpose of these being to dampen the pressure transients in the connecting pipes.

508. The by-pass relief valves connected to the inlet ports of the pressure compensated flow control valves are not included in the simulation programs. The set cracking pressure of these valves is extremely high and they should not pass fluid except in the case of emergency. However, the pressure levels at the flow control valve inlets must be noted following each simulation to ensure that this cracking pressure has not been exceeded.

509. The interlock valve should remain in the position shown throughout the operation of the slide. Therefore, this valve is not included as a model in the simulation programs. However, its effect in terms of pressure loss should be noted and accounted for in an appropriate manner (see para.511).

510. It was required that it be possible to determine the effect of the sharp-edged orifice on the annulus side of the actuator to the

extent of its removal from the circuit. It would be possible to include the orifice and in order to reduce its effect, the user could define an orifice diameter equal to the pipe diameter. Unfortunately, this would produce a system which was, mathematically, extremely stiff and would result in unacceptably long run times. Therefore, two simulation programs were produced: FC1A which does not include the orifice and FC1B which does.

511. Figure 5.4 shows the computer linking diagram corresponding to the hydraulic circuit in figure 5.3. Brief descriptions of the mathematical models are given below:

DC3Z01 represents the slide "open" directional control valve. The model also incorporates the orifice on the supply port of the directional control valve. The dynamic response of the spool is ignored.

DE0101 is a valve controller which defines the fractional displacement of the spool of the directional control valve DC3Z01.

TK0001 represents the return tank.

PI05 models the dynamic response of a frictionless pipe. The model also accounts for the effects of air release and cavitation. PI0502 represents the fluid volume between the directional control valve and the pressure compensated flow control valve. PI0503 represents the pipe line between the

pressure compensated flow control valve and the damping orifice. PI0504 exists only in task FC1B and represents the pipeline between this optional damping orifice and the pressure compensated flow control valve. The pressure loss in these lines should be accounted for by adjusting the effective restriction of the adjacent orifices or valve models.

FC1V models the instantaneous response of a pressure compensated flow control valve. The model includes the effect of limiting the compensating spool travel. FC1V01 represents the retraction flow control valve and FC1V02 represents the extension flow control valve.

OR3Z models a restrictor. OR3Z02 represents the damping orifice situated on the full bore side of the slide actuator. OR3Z03 exists only in task FC1B and represents the optional damping orifice situated on the rod side of the actuator.

PI06 is similar to PI05 the difference being that PI06 accounts for a variable effective pipe volume. In this case, the variation in volume is due to the motion of the actuator rod.

ALAT01 represents the slide actuator. ALAT models the dynamic response of the actuator rod and accounts for stiction, coulomb friction, viscous friction and windage.

The level of these friction effects are user defined and so may be ignored if required. Internal leakage is also accounted for.

CAP SUB-CIRCUIT

512. Figure 5.5 shows the cap circuit. The basic design philosophy is the same as that of the slide circuit. However, an overcentre valve which also operates as a pilot operated reverse free flow check valve and a two stage pressure relief valve is also incorporated in the line connected to the piston side of the actuator. When the cap is to be kept fully retracted, the check valve is piloted open allowing the piston pressure to be maintained at tank pressure. Therefore, since the pressure on the annulus side of the actuator is always of the same order as the supply pressure, the cap will remain retracted even under high external acceleration forces. Furthermore, the differential area of the main stage poppet of the overcentre valve is such that if the actuator piston pressure increases, then the force holding the poppet on its seat also increases. Thus, if the cap is extended and the system is subjected to high acceleration forces, then only the fluid in the line connecting the overcentre valve to the actuator will be compressed. Therefore, any tendancy for the actuator to retract will be eliminated. However, this is only true provided this piston pressure does not exceed the cracking pressure of the second stage of the overcentre valve. The main overcentre pilot line is connected to the directional control valve.

This pilot is connected to supply during retraction of the actuator, thus holding the valve open.

513. As in the slide simulation programs, the by-pass relief valves connected to the inlet ports of the pressure compensated flow control valves are not accounted for. Similarly, the tube sealed interlock valve is not included.

514. Also, as in the case of the slide simulations, two cap simulation programs were produced: FC2A and FC2B (see para.510).

515. Figure 5.6 shows the computer linking diagram corresponding to the hydraulic circuit in figure 5.6. Brief descriptions of the mathematical models are given below:

OR2Z01 represents the pressure reducing orifice. It is intended that the orifice should affect the flow on extension of the cap actuator but not on retraction. Therefore, a reverse free flow check valve was fitted in parallel with the orifice. It was decided to introduce the orifice model upstream of the supply port of the cap directional control valve since this would have a similar overall effect and such a model already existed (OD0Z).

PI05 models the dynamic response of a frictionless pipe. PI0502 represents a non-existent pipe and is required solely for linking purposes. PI0503 represents the fluid volume between the overcentre valve and the pressure compensated

flow control valve. PI0504 represents the pipe line between the pressure compensated flow control valve and the damping orifice. PI0505 exists only in task FC2B and represents the pipe line between the optional damping orifice and the pressure compensated flow control valve. The pressure loss in these lines should be accounted for by adjusting the effective restriction of the adjacent orifices or valve models.

OD0Z01 is an instantaneous combination model which represents the cap directional control valve and the overcentre valve (latch cap open).

DE0101 is a valve controller which defines the fractional displacement of the spool of the directional control valve.

TK0001 represents the return tank.

FC1V models the instantaneous response of a pressure compensated flow control valve. FC1V01 represents the retraction flow control valve and FC1V02 represents the extension flow control valve.

OR3Z models a restrictor. OR3Z02 represents the damping orifice situated on the full bore side of the cap actuator. OR3Z03 exists only in task FC2B and represents the optional damping orifice situated on the rod side of the actuator.

PI06 is similar to PI05 the difference being that PI06

accounts for a variable effective pipe volume. In this case, the variation in volume is due to the motion of the actuator rod.

ALBT01 represents the cap actuator and its associated load. This load consists of the variable effective inertia of the cap and also of the direct force which exists when the actuator is not horizontal.

SHUTTER SUB-CIRCUIT

516. Figure 5.7 shows the shutter circuit. The circuit is not regenerative on extension or retraction. The circuit employs meter out flow control in both the extend and the retract strokes by switching the actuator outlet line to the pressure compensated flow control valve via the directional control valve. The two overcentre valves are similar to the overcentre valve of the cap circuit, the difference being that the reverse free flow checks are not pilot operated. The overcentre valves are held open when the external pilot pressure is greater than some fraction of the main inlet pressure. This has the effect of minimising the possibility of cavitation should the shutter tend to over-run. As in the case of the cap, the shutter is locked open by the overcentre valve connected to the piston side of the actuator. Furthermore, it is locked open by the overcentre valve connected to the annulus side of the actuator. The reason for there being two overcentre valves in the shutter circuit whilst in the cap circuit there is only one is that

in the case of the cap, the line connectd to the rod end of the actuator is always connected to supply. However, in the case of the shutter, following retraction the line connected to the rod end of the actuator is exhausted to tank. Therefore, in order to ensure that the piston remains on its end stop when retracted, pressure must be trapped in the rod end of the actuator, a requirement achieved by the inclusion of the second overcentre valve.

517. As in the slide simulation programs, the by-pass relief valve connected to the inlet port of the pressure compensated flow control valve is not accounted for. Similarly, the tube sealed interlock valve is not included.

518. Figure 5.8 shows the computer linking diagram corresponding to the hydraulic circuit in figure 5.7. Brief descriptions of the mathematical models are given below:

PI2Z01 is a dynamic model of a pipe. This model has been included since, in the case of the shutter simulation, there are no models which can account for pressure loss between the accumulator (AC2R02) and the shutter directional control valve (OD1Z01). Great care had to be exercised to ensure that the friction loss along the pipe is significant since if a user defines parameters which result in an extremely low pressure loss, then he runs the risk of creating a simulation which is mathematically stiff and may therefore take an unusually long time to complete.

OD1Z01 is an instantaneous combination model which represents the shutter directional control valve and the two overcentre valves (latch shutter open, latch shutter shut).

DE0101 is a valve controller which defines the fractional displacement of the spool of the directional control valve.

PI06 is similar to PI05 the difference being that PI06 accounts for a variable effective pipe volume. In this case, the variation in volume is due to the motion of the actuator rod.

AL3Z01 represents the shutter actuator and its associated load. This load consists of the variable effective inertia of the shutter and also of the direct force which exists on the shutter.

PI05 models the dynamic response of a frictionless pipe. PI0502 represents the fluid volume between the directional control valve and the pressure compensated flow control valve.

FC1V01 models the instantaneous response of the pressure compensated flow control valve used for both the extend and the retract strokes.

TK0001 represents the return tank.

CATCH SUB-CIRCUIT

519. Figure 5.9 shows the catch circuit. Due to the low flowrates and extension/retraction times of the catch actuator, the supply pressure can be assumed constant. Therefore, the supply system is not modelled.

520. Neither the shutter open interlock valve nor the shuttle valve are required in the simulation. However they do restrict the flow and their effect must be accounted for in an adjacent model.

521. Figure 5.10 shows the computer linking diagram corresponding to the hydraulic circuit in figure 5.9. Brief descriptions of the mathematical models are given below:

DE00 models a pressure source, the level of which may vary with respect to time as defined by the user. DE0001 represents the pressure source connected to the catch circuit via the slide, cap and shutter open interlock valves. DE0002 represents the pressure source connected to the circuit via the slide annulus pipe line.

PI2Z is a dynamic model of a pipe. The model accounts for pressure loss along the pipe due to friction. PI2Z01 represents the pipe line connected to the circuit via the interlock valves. Pressure loss across these valves should be accounted for by increasing the effective length of the pipe represented by PI2Z01. PI2Z02 represents the pipe line

connecting the circuit to the slide valve annulus pipe line. This pipe model should also account for the pressure loss due to the reverse flow through the pressure compensated flow control valve (figure 5.3). Great care should be exercised to ensure that the friction loss along the pipe is significant. If a user defines parameters which result in an extremely low pressure loss, then he runs the risk of creating a simulation which is mathematically stiff and may therefore take an unusually long time to complete.

OR3Z01 represents the catch lower flow controlling orifice.

PC0001 represents the reverse free flow check valve connected in parallel with the flow controlling orifice. The model supplies flowrate to the adjacent models based upon a user-defined pressure/flow characteristic. The characteristic is assumed linear and leakage, saturation and valve dynamics are ignored.

PI05 models the dynamic response of a frictionless pipe. PI0501 represents the fluid volume between the check valve, the flow limiting orifice and the catch actuator.

AL1V01 represents the catch actuator and its associated load. This load consists of the variable effective inertia of the gears, the top stop and the rear catch and also of the direct force which exists due to the weight of the top stop and rear catch.

MODEL VALIDATION AND VERIFICATION

522. It is important to recognise that model verification and model validation are distinct processes. To verify a model, one must ensure that the equations developed to represent a component are correctly coded. When validating a model, one is in fact validating the equations that have been derived. This validation process may take a variety of forms depending upon the model(s) being validated.

523. All models developed are verified in isolation. Some models are also validated either in isolation or as a combination as required. The verification process consists of linking the model under test to a series of algebraic test models. These test models take the form of sources of pressure, flow, torque etc. With the aid of the test models, the model under test may, for example, be subjected to step changes in torque, or to a differential pressure which sweeps across a complete operating characteristic. Diagnostic write statements are occasionally included in a model to aid verification.

524. Provided a model has been adequately verified, it is not always necessary to carry out a lengthy validation process. For example, a model of a relief valve which consists of pressure/flow characteristics relies purely upon the information obtained from a manufacturer's catalogue. It can be argued that a model is valid in certain conditions and not in others, thus making its validity dependent, amongst other things, upon user defined parameters. Therefore, it is necessary to clearly state the assumptions upon

which a model is based and place the onus upon the user to ensure that these operational limits are not violated. However, a certain amount of validation is carried out, primarily on models where the equations being used have lengthy derivations, or on combinations of models which initially produce dubious results. For example, the shutter model (AL3Z) includes a great deal of trigonometric formulae which tend to attract mistakes. Therefore, a graphical validation of the effective inertia of the shutter and the resultant force on the actuator rod was carried out (see Appendix B.100). This particular validation method also verified the coding. A further example of validation is given below in the form of a test rig to validate the slide valve simulation.

525. A small amount of test work was carried out in order to validate the slide circuit with particular emphasis being given to the retraction stroke. The reason for this is that the retract stroke displays somewhat oscillatory behaviour. The parameters used in the simulation program were adjusted in order that they should match the parameters of the test rig. Figure 5.11 shows the test circuit and figure 5.12 the test results. Comparing these test results with the simulation results to be decsribed in the next section, good aggreement is found.

THE RESULTS

526. Figures 5.13 to 5.22 show selected results from a series of simulation runs of the circuits described. Not all results are shown

since just one run of all four simulations would produce approximately 180 graphs. Many of these are, of course, of little interest and with experience, a user can select and view the relevant results quickly and easily.

Results of the Slide simulations

527. Figure 5.13 shows a selection of responses of the slide simulation. This was considered as a standard run since the data defined represented the proposed normal running of the circuit. Figure 5.13(a) shows the variation of the displacement of the actuator with respect to time. The time taken for the actuator to reach maximum stroke is slightly less than 0.75 seconds. The actuator is held open for only a fraction of a second before retraction. This time interval is kept short due to the fact that the simulation often runs at the same speed or even slightly slower when actuators are at their end stops. The time interval between extension and retraction is, of course, large enough to ensure any system transients have died away. Figure 5.13(b) shows the variation of actuator velocity with respect to time.

528. Figure 5.14(c) shows the variation of the actuator piston pressure. These systems incorporate tanks which are pressurised at 1 bar (gauge). Therefore, when the actuator is fully retracted, the pressure in the piston end is that of the tank (or the pressure in the return line should other sub-circuits be active). When the directional control valve is actuated, the pressure in the actuator

rises until the stiction is overcome and the mass begins to move. Once the actuator piston is moving, the friction reduces to the normal level of coulomb friction (plus viscous friction if applicable) causing the pressure response to overshoot slightly. The pressure quickly reaches a stable value though it does slightly decrease during the extension of the actuator. This is due to the fact that, even though the system is regenerative, the flow required from the supply system is large enough to cause the supply pressure to reduce. After approximately 0.75 seconds, the actuator reaches its end stop and the pressure rises to that of the supply system (again assuming that no other systems are active). The pressure now rises slightly as the accumulators recharge. After a short length of time, the directional control valve is returned to its original position and the pressure reduces. Again, the response overshoots due partly to the effects of stiction and inertia as mentioned above, and partly due to the response of the flow control valve, the compensating spool having to move from an inactive state to a controlling state.

529. This is a region where a design engineer should pay particular attention. A requirement of the system designers was that the system should be reasonably quiet. The oscillation in the pressure level would almost certainly cause some vibrational noise and there is also the remote possibility of cavitation occurring. A system designer should recognise these facts and be aware of the parameters which affect the response. The simulation should be rerun, varying

relevant parameters, in order to investigate the possible adverse consequences. It should be remembered that these parameters should be varied beyond what may be considered as reasonable for the particular components in order to account for the effect of modelling inadequacies.

530. Figure 5.13(d) shows the variation of flowrate into and out of the piston side of the actuator. It should be noted that a sign convention is applied to flowrate throughout the package. A flow is positive if it flows into a pipe such as PI05. The variation of flowrate is of course similar to that of actuator velocity, the only difference being due to acuator leakage and the sign convention mentioned above.

531. Figure 5.13(e) shows the variation of pressure in the annulus side of the actuator. The only interesting feature of this response is the pressure intensification during extension of the actuator causing a pressure level of almost 300 bar. Again, the levels slowly increase and decrease due to the recharging and discharging of the supply system accumulators. The supply system pressure is shown in figure 5.13(f).

532. Figure 5.14 shows the effect of changing certain parameters from the standard set defined above.

533. Figure 5.14(a) shows the variation of actuator piston pressure for a circuit which includes the damping orifices on the actuator ports. As can be seen, the oscillatory response has been reduced

without adversely affecting the 'steady state' pressure levels.

534. Figures 5.14(b) to (e) show the effect of starting the simulation with a substantially lower system pressure. This is possible following a series of operations of the four circuits. In this case, the pressure in the actuator piston rises even during extension due to the rapid recharging of the accumulators. The pressure compensated flow control valves still operate normally, thus ensuring that the extension and retraction times of the actuator remain around 0.75 seconds. The only possible problem is in the level of piston pressure following commencement of actuator retraction. The pressure now reduces to approximately 10 bar warning of possible problems.

535. Figure 5.14(c) shows the effect of a low initial system pressure on the circuit including the damping orifices. It can be seen that the oscillatory response is reduced still further.

536. These simulation runs have shown three important points. Firstly, the system behaves acceptably under normal conditions. Secondly, an initial pressure level of half the normal value does not have an adverse effect on the extension and actuation times of the actuator. Finally, a possible problem exists in the slightly oscillatory response of the actuator piston pressure warrenting further investigation.

Results of the Cap simulations

537. Figures 5.15 and 5.16 show a selection of responses of the cap simulation. Figure 5.15 shows the standard run. The design and operation of this circuit is similar to the Slide circuit and, as such, little detailed explanation is required. However, there are two important points to note. Firstly, due to the much lower velocity of the actuator piston on retraction, the worrying pressure oscillation present in the slide circuit results is not of an important magnitude in this simulation. Secondly, due to the design of the cap actuation mechanism, the effective mass of the cap is extremely high (approx. 1750 kg, figure 5.15(c)).

538. Figure 5.16 shows the results of two further simulation runs. Figures 5.16(a) and (b) show the actuator piston and annulus pressure (respectively) due to a low initial system pressure. The remaining two graphs also show these two pressures but due not only to low initial pressure, but also to an operational temperature of 0 C. As can be seen, the important features of actuator extension and retraction times are not affected.

Results of the Shutter simulations

539. Figures 5.17 to 5.20 show a selection of responses of the shutter simulation. In this case, there is no standard run since the load on the shutter is not known sufficiently accurately. Therefore, the purpose of this simulation is to show the limits of operation.

The four figures show the change in responses due to increasing shutter loads.

540. Figure 5.17 shows the system response due to a shutter load starting at 10kN when the shutter is closed (actuator retracted) and reducing in a linear fashion with respect to shutter angle, becoming zero when the shutter is horizontal. Figure 5.17(a) shows the variation of actuator displacement. The extension and retraction times are significantly longer than those of the previous simulations (approx 5.5 and 8 seconds respectively).

541. Figure 5.17(b) shows the variation of force on the actuator rod due to the external shutter force with respect to actuator displacement. It can be seen that at approximately half stroke, the load of 5kN is intensified to approximately 30kN at the actuator rod. Similarly, figure 5.17(c) shows the variation of shutter inertia reflected to the actuator as an effective mass.

542. Figure 5.17(d) shows the variation of actuator piston pressure with respect to time. In this case, the effect of the external load can be seen on the pressure response. On extension, the pressure is that of the supply system, apart from a small pressure loss in the supply lines. However, on retraction, the pressure level reduces to compensate for the external load, becoming a minimum at approximately half the actuator stroke. It should be noted that this pressure is approximately the inlet pressure of the pressure compensated flow control valve. Therefore, the compensating spool of the flow control

valve is moving in order to attempt to maintain a constant pressure drop across its preset metering orifice.

543. The pressure transient present in both previous simulations exists also in the shutter simulation. In this case, due to the much lower velocities, the amplitude of the transient is smaller than in the slide or cap simulations. However, it does seem that the oscillation is less well damped. It was, in fact, this response which caused the dynamic pressure compensated flow control valve model (FC8Z) to be written. This simulation was carried out with an instantaneous version of the model (FC3Z) which implicitly assumes that the response of the compensating spool was instantaneous. By replacing FC3Z with FC8Z, this oscillation could no longer be detected on the simulation responses. This example emphasises the importance of the correct selection of models by the user, an ability relying upon hydraulics experience rather than computational ability.

544. Figure 5.17(e) shows the variation of the actuator annulus pressure. Again, the effect of the external load can be seen, this time as a pressure intensification during actuator extension. The pressure level reaches a maximum of approximately 335 bar and it should be noted that the pressure/flow characteristic of the pressure compensated flow control valve begins to droop significantly at around 350 bar.

545. Figure 5.18 shows the effect of increasing the external load on the closed shutter to 35 kN. The remainder of the load definition

remained the same. Qualitatively, the reponses are similar to those in the previous run described above. However, an important difference exists. Firstly, due to the intensification of the actuator annulus pressure causing pressure levels of around 460 bar, the pressure compensated flow control valve no longer controls at the set value and the actuator takes longer to extend. This can be seen in figure 5.18(e) which shows the flow through the valve (the set level being 0.25 l/s).

546. Figure 5.19 shows the effect of increasing the shutter load still further. In this case, the load on the closed shutter is defined as 40kN. Figure 5.19(a) shows a significant problem in the motion of the actuator. On retraction, the actuator slows for a time before returning to its original velocity. Figure 5.19(c) shows the cause of this problem. The differential pressure across the pressure compensated flow reaches an extremely low level and the valve no longer accuractely controls. The differential pressure is, in fact, lower than that required across the preset metering orifice of the valve. The valve has become a simple restriction.

547. Figure 5.20 shows the logical conclusion of this examination. The load on the closed shutter is increased to over 50kN. In this case, the extension time of the actuator is affected, but on retraction, the supply pressure is insufficient to close the shutter. This effect could obviously be obtained with a lower system pressure and a lower load.

548. This simple series of simulation runs shows the important facility of being able to examine the effects of various load conditions. This is extremely important since it is the often the load which is most difficult to define to any degree of confidence.

Results of the Catch simulations

549. Figures 5.21 and 5.22 show a selection of responses of the catch simulation. The extension of the catch actuator is extremely fast since there is no pressure compensated flow control valve present. In fact, there is little restriction of any kind. However, on retraction, the check valve closes and the flow is metered out of the actuator. Figure 5.21(a) shows the variation of actuator displacement and 5.22(b) the variation of the velocity of the actuator rod. As can be seen, the response times are approximately 0.05 seconds on extension and 0.2 seconds on retraction.

550. Figure 5.22 shows the effect of operating the circuit with a reduced system pressure and without recharging the accumulators of the supply system. The important point to note is the now much slower response of the actuator, requiring 0.2 seconds to extend and 0.35 seconds to retract.

Conclusion of the results

551. With these results, it was the author's intention to give the reader an idea of what is possible. It was not intended that they

should show the complete design process from initial concept to the tuning of components with the aid of the simulations, though the feasibilty of this process should be clear.

552. The results show how a user can investigate extremes of expected operating conditions (the effects of initial supply pressure and low temperatures in the slide, cap and catch simulations), or search for failure conditions (the external load on the shutter). In some cases, it was the dynamic response which of most interest (the transient response of the actuators when commencing retracting and the response of the pressure compensated flow control valve); in others, the steady state operation was important (the shutter simulation). In all cases, whatever system could be imagined by the user, can be simulated.

CHAPTER 6

MANAGERIAL ASPECTS

INTERNAL MANAGEMENT

600. The method of management of HASP that has been adopted, though a somewhat vague subject, is of general interest and may be found to be applicable to many other CAD packages. The particular topics of interest are documentation, categorisation and standardisation.

Documentation

601. The production of detailed documentation is a generally accepted necessity. However, it is still considered a chore and the lesson is always learnt through experience (usually through bad experience). There are two types of HASP documents produced. The first is the manuals which describe the overall structure of HASP. The second type of document is the component model report. This consists of a detailed report of a component model and should contain an introduction, basic modelling assumptions, information on linking, the parameters to be defined by the user and finally a complete explanation of the equations which have been used to represent the behaviour of the component. These reports should contain sufficient information for any other modeller to enhance or debug the model. Due to the number of equations in these reports, it is impractical to employ a computer word processor to produce them. The onus is upon individual modellers to produce such a report and this documentation

must be screened by all users before the corresponding model is accepted into the component model library.

Categorisation

602. All new and updated software produced for HASP is categorised. In general, there are three categories. The lowest category contains experimental software. The second category contains software which has been employed by users other than the developer and to which some degree of confidence may be attached. The highest category contains software which has been used a great deal by a number of users and which seems to have no outstanding problems. Software from one category may be promoted to a higher category following a meeting of HASP users. Loss of integrity of the program generator is obviously very serious. Therefore, great care is exercised in replacing existing program generators by experimental generators and the retiring standard version is always archived. Though this method of categorisation applies to all HASP software, component models are the most usual to be dealt with.

Standardisation

603. Once a large program such as the generator or the graphics programs have been produced, then the general structure is set. However, component models are produced by many modellers and provided the link between these subroutines and the main simulation routines is in order, modellers are able to produce coding of any form. Unrestricted production of model coding would have two adverse consequences. Firstly, it would mean that any person attempting to

debug or enhance a model of which he is not the author would be faced with unfamiliar coding making the model extremely difficult to amend. Secondly, there would be few guidelines for new modellers. It is obviously important to attempt to pass on any good or bad experience gained through modelling. Therefore, standard methods of mathematical modelling and coding have been developed [18]. In fact, the methods have become so refined, that a component model generator now seems feasible (see para.711).

DIRECTORY STRUCTURE

604. An important but often neglected aspect of the installation of a CAD package is the location of the files and who should have access to them. Ultimately, this is obviously the decision of the system or group manager. However, certain elements of the package are tailored for the computer system employed at the University. An explanation of the directory structure adopted by the author will aid the adaptation of the package to other operating environments in the future.

605. Users of the VMS operating system are given the option of creating sub-directories. The use of sub-directories basically arranges the contents of the user directory into segments. Therefore, certain types of files may be grouped for easier access. An example of a user directory might be [HULL]. An example of a sub-directory associated with this user directory might be [HULL.PHD] (in fact, the sub-directory where the manuscript of this thesis was

prepared).

606. HASP is contained in several user directories, each sectioned into several sub-directories. The program generator, the command line interpreter and the graphics programs should not require a great deal of access and furthermore, access other than that required for execution should not be granted to general users. These programs together with their source reside in the directory [PG]. Figure 6.1 is a schematic of the sub-directories associated with [PG].

607. The sub-directory [PG.A] contains the source of the standard program generator PGA. It also contains the task image of this program generator and the system files required to produce it. The users known as group users and world users are only granted "execute" access.

608. The sub-directory [PG.Z] contains experimental versions of the program generator created by the author. Any new program generator resides in this directory until it is decided that it may be promoted to PGA. The file protection is as for [PG.A].

609. The sub-directory [PG.ARK] contains old versions of the standard program generator which have been superseded. Again, the file protection is as for [PG.A].

610. The sub-directory [PG.GRAF] contains the task images of the four graphics programs. It also contains the source of the main segments and certain controlling subroutines of the graphics programs

and the system files required to produce them. There is also an associated sub-directory (a level 2 sub-directory) called [PG.GRAF.P10] which contains the Tektronix Plot-10 graphics source and the corresponding object module library. These files have the same protection as those mentioned above.

611. Finally, the sub-directory [PG.CLI] contains the HASP command line interpreter. This consists of several files containing command procedures. As such, the file protection must allow read access to all users.

612. The second directory used is called [HASP]. Primarily, this directory contains the component models. Therefore, unlike [PG], the file protection associated with [HASP] is greatly relaxed. A schematic of its sub-directories is shown in figure 6.2.

613. By far the largest and also the most important sub-directory of [HASP] is [HASP.COMPON]. This sub-directory contains both the source and the object code of all the component models. It may be considered as the library of components though the obect code exists as separate files rather than being accumulated in an object module library. This sub-directory also contains the component model attributes file COMPON.DAT. Modellers must have free access to [HASP.COMPON] in order to add new models to the system and also to obtain copies of existing models upon which new models may be based.

614. The sub-directory [HASP.COMPONARK] is, as the name suggests, an archive for obsolete or superseded versions of a component model.

Its file protection is identical to that for [HASP.COMPON].

615. The sub-directory [HASP.MANUALS] contains the user manuals that have been produced to provide sufficient documentation for all levels of user. The manuals have been created using the word processor known as Digital Standard Runoff (DSR). In a working environment, all users should have online access to these manuals. Therefore, the file protection for this sub-directory must allow "read" access to all HASP users.

616. It is also useful to look at a typical user directory belonging to a HASP user as this details certain interesting features of the program generator. Figure 6.3 is a schematic of the directory structure of a typical HASP user.

617. The most important group of sub-directories are [HULL.HASP.SIMUL] and [HULL.HASP.TEST]. The former is the sub-directory where all HASP simulations are carried out. The directory contains all the data files describing the circuit layout, the component parameters and the simulation results. The directory may also contain the source and object code of experimental component models. When a component model is being developed and undergoing trials, it would be extremely tiresome to continually copy files to and from the standard model library [HASP.COMPON]. Therefore, a facility exists for the user to specify that a model is experimental and will be found in the user (default) directory rather than in the standard library. This is achieved by placing an asterisk (*) after

the component name when defining the circuit data for the program generator (para.304). The generator segment PGIN then sets flags to indicate which components are experimental. The segment PGSEL reads these flags and duly acts upon them when writing the options file CAD.OPT.

618. The sub-directory [HULL.HASP.TEST] was used to develop experimental versions of the program generator and to investigate structural changes to HASP without endangering the integrity of the standard version of HASP.

619. The sub-directory [HULL.PAPERS] contains reports written using the word processor DSR, the sub-directory [HULL.PRIVATE] contains source, object code and other files not associated with HASP, and [HULL.SETUP] contains programs which setup terminal characteristics and also contains the log-in command procedure. The log-in command procedure is a useful file which sets characteristics when the user begins a session. It also sets up a large number of command abbreviations. For example, a useful abbreviation might be to set WORK to represent the command SET DEFAULT [HULL.WORK.SIMUL].

620. The file protection of the various sub-directories should be left largely to the user. However, it should be remembered to always allow the system access to all files. The same is obviously true with respect to the owner's access. Furthermore, it is important to ensure that the default protection set to files edited within the same group is sufficiently relaxed. It is likely that the user would

edit the component attributes data file [HASP.COMPON]COMPON.DAT from his own directory, thus forcing his own file protection on the new version (since he is in fact the owner of the new version).

METHODS OF USE

621. An important consideration in the development of a CAD system is the method of implementation that is likely to be adopted. It is necessary to determine the type of personnel who are to use the system. If possible, it is desirable to produce a system which may be applied by different types of users, albeit to different levels of problems. It is worth considering the types of users likely to find HASP useful.

622. Firstly, there is the experienced user. He will have an understanding of the simulation program at the source level and will be able to amend existing models or create new models manually as required. The package probably caters for this type of user better than any other. The user can assemble simulation programs in a matter of minutes and if necessary, produce new component models in a matter of hours provided he follows the standard methods of modelling.

623. Secondly, there is the so-called "computer-naive" user. He cannot be expected to have any understanding of high level languages. In practice, he may be a hydraulics design engineer involved in initial design or in system trouble-shooting. The package has been

designed with this type of user in mind. However, at present he must suffer the restriction of only being able to employ component models which already exist in the component model library. The development of a model generator would greatly alleviate this problem (see para.711).

624. Finally, there is the person who will require the simulation work to be carried out by others. He would not be directly involved with the package. The person carrying out the simulation work need not have an understanding of basic hydraulic principles as he would be directed and closely supervised by an experienced hydraulics engineer. This is a perfectly feasible method of use. However, the purpose of the package is to entice engineers to make use of the computer and to break away from the tradition where the engineer defines a problem and a programmer sets out to solve it. As such, this type of user has not shaped the development of the package in any way.

625. Furthermore, it is important that in all the categories outlined above, the user must thoroughly understand the engineering problem. In the development of HASP, the goal has always been to produce a package which allows the user to define problems in a very simple manner. However, it is possible to envisage this ease of use becoming counterproductive. The author has experience of users of the system who had badly thought out their problems; ones which create simulations of hydraulic circuits which are at best inapplicable and at worst unrealistic. The manner in which this

problem can be overcome is not obvious since taking away the chore of computing necessarily allows the user to create a multitude of possibly useless simulations. However, the problem has been overcome to some degree by the inclusion of the warning routine MESAGE (see para.434 to 436) which causes the user to think twice about data of "unusual" proportions. Furthermore, it should be stressed that the user should take as much care in preparation of the data and analysis of the results as would be taken if equivalent calculations had been carried out by hand.

INSTALLATION OF THE SYSTEM

626. An important question which should never be overlooked when producing a CAD is system is 'what problems are there in transporting the package onto other computers?'. This introduces several aspects; computational portability, patents and copyright and legal liability.

Portability

627. The problem of portability, i.e. system dependence, is one of a very mechanical nature. Firstly, it must be accepted that at best, HASP will only work on a machine which has a Fortran compiler. When developing HASP, it had to be decided which level of Fortran should be used. The package was originally developed in Fortran IV (ANSI-X3.9 1966) with few deviations. In the main, the rich library of functions available under DEC Fortran were ignored. However, when one considers the compilers available under other operating systems [19], then some deviation is justified. An example of such a

deviation is the use of quotes to denote literal strings in FORMAT and OPEN statements rather than the near obsolete Hollerith character. A more recent decision has been made to adopt Fortran-77 (ANSI-X3.9 1978) as the standard language and to make use of its facilities such as the block IF and ELSE statements. This language is no more permissive than Fortran IV but does allow the programming to become more structured and also maintains a high degree of portability.

628. A further aspect when considering portability is the accessing of internal files (i.e. accessing files directly from a program). Internal files are used by both the program generator (system data file, component attributes file and the simulation program source) and the simulation program (parametric data and results data). Both sequential and direct access files are used, their structure being defined in the OPEN statements. The keywords used in the OPEN statements all conform to the Fortran-77 standard and should therefore be compatible with any Fortran-77 compiler. However, the file names defined by the FILE keyword often contain directory names and will therefore not be common to all operating systems.

629. A number of files associated with HASP are not (and cannot) be standard. All command procedures such as the command line interpreter can only be used on machines that support the Digital Command Language (DCL) (see para.204 and Appendix A). However, the basic structure of the procedures may be applied to other operating systems. The other important file which is system dependent is the

component selector file produced by the program generator (see paras.315 to 342). Again, the structure of the routines designed to write the selector file (PGODL or PGSEL) may be adopted for use with other systems.

630. If the package were to be transferred to a compatible user system where there would be no further modelling carried out, then the library version of HASP outlined in para.343 would be of use. The package would then merely consist of five permanent command procedures (HASP-CI), seven task images (the program generator and six graphics programs), one data file (the component attributes file) and one library (the component object module library); a total of thirteen files occupying approximately 0.8 mega-bytes of disc space (with a library of one hundred component models and the files compiled and linked under VMS). All other files are transient and are automatically kept in order by the interpreter.

Patents and copyright

631. The legal problems associated with computer software have been investigated by several authors [20,21] and only a brief summary is required here. As far as ownership is concerned, the two main paths open to developers of goods is to attempt to patent them or to accept their copyright. If they are patented, then in return for their disclosure, the supplier is given a monopoly in their use for twenty years [22 para.301]. However, the United Kingdom Patents Act of 1977 specifically states that computer software (together with mathematical methods and scientific theory) cannot be

patented [22 para.566]. The possibilty of patenting, however, still exists if the software is associated with a piece of hardware in some way. For example, a read-only micro-electronic chip containing a program (often termed firmware) can be patented. The possibility of producing chips containing HASP is obviously remote and therefore the question of patenting the package is inapplicable.

632. A more effective method of asserting ownership of computer software is through copyright laws. Copyrights do not provide a monopoly but they do protect the form in which an algorithm is coded. Copyright is not concerned with originality of ideas but with expression of thought [23 para.831]. The United Kingdom Copyright Act of 1956 states that all literary work is automatically copyrighted from the time the work was produced even if it remains unpublished [23 para.834]. For the purposes of this Act, computer programs are considered literary works [23 para.835]. Further requirements of the Act are that the work should be original, should be documented and should be written by a "qualified person" [23 paras.830-831]. Therefore, HASP is automatically copyrighted under this Act.

Liability

633. The problem of liability is complex in comparison to those of patenting and copyright and any conclusions that may be drawn from the statutes are far more vague. The Acts which normally affect the legal rights of the two parties are concerned with contract law [23] and misrepresentation [24]. Liability is a problem which is of great

importance to software developers since their product may be used in circumstances where undetected errors or unintentional misrepresentation by the developers may cause substantial loss to the user.

634. It is useful to examine the two methods in which HASP may be used which are most affected by legal liability. Firstly, if the package were to be used in industry and the subject of cost was important, then as with the majority of software packages, it should be leased rather than sold. This allows the developers to retain a proprietary interest in the package since only object code need be supplied and no information concerning the design philosophy of the software need be revealed [20]. Secondly, it is possible to supply only a set of programs which simulate hydraulic circuits specified explicitly by a customer. In this case, the customer would probably attempt to secure a fixed price contract since the estimation of the time and cost required to develop software is an extremely difficult task.

635. In both these cases, it is possible to provide a contract with a general exemption clause which limits the liability of the developers to a sum such as the total cost of the contract. There can be no exemption from liability due to a breach of contract, but software developers can do their best to exempt themselves from liability due to accidental misrepresentation and undetected errors. However, exclusion clauses are now governed by the United Kingdom Unfair Contract Terms Act of 1977 [25 para.366]. This states that

contracts may include exclusion clauses but only if they satisfy certain tests of "reasonableness". For example, it is reasonable to write an exclusion clause if the software is known to be of an experimental nature [20]. However, the most secure method of writing an exclusion clause is to involve an arbitrator since the Act states that a person with relevant experience is better able to decide what is reasonable than the Court [25 para.366A.2].

CHAPTER 7

DISCUSSION

AN OVERVIEW

700. The objective of this work was to take an existing CAD package and to examine and correct its downfalls. The package had to be looked at with the eye of a user, not a simple task when the complete structure of the package and its subsequent use has become second nature. Making the package usable necessarily entails correcting anything that might be construed as being detrimental to the smooth running of the package and also the development of new features to achieve the aim.

701. To quantify the work carried out by the author is not simple. It is insufficient to examine the state of the package as others have contributed to its advancement. It would be possible to merely list the work carried out but that would be to ignore the overall effect of this work. The magnitude of a particular task does not reflect its impact on the basic operation of the package. In this discussion, an attempt has been made to examine the effect of each of the advancements in terms of the overall philosophy of HASP, be it on the small scale of introducing a new modelling technique or on the somewhat larger scale of producing a new user interface. It must be accepted that the fact that some of the work has been involved with the further development or rewriting of existing software should not

detract from the originality of achieving a similar effect in a new manner. The remainder of this section discusses the work in the order it has been presented in the thesis.

The command interpreter

702. Having attained the original requirement of not requiring programming experience in order to use HASP, the user was still left to understand the complete structure of the package together with the an array of system commands. The command interpreter has made the package accessible by users with no experience of computer systems. With its use, the engineer need not learn any system commands other than those concerned with logging in and logging out.

The program generator

703. The program generator is the most important feature of the package and has required a great deal of attention. It is unique in the field of hydraulic simulation. Other simulation languages employ coding generators but these work from a description of the mathematical model in the syntax of that language. The HASP program generator produces coding which controls existing mathematical models.

704. During development of the package, changes to the basic structure of the simulation program usually necessitates larger changes in its creator, the program generator. In addition, the structure of the program generator itself has been substantially altered in order to correct, extend or make the generator more

efficient. Four versions of the program generator have been produced by the author: Firstly, the standard version of the generator which operates under RSX; secondly, the version of the standard generator which is compatible with an object module library containing all the component models (para.343); thirdly, the version of the standard generator which produces a simulation program allowing graphics interaction (paras.446 to 455); and finally, the program generator which operates under VMS.

The object module library

705. Transferring the package from one installation to another can be a cumbersome and time-consuming process. This task is made worse by the fact that there are so many files. This problem can be alleviated to a great extent by employing an object module library to contain the object code of the complete set of component model subroutines and also the routines which constitute the integrator. Using a library reduces the number of files necessary to produce a running version of HASP from several hundred to approximately ten.

The attributes file editor

706. The component attributes data file COMPON.DAT is an important file. It is essential that its integrity is not violated if simulation programs are to be successfully generated. However, during the devlopment of component models, this file is extremely susceptible to errors in the form of erroneous components descriptions. These errors can be difficult to detect. The COMPON.DAT editor alleviates this problem by providing an interface

between the user and the data file.

New modelling techniques

707. Many models have been developed, a large number of which are mundane and show no new techniques. However, several of the models do bring forward new ideas in terms of HASP modelling. The most important of these are the use of the standard utilities in models such as AL3Z and PC1Z, the instantaneous characteristic model such as FC8Z, the use of iterative techniques as employed by PU0Z and LL1Z and the modelling of complex mechanisms as in AL2Z and AL3Z.

The standard utilities

708. The standard utilities provide a set of useful routines which may be used in any number of new component models. The work required to incorporate these tools into HASP included amendments to the program generator.

Graphics/simulation interaction

709. The online graphics facility now available is an important feature of HASP. It employs an interrupt process which allow the user to interrogate the simulation whilst the calculation is in progress. This has proved a valuable asset in the day-to-day use of HASP.

The practical application

710. The practical application described in Chapter 6 shows the package being used for a complex mechanical handling circuit. The example shows the manner in which the circuit may be segregated and

the assumptions which may be made when a circuit is constructed. It shows how the onus of making assumptions rests with the user as much as with the modeller. The modeller provides a set of component models, often several versions of the same component, each having different assumptions. However, the user is left to connect these components as he will. Therefore, the complete set of assumptions for a simulation is dictated by the user.

THE FUTURE

711. In terms of usability, the package would benefit from several further advancements. Firstly, the development of a model generator would alleviate the problems involved with users having to write their own models should they not exist in the standard library. The user would still be expected to define the fundamental equations of the model but would not be burdened with such tasks as editing, calculating smoothing polynomials or implementation of region indicator logic. Secondly, the manual method of producing linking diagrams could be superseded by an automated method. This would allow the user to produce diagrams on the computer terminal and to have this diagram automatically compiled into the data required by the program generator. Thirdly, it would be desirable to produce a large data base which contained dimensional and other parametric data for a large number of commonly used components. The data file would be produced by the component parameter definition routines which would insert all the user defined data together with the

manufacturer's name and component code number. Any user could then retrieve the data by simply stating the particular make and number of the the required component.

712. Feasibility studies have been carried out on the model generator and the graphical input program and are described in Appendix E. Sufficient work has been carried out to indicate the algorithms required and a possible structure for the software.

CHAPTER 8

CONCLUSION

800. The simulation package described provides a valuable tool for hydraulics engineers, including those with no previous experience of computing.

801. Firstly, the package goes further than simulation languages in that the user need not learn a vocabulary of mathematical mnemonics. This is achieved by the development of a program generator and the provision of a comprehensive library of component models. The author has structured a major portion of the program generator, developed the simulation program control segments and produced new modelling techniques which may be adopted by both users of HASP and others in the future.

802. Secondly, a package has been built around the coding generator and the simulation program. This package consists of all the necessary graphics programs and also an automatic editor which allows advanced users to carry out modelling more efficiently by reducing the possibility of logical errors. Further work on the graphics programs has given the user the important capability of interrupting simulations and examining results. The editor and the simulation interrupt logic were produced solely by the author.

803. Finally, a command interpreter was produced which is able to carry out a complete range of system operations following a simple

high level command issued by the user. This interpreter is a shell around the whole package and means that the user need never explicitly issue a command to the computer operating system. The interpreter allows efficient use of the rich capabilities of HASP by reducing the possibility of user-errors. An equally important facet of the interpreter is that it creates a facade which is simple to understand, practicable and aesthetically pleasing.

804. Gordon and Riesenfeld [7] have succinctly described the aim of all interactive design systems: 'In order to be successful, a system must have appeal to the designer - it must be simple, intuitive and easy to use. Ideally, an interactive design system makes no demands on the user other than those to which he has been formerly accustomed through ... design experience.' Though obvious, this aim is rarely seen so clearly.

805. Through the current work, a system has been developed which can be considered as the only package in the field of hydraulics engineering to approach satisfying this criterion.

ACKNOWLEDGEMENTS

The author would like to thank his supervisor D.E. Bowns and also Steve Tomlinson for the many useful discussions. Also, thanks go to Jennifer Griffin for her help in preparing the manuscript. The financial support given by the Science and Engineering Research Council, the Fluid Power Centre of Bath University and the Institution of Mechanical Engineers is gratefully acknowledged. Finally, the author would like to thank Will Richards for carrying out much of the groundwork of the package.

REFERENCES

1. CHU, Y. Digital Simulation of Continuous Systems. McGraw Hill, 1969.

2. BIRTWISTLE, G. Discrete Event Modelling on Simula. Macmillan Computer Science Series, 1979.

3. AMIES, G., LEVEK, R., STREUSSEL, D. Aircraft Hydraulic System Dynamic Analysis, Volume II - Transient Analysis (HYTRAN). Computer Program Technical Description. McDonnel Douglas Corporation. Report AFAPL-TR-76-43, 1977.

4. LEVEK, R., YOUNG, B. Aircraft Hydraulic System Dynamic Analysis, Volume VI - Staedy State Flow Analysis (SSFAN). Computer Program Technical Description. McDonnel Douglas Corporation. Report AFAPL-TR-76-43, 1977.

5. AMIES, G., LEVEK, R., STREUSSEL, D. Aircraft Hydraulic System Dynamic Analysis, Volume I - Transient Analysis (HYTRAN). Computer Program User Manual. McDonnel Douglas Corporation. Report AFAPL-TR-76-43, 1977.

6. BACKE, W. The hydraulics simulation program DSG. Proc. of the Instn. of Mechanical Engineers, Vol. 199, 1985.

7. GORDON, W.J., REISENFELD, R.F.
B-spline Curves and Surfaces.
Computer aided geometric design (Barnhill and Reisenfeld - editors). Academic Press, 1974.

8. McARTHUR, C.D. A User's Guide to CSMP. Scientific and Social Sciences Program Library, University of Edinburgh. Program Library Services No. 7, 1972.

9. ANONYMOUS Advanced Continuous Simulation Language (ACSL). User Guide/Reference Manual. Mitchell and Gauthier, 1975.

10. GEAR, C.W. Numerical Initial Value Problems in Ordinary Differential Equations. Prentice Hall, 1971.

11. ANONYMOUS Fortran IV. ANSI X3.9, 1966.

12. ANONYMOUS Fortran 77. ANSI X3.9, 1978.

13. ANONYMOUS User Manual for RSX-11M, Version 3.2, Volume 3a. The Taskbuilder (pp. 4.1-4.44). Digital Equipment Corporation, 1980.

14. TOMLINSON, S.P. The Hydraulic Automatic Simulation Package (HASP), User Manuals Volume 2 - Component Model Library. University of Bath, School of Engineering, Report No. 602, 1981.

15. CARNAHAN, B., LUTHER, H.A., WILKES, J.O.
Applied Numerical Methods. John Wiley and Sons, 1969.

16. KIMBARK, E.W. Power System Stability, Volume III - Synchronous Machines. John Wiley and Sons, 1964.

17. BOWNS, D.E., BONSON, L.A., RICHARDS, C.W., CANEY, K.
The simulation of Hydraulic Systems.
International Federation of Automatic Control, IFAC Symposium, Warsaw, May 1980.

18. HULL, S.R., BOWNS, D.E.
The Development of an Automatic Procedure for the Digital Simulation of Hydraulic Systems.
Proc. of the Institution of Mechanical Engineers, Volume 199 No. B1, 1985.

19. ANONYMOUS Fortran Dialects. Cranfield Institute of Technology Computer Centre, April 1979.

20. NIBLETT, B. Legal Aspects of Numerical Software. Proceedings of the IFIP TC-2 Working Conference on Performance Evaluation of Numerical Software, December 1978. North-Holland 1979.

21. TAPPER, C. Legal Remedies for Misrepresentation of Software. Proceedings of the IFIP TC-2 Working Conference

on Performance Evaluation of Numerical Software, December 1978. North-Holland 1979.

22. LORD HAILSHAM Halsbury's Laws of England, Fourth Edition, Volume 35. Patent and Inventions paras.301-733. Produced under the editorship of Lord Hailsham of St. Marylebone, Butterworths 1981.

23. LORD HAILSHAM Halsbury's Laws of England, Fourth Edition, Volume 9. Contract paras.201-699, Copyright paras.801-970. Produced under the editorship of Lord Hailsham of St. Marylebone, Butterworths 1974.

24. LORD HAILSHAM Halsbury's Laws of England, Fourth Edition, Volume 31. Misrepresentation and Fraud paras. 1001-1137. Produced under the editorship of Lord Hailsham of St. Marylebone, Butterworths 1980.

25. LORD HAILSHAM Halsbury's Laws of England, Cumulative Supplement, Fourth Edition 1979, Volume 9 paras. 201-699. Produced under the editorship of Lord Hailsham of St. Marylebone, Butterworths 1979.

26. ELGERD, O.I. Electric Energy System Theory: An Introduction.

John Wiley and Sons, 1971.

27. KIMBARK, E.W. Power System Stability, Volume II - Power Circuit Breakers and Protective Relays. John Wiley and Sons, 1965.

28. JOHNSON, L.W., RIESS, R.D.

Numerical Analysis. Addison Wesley, 1982.

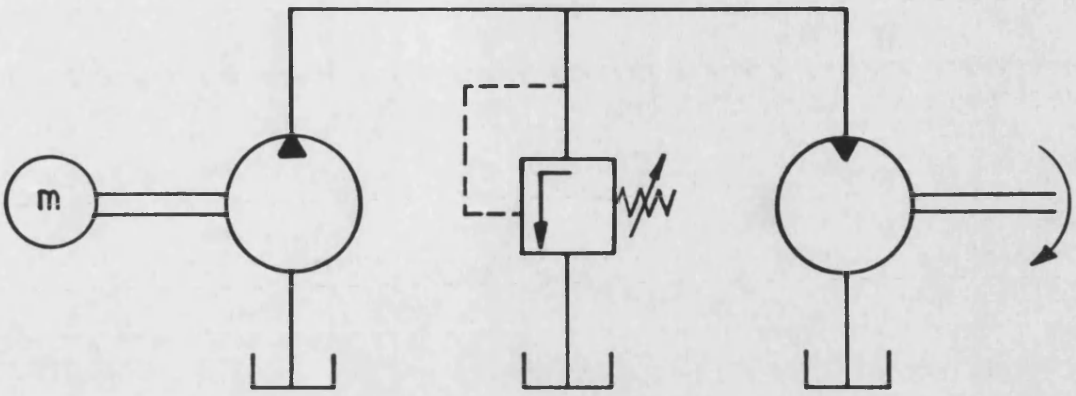


FIG. 1.1 An example of an hydraulic circuit to be simulated

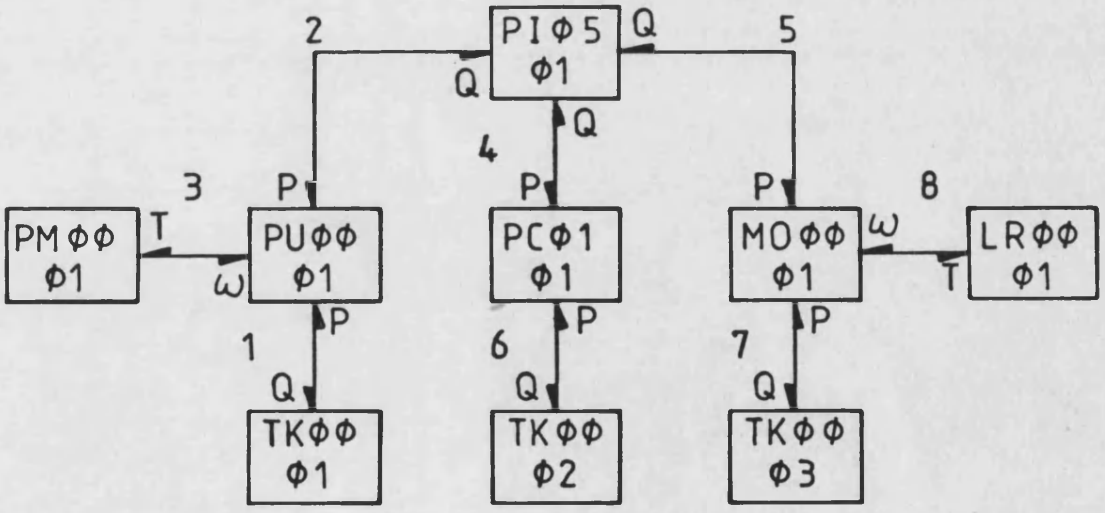


FIG. 1.2 The circuit linking diagram corresponding to figure 1.1

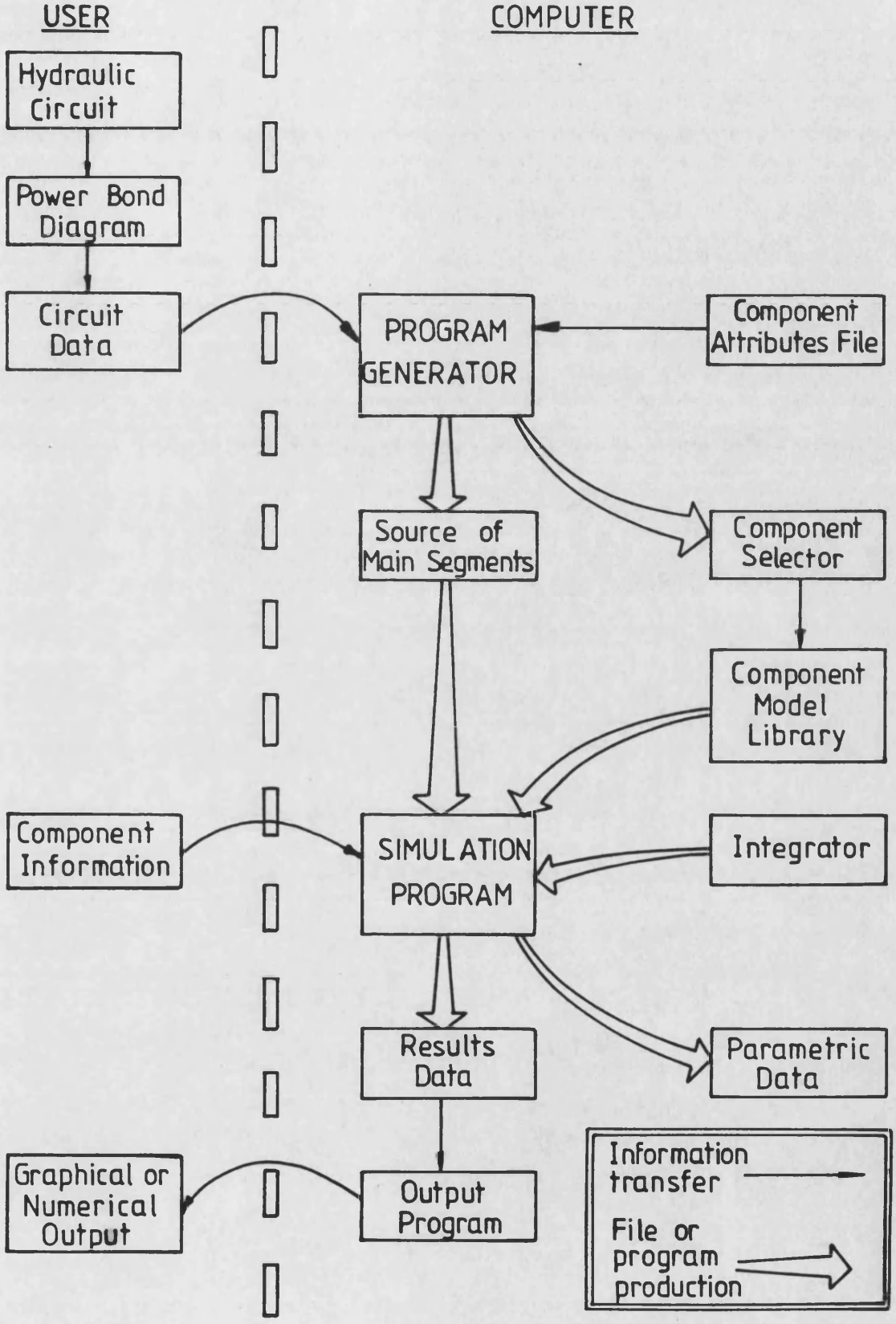


FIG. 2.1 A schematic of the structure of HASP

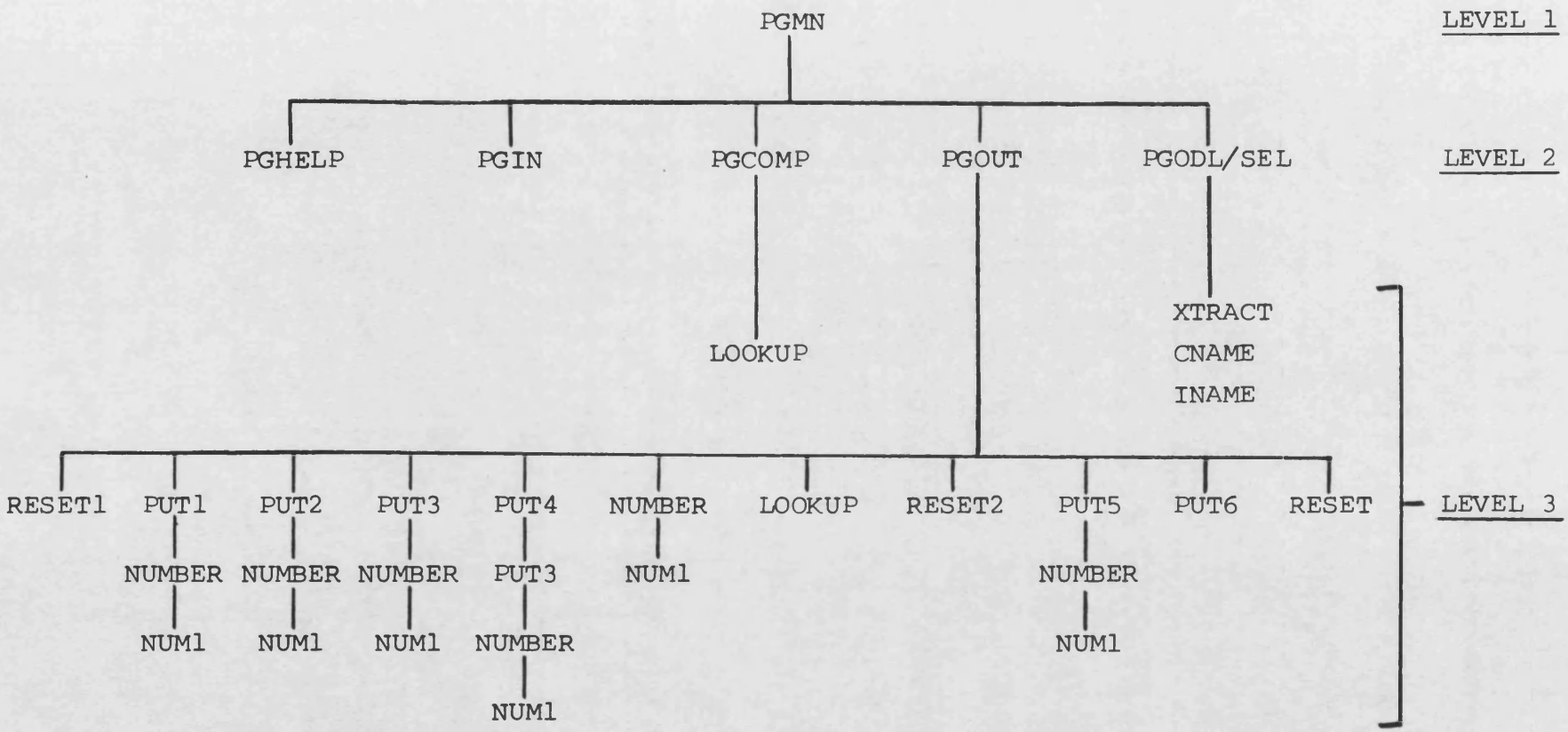


FIG. 3.1 The three levels of Program Generator segments

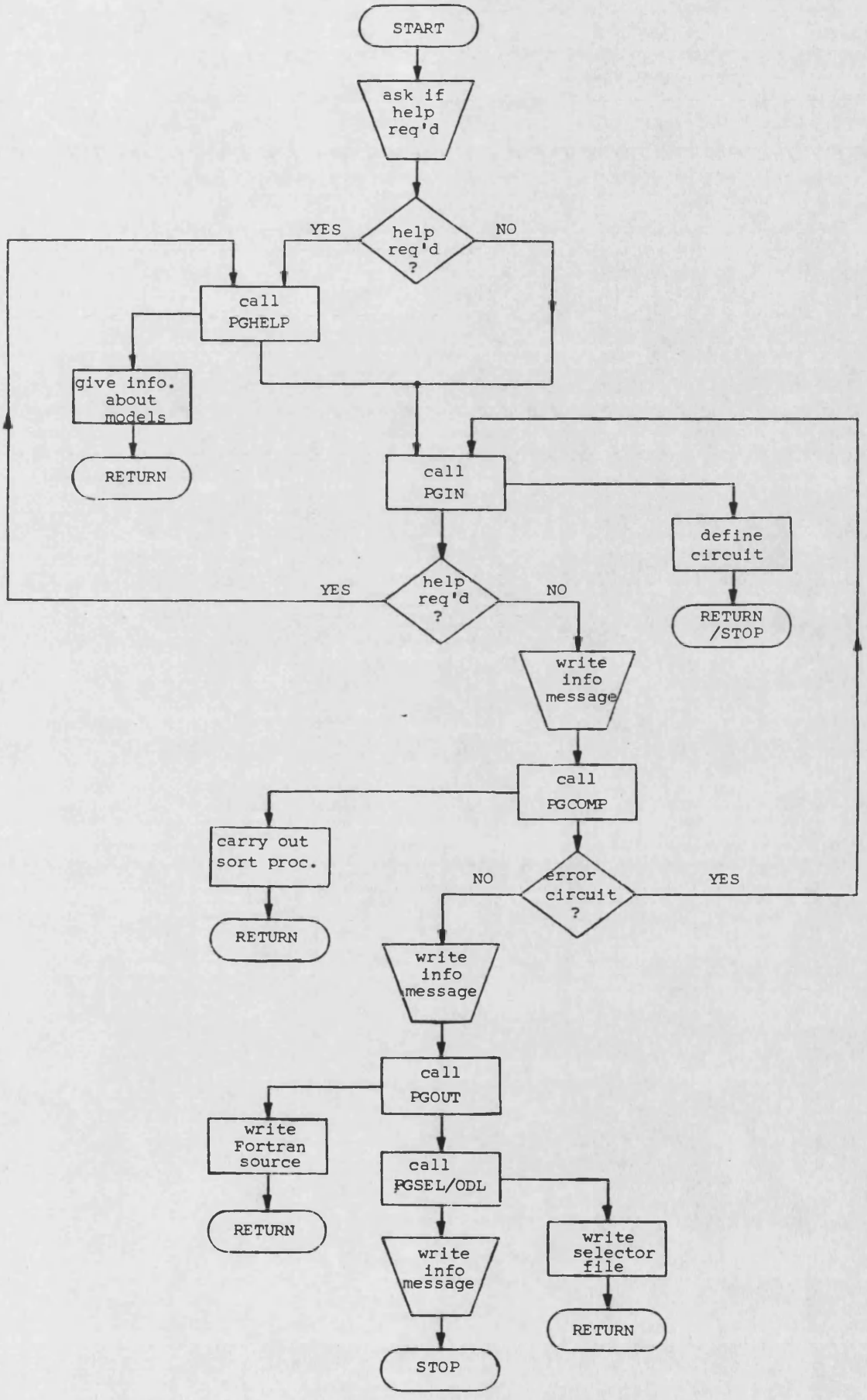


FIG. 3.2 Flow diagram for Program Generator main segment

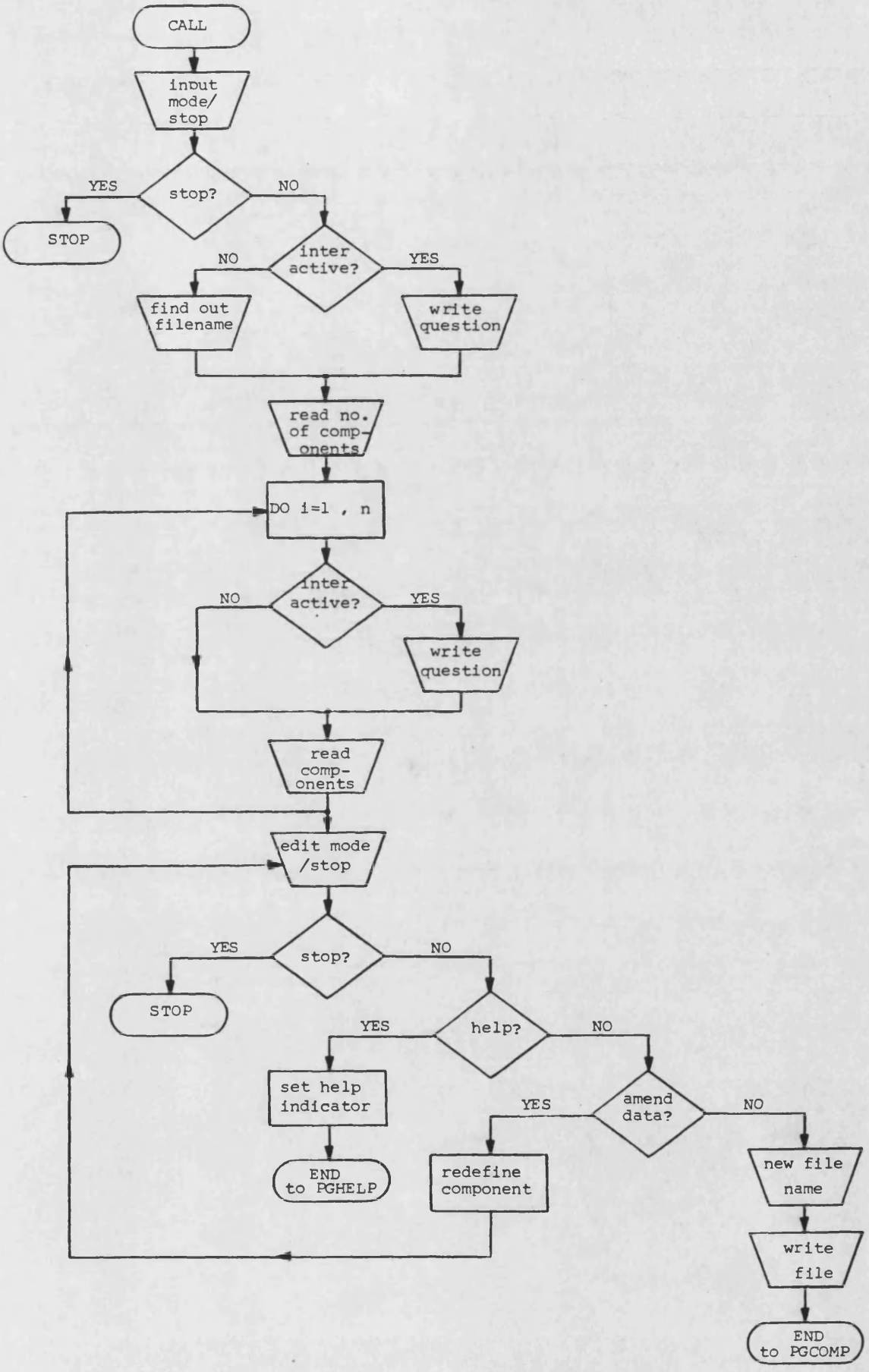


FIG. 3.3 Flow diagram for subroutine PGIN

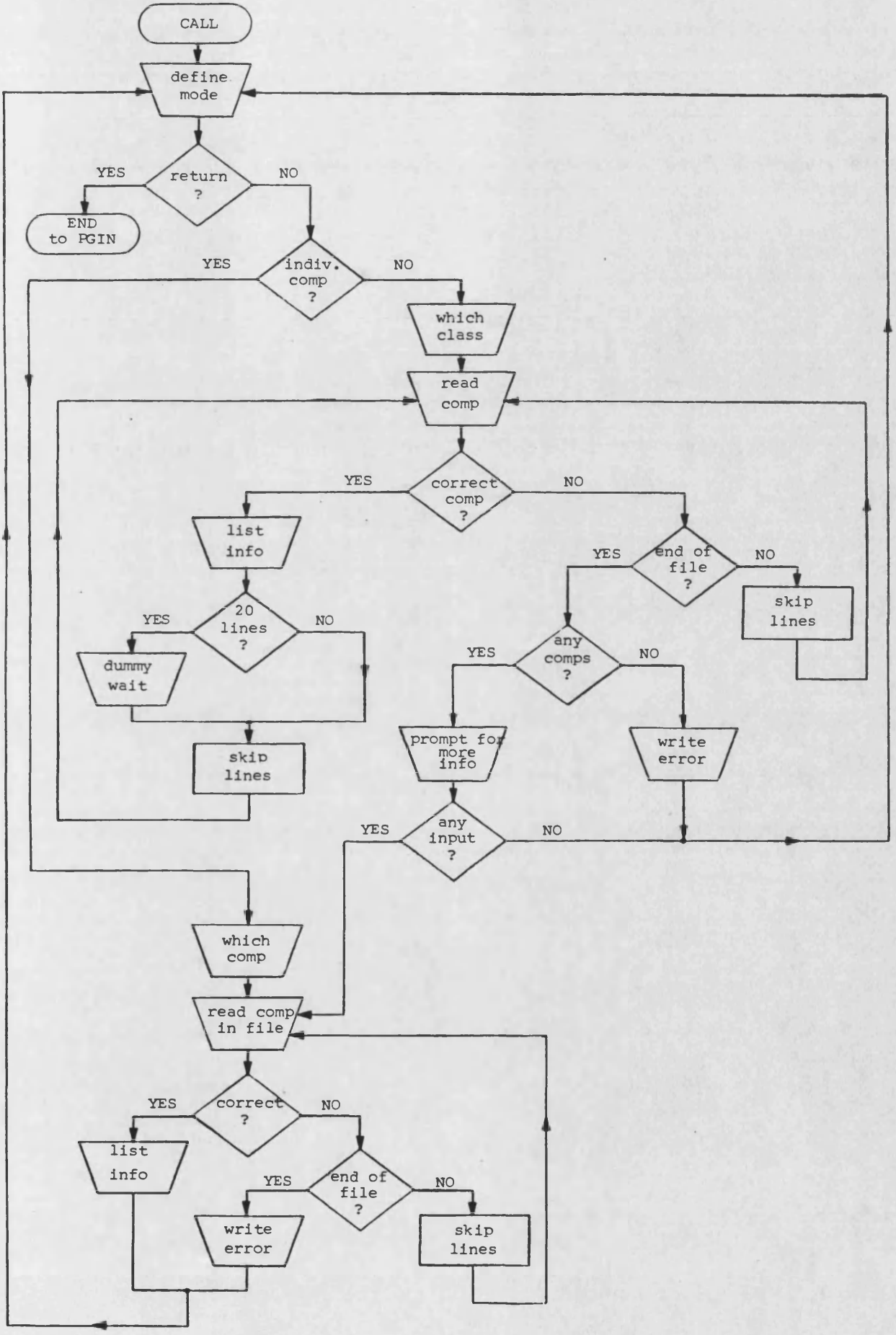


FIG. 3.4 Flow diagram for subroutine PGHELP

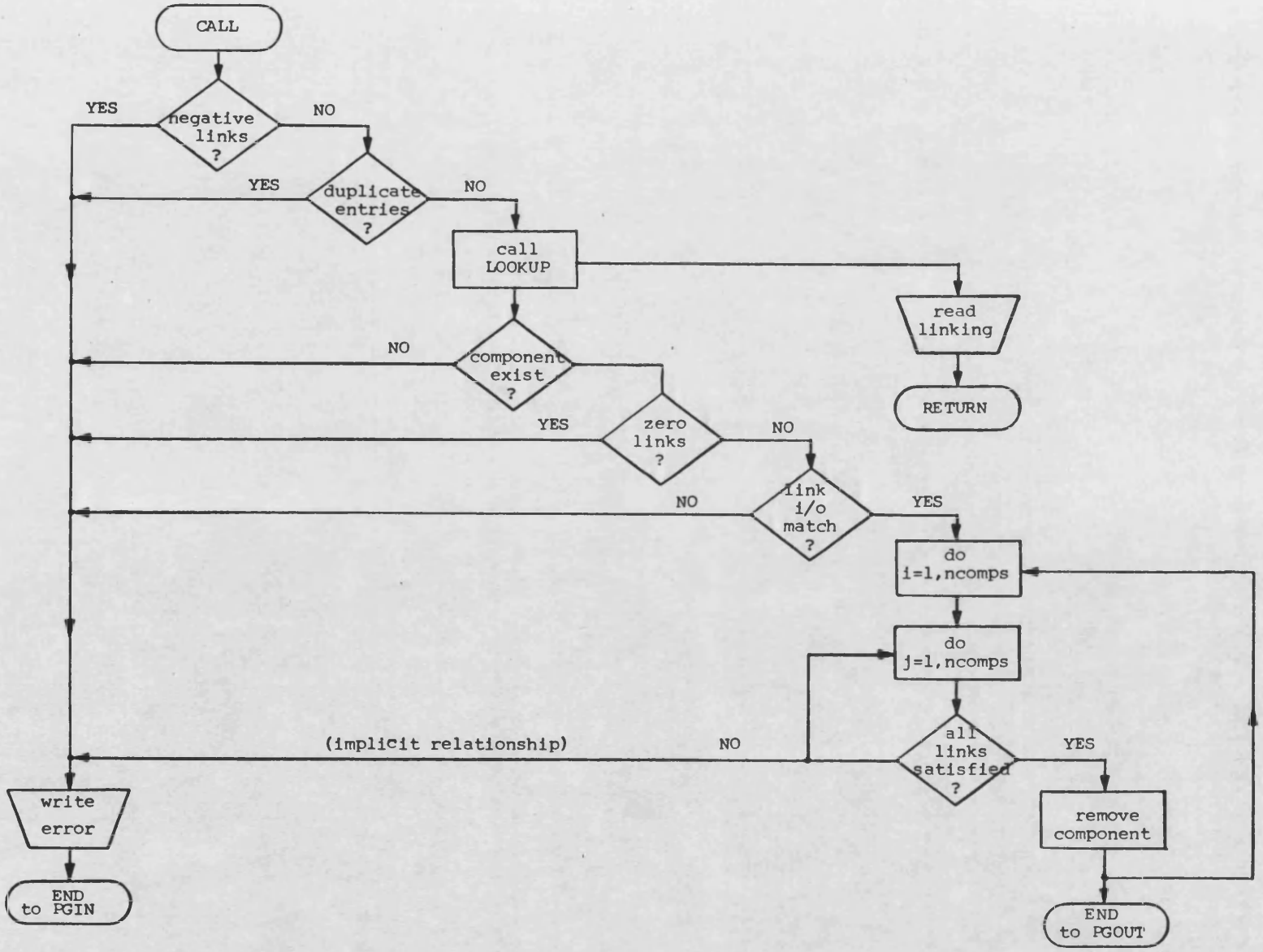


FIG. 3.5 Flow diagram for PGCOMP

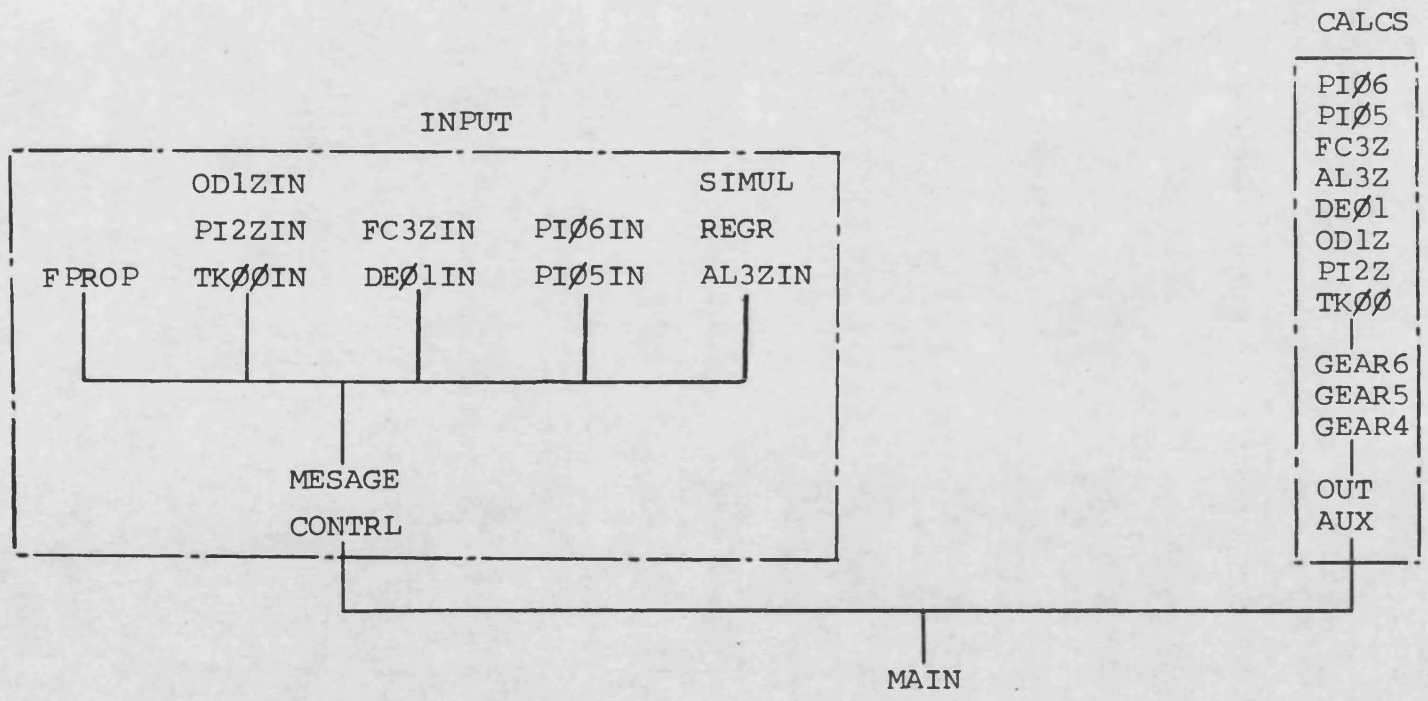


FIG. 3.6 An overlay tree for a simulation program

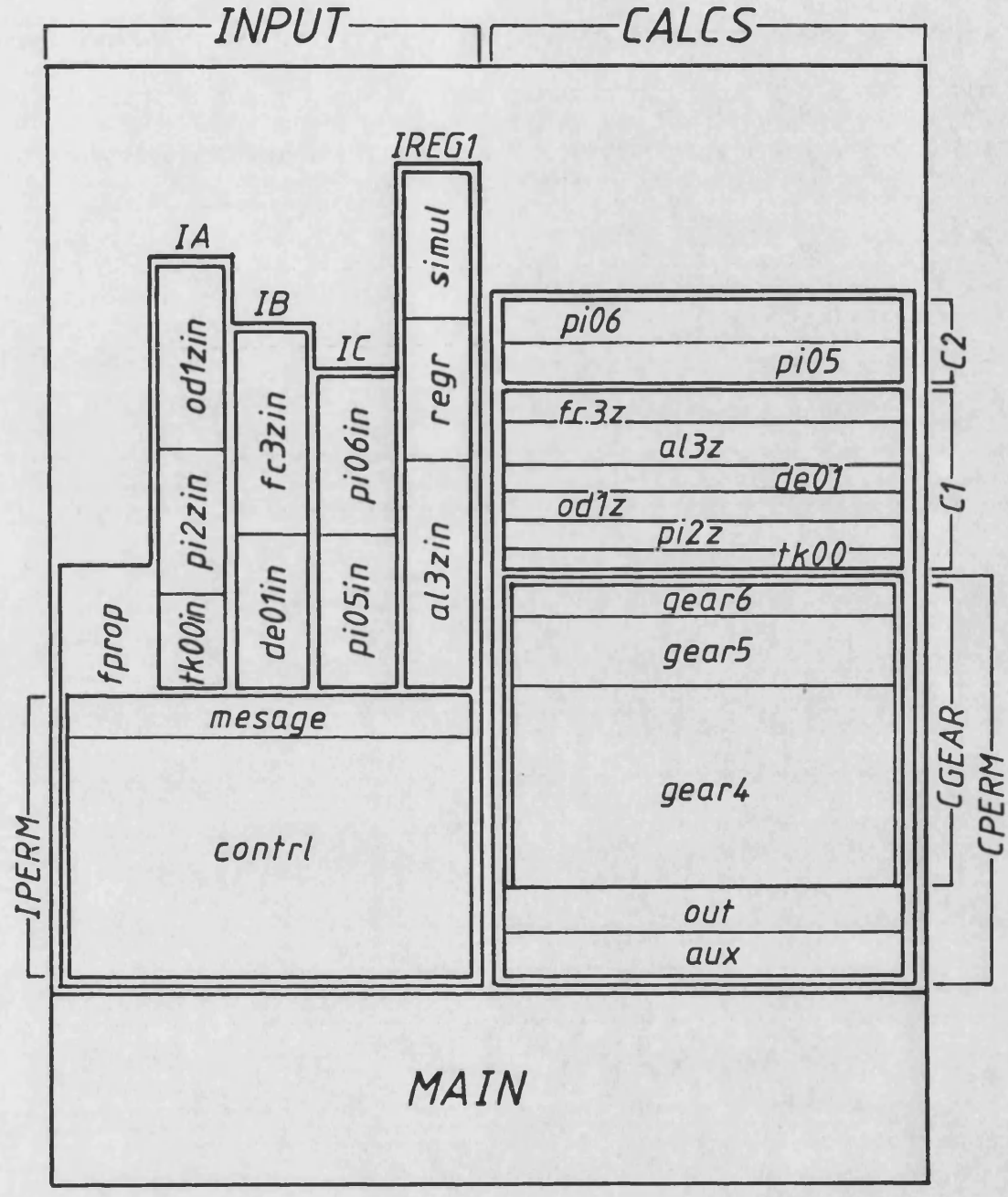


FIG. 3.7 A memory diagram for a simulation program

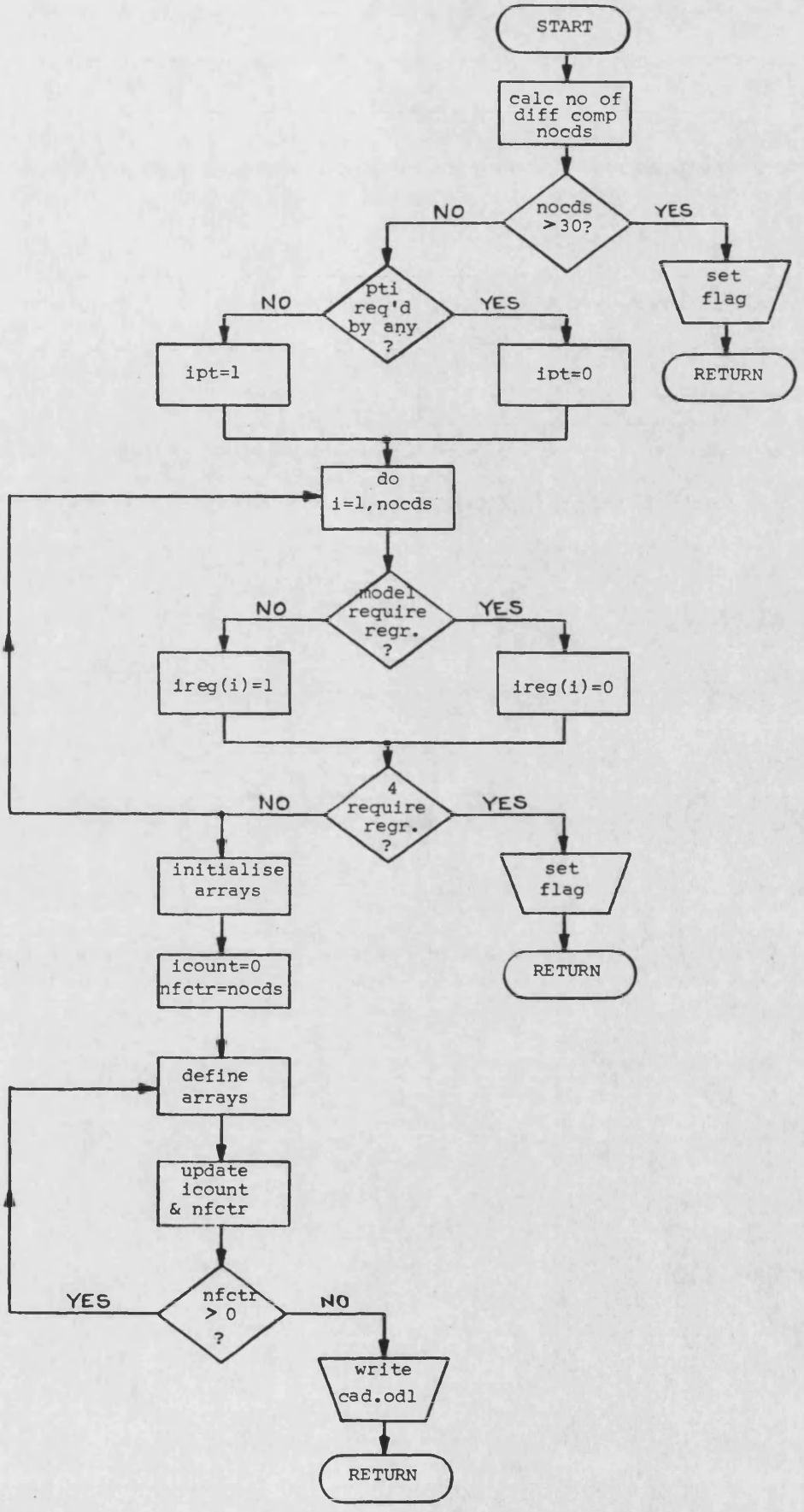


FIG. 3.8 Flow diagram for subroutine PGODL

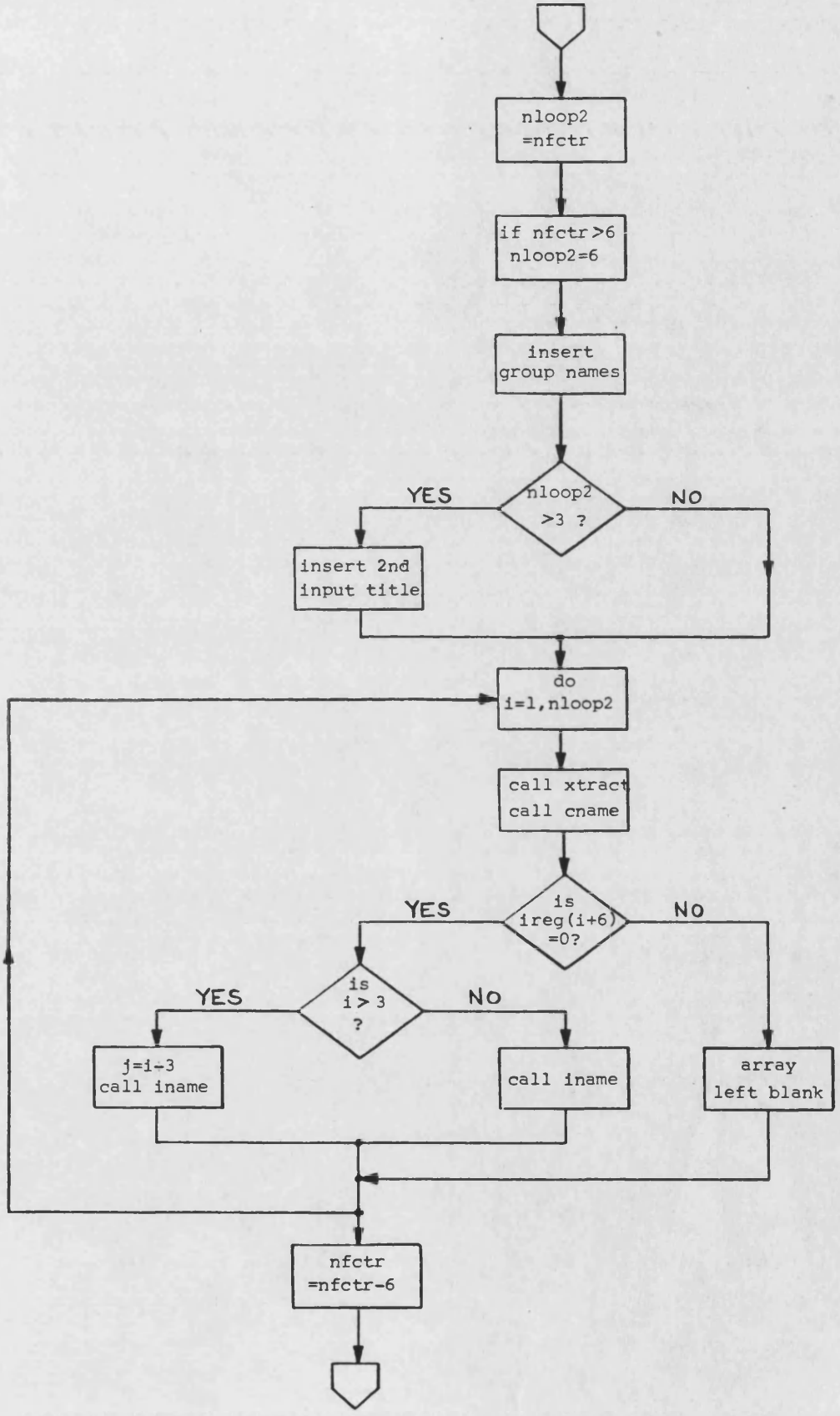


FIG. 3.9 Flow diagram for the definition of the logical arrays in subroutine PGODL

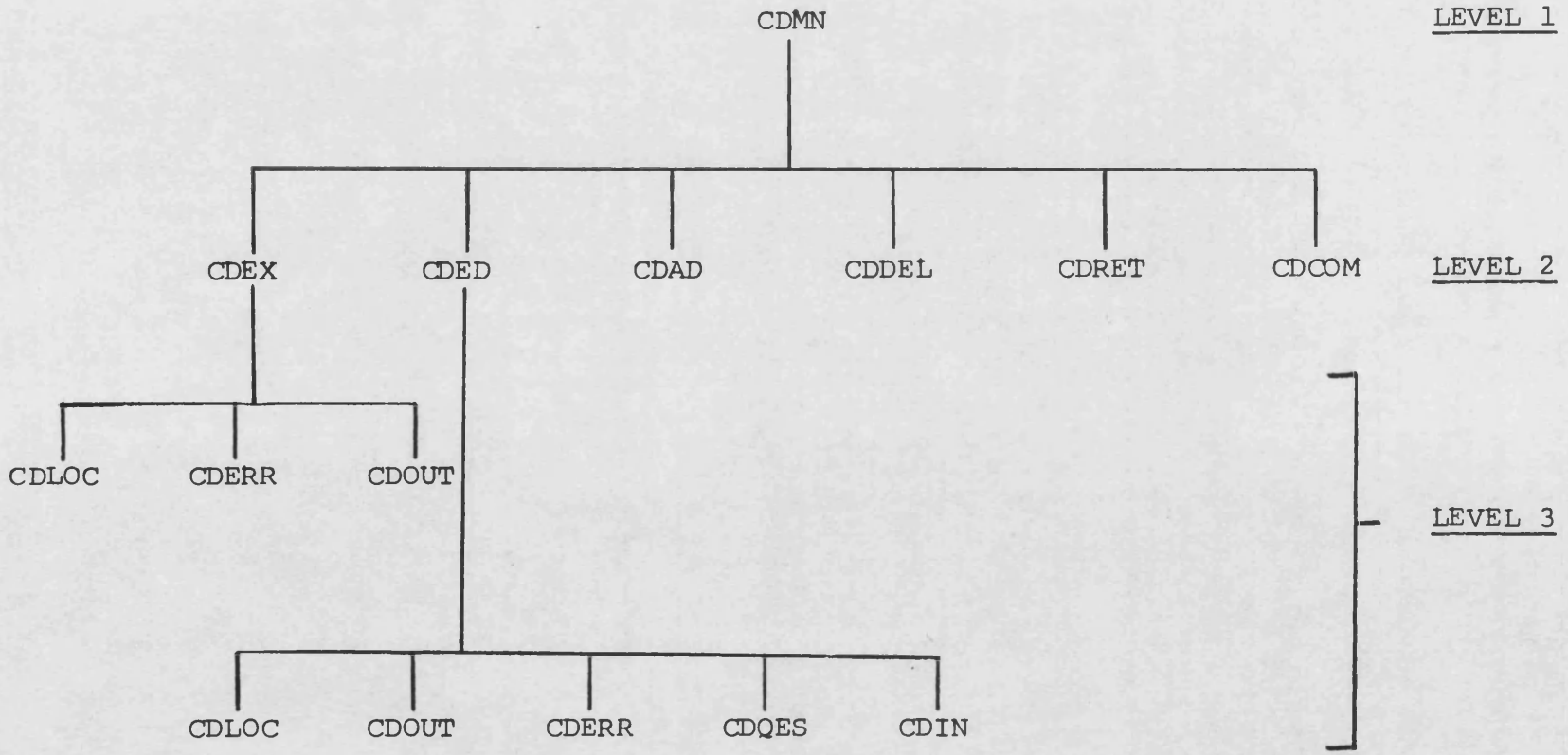


FIG. 3.10 The three levels of the component attributes file editor

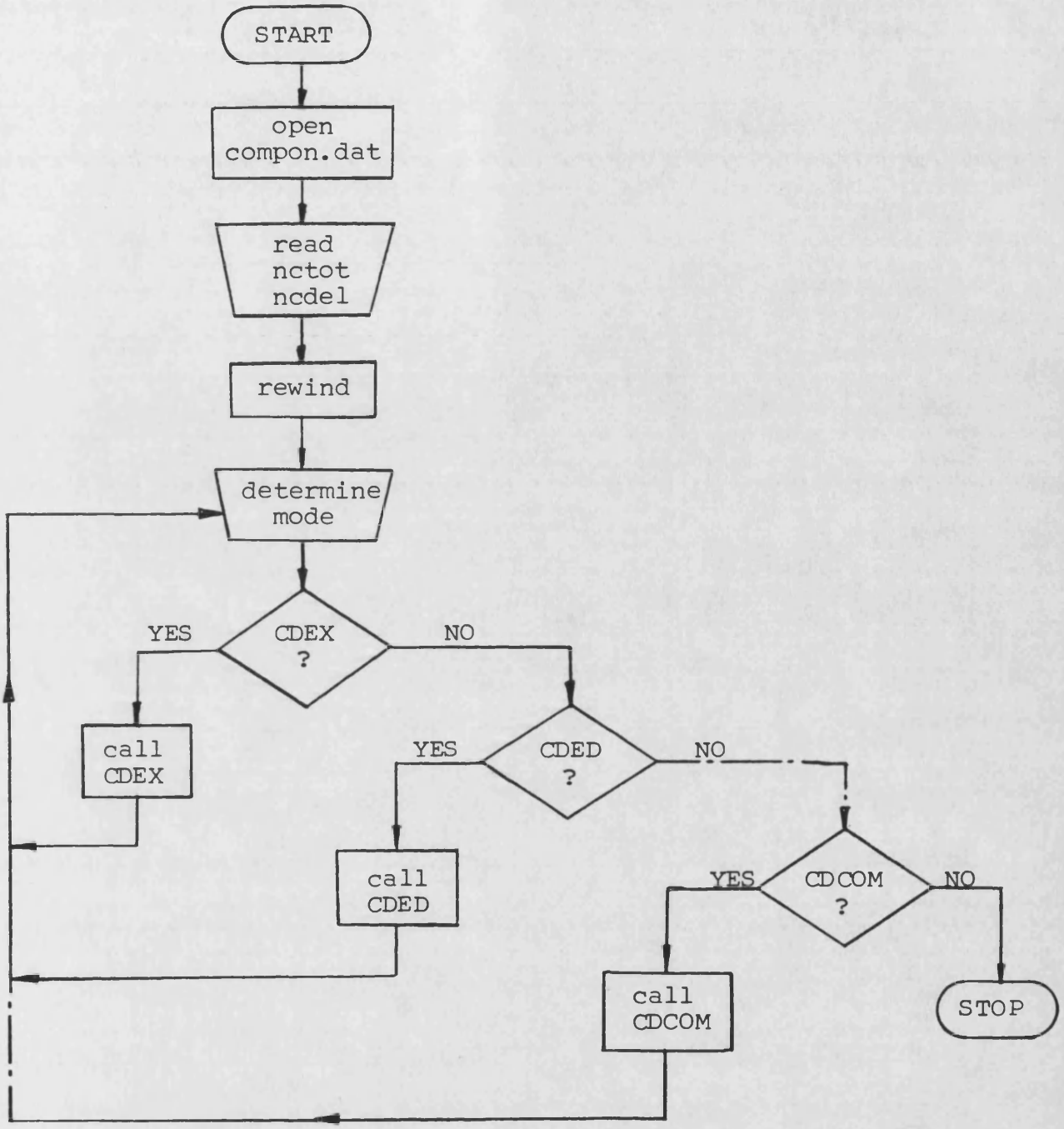


FIG 3.11 Flow diagram for component attributes file editor main segment

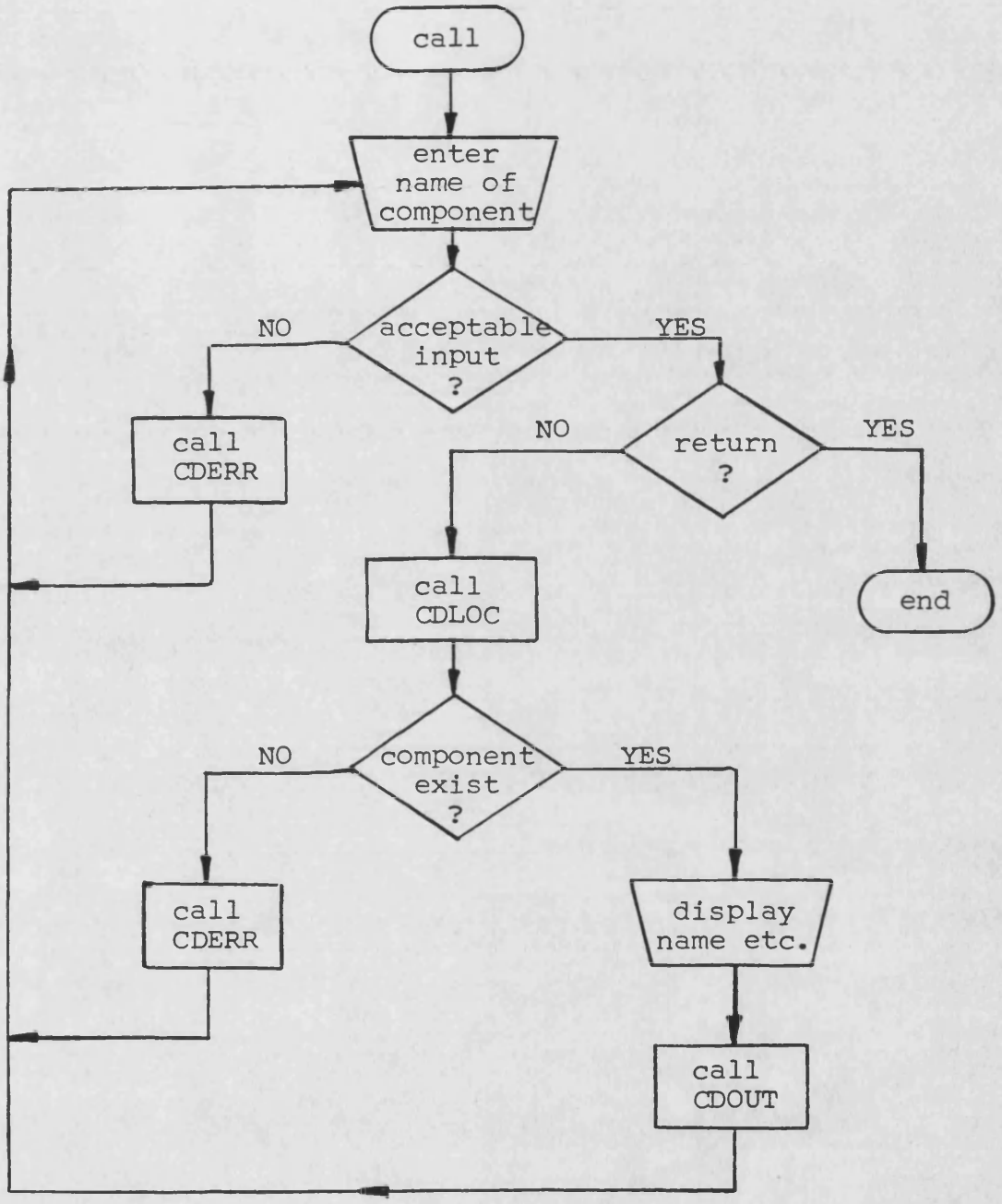


FIG. 3.12 Flow diagram for subroutine CDEX

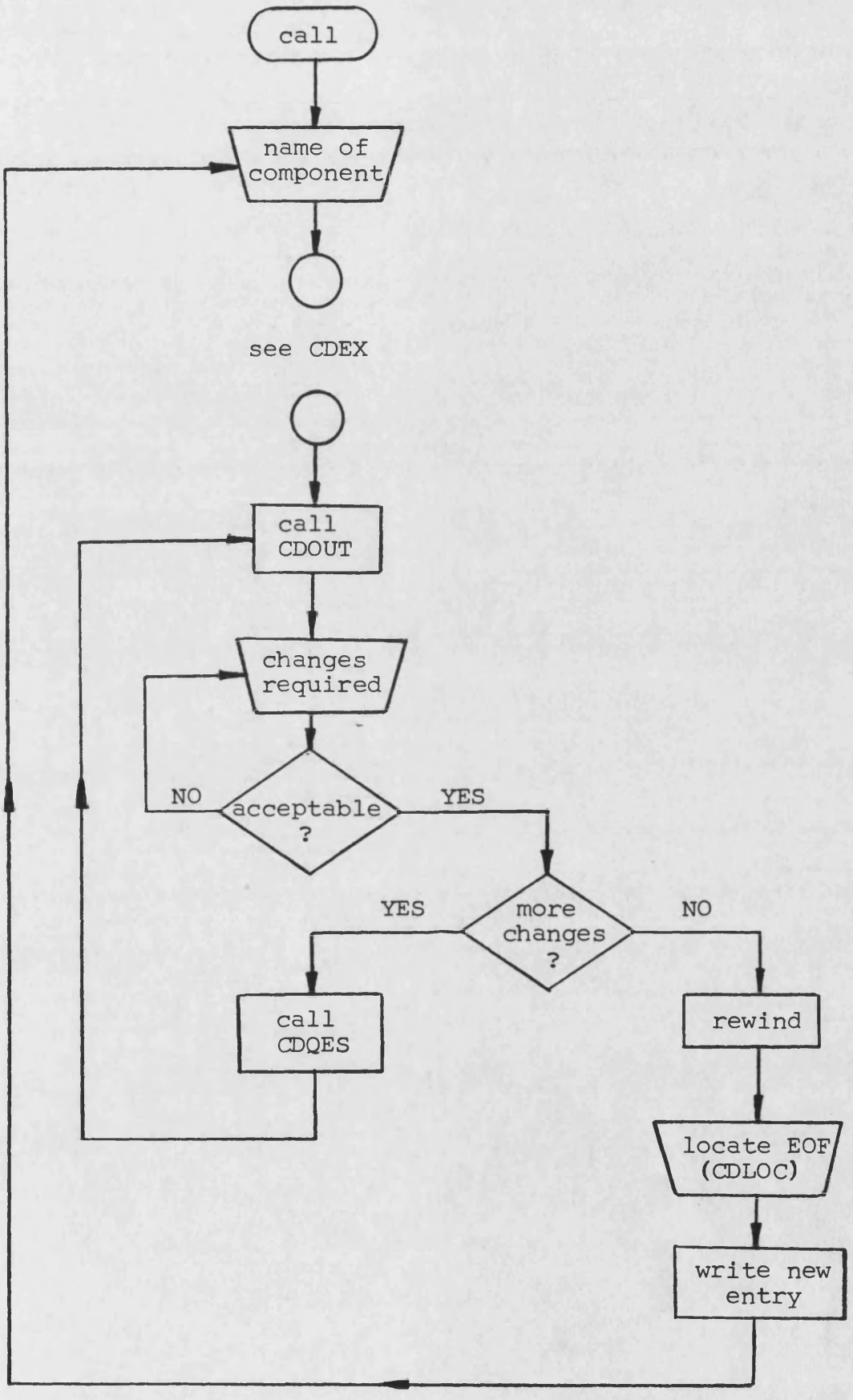


FIG. 3.13 Flow diagram for subroutine CDED

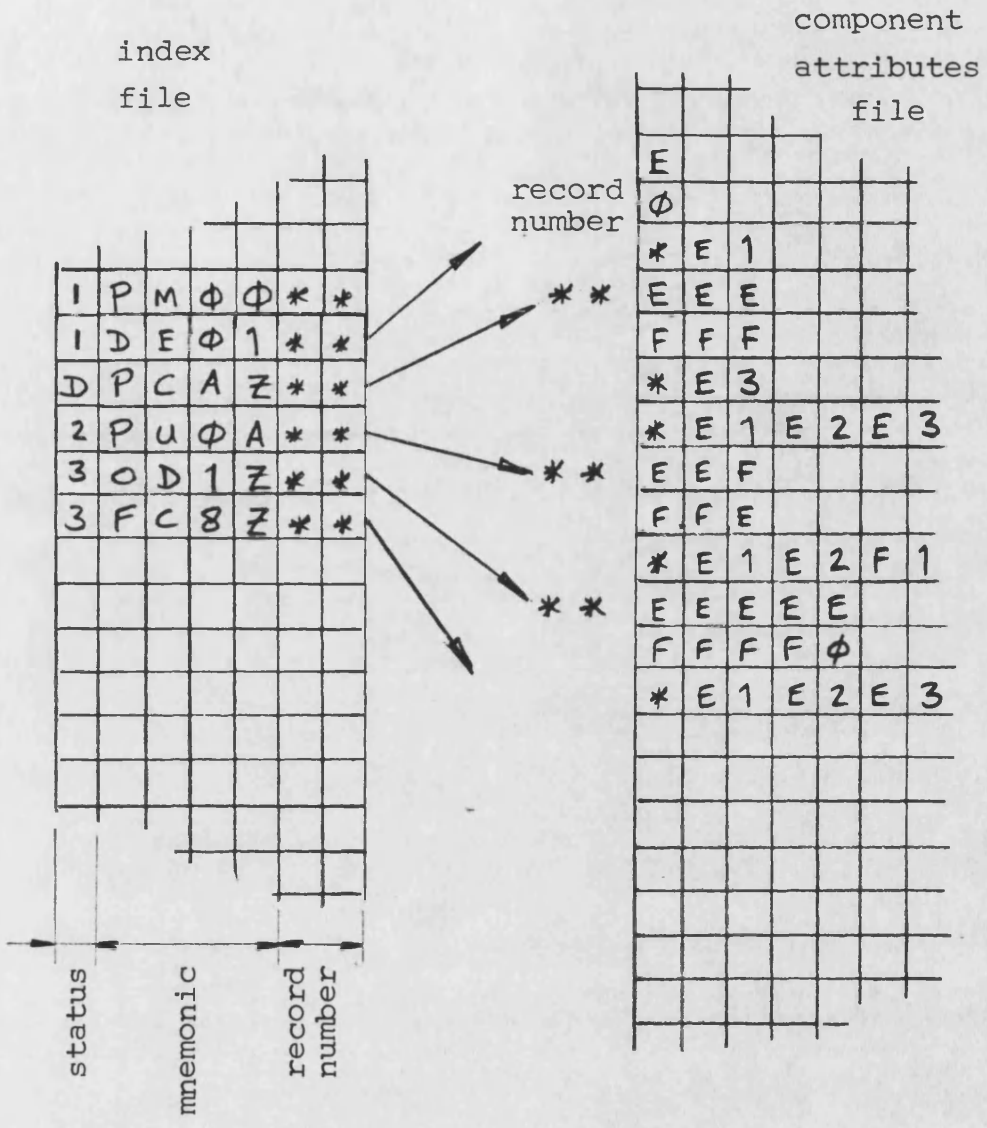


FIG. 3.14 An example of an indexed file structure for the component attributes file

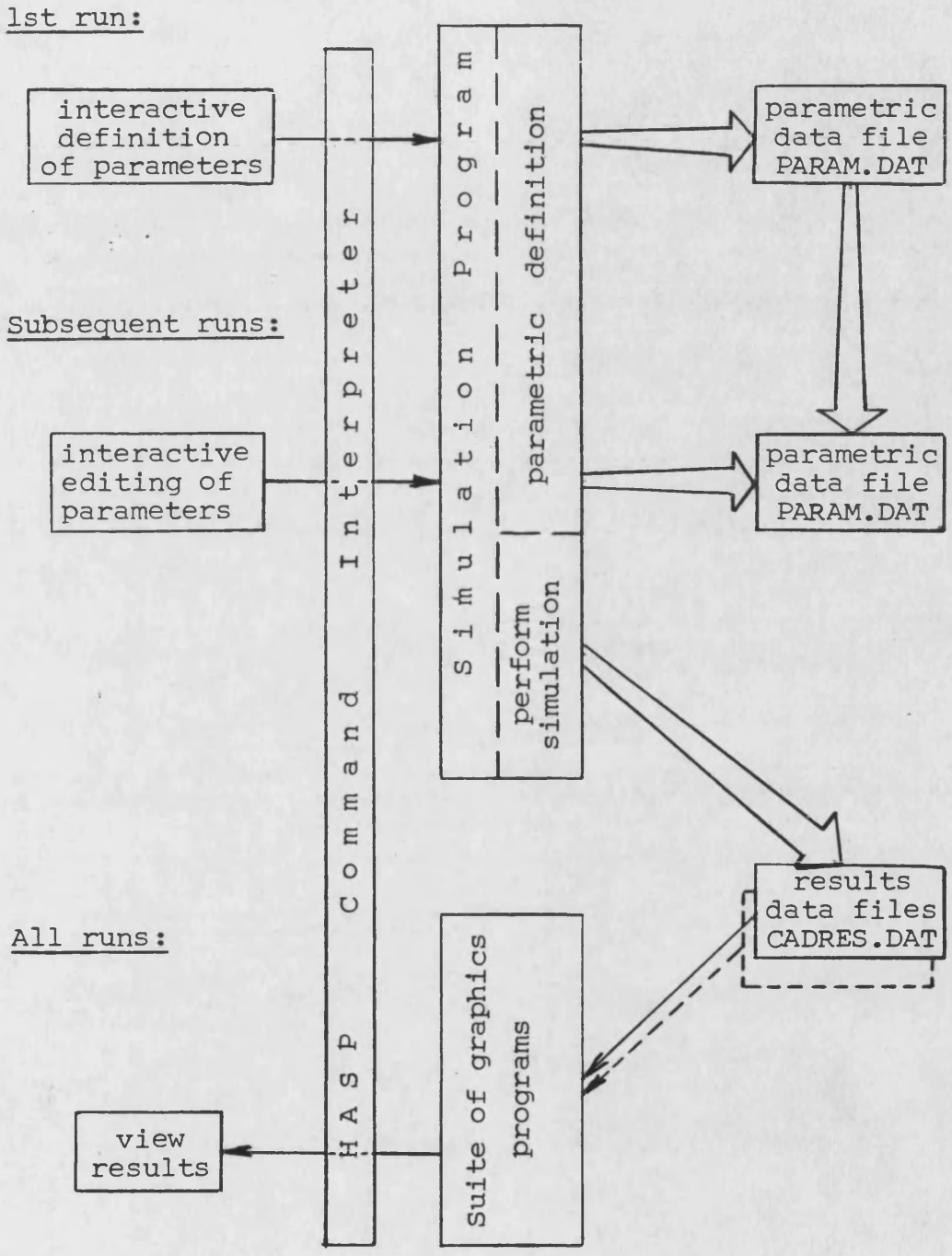


FIG. 4.1 A schematic of the interaction between the user and the simulation program

```
Type internal diameter of pipe in mm
20
Type pipe length in m
1000

*** value is outside normal working limits ***

Type pipe length in m
1000
Type 0 to input effective bulk modulus
     1 to calculate a value based on pipe dimensions
1
Select a pipe material from the following options:
.
.
.
```

FIG. 4.2 Typical question and answer sequence

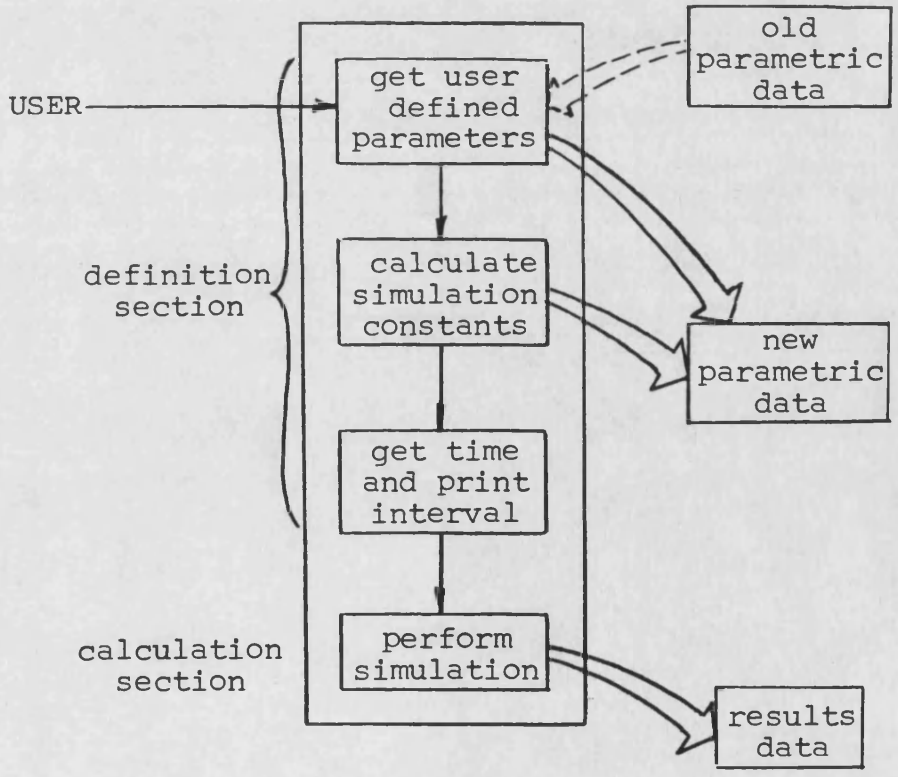


FIG. 4.3 A schematic of the standard simulation program

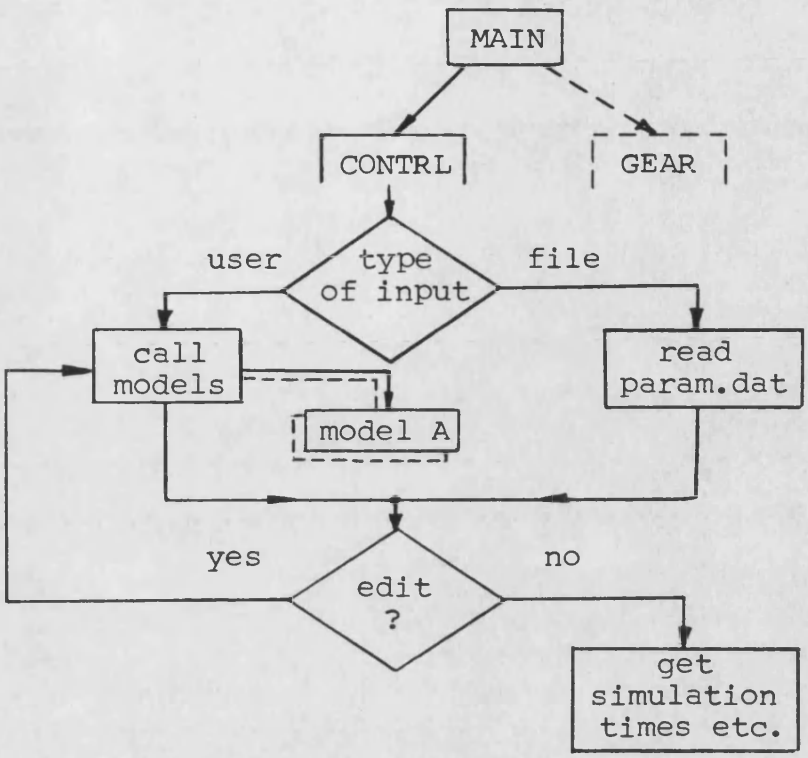


FIG. 4.4 Flow diagram of the parameter definition section

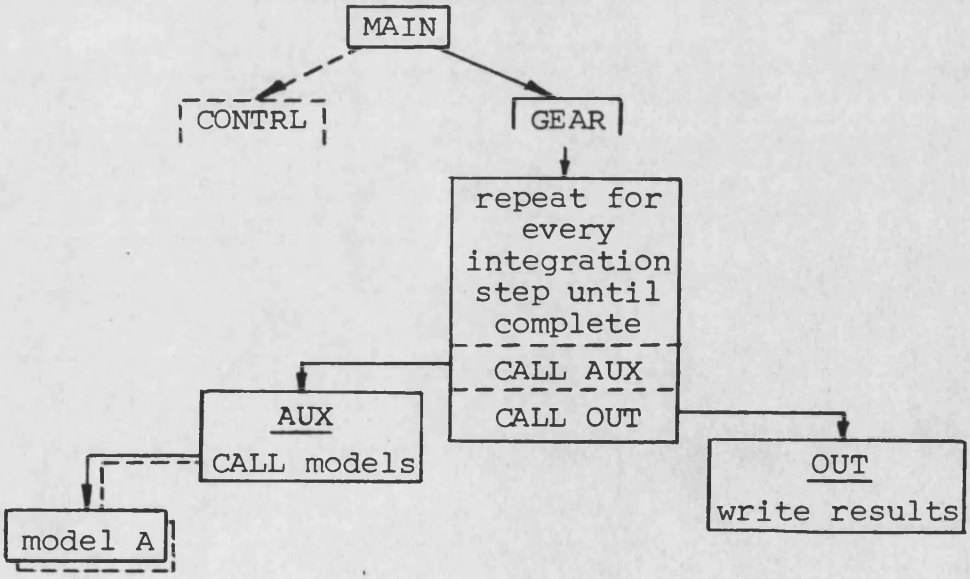


FIG. 4.5 Flow diagram of the calculation process

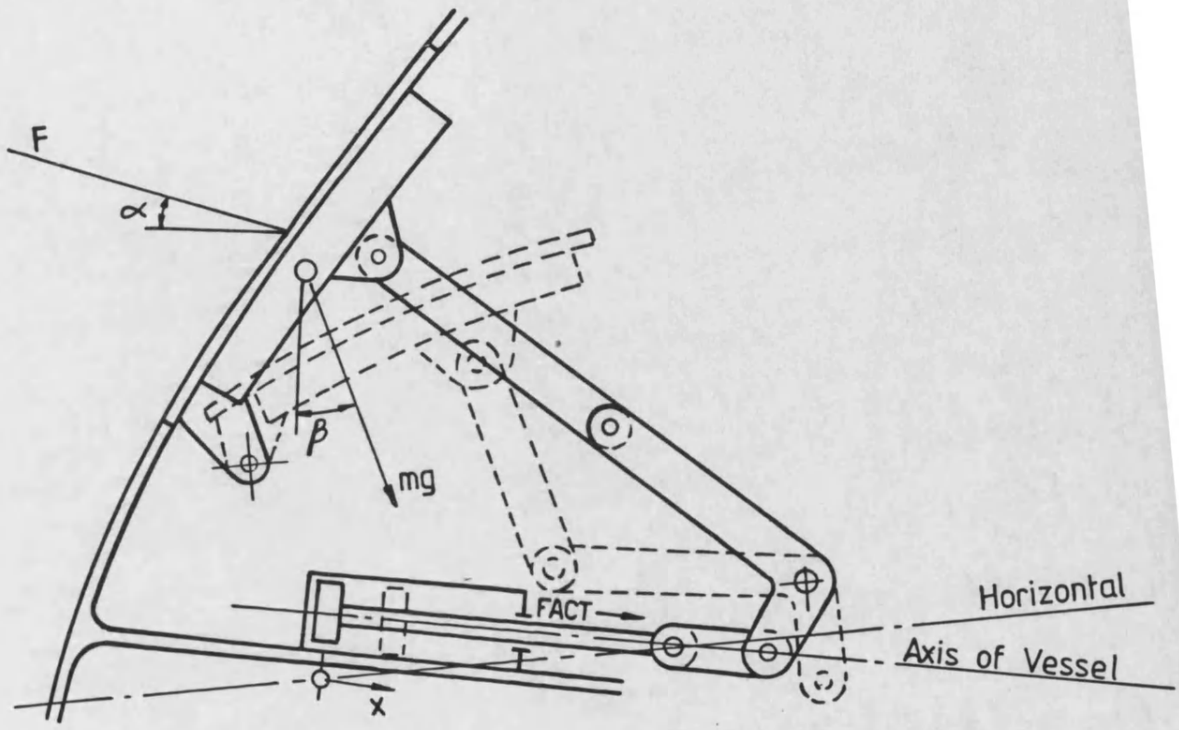


FIG. 4.6 Actuator with a mechanical linkage

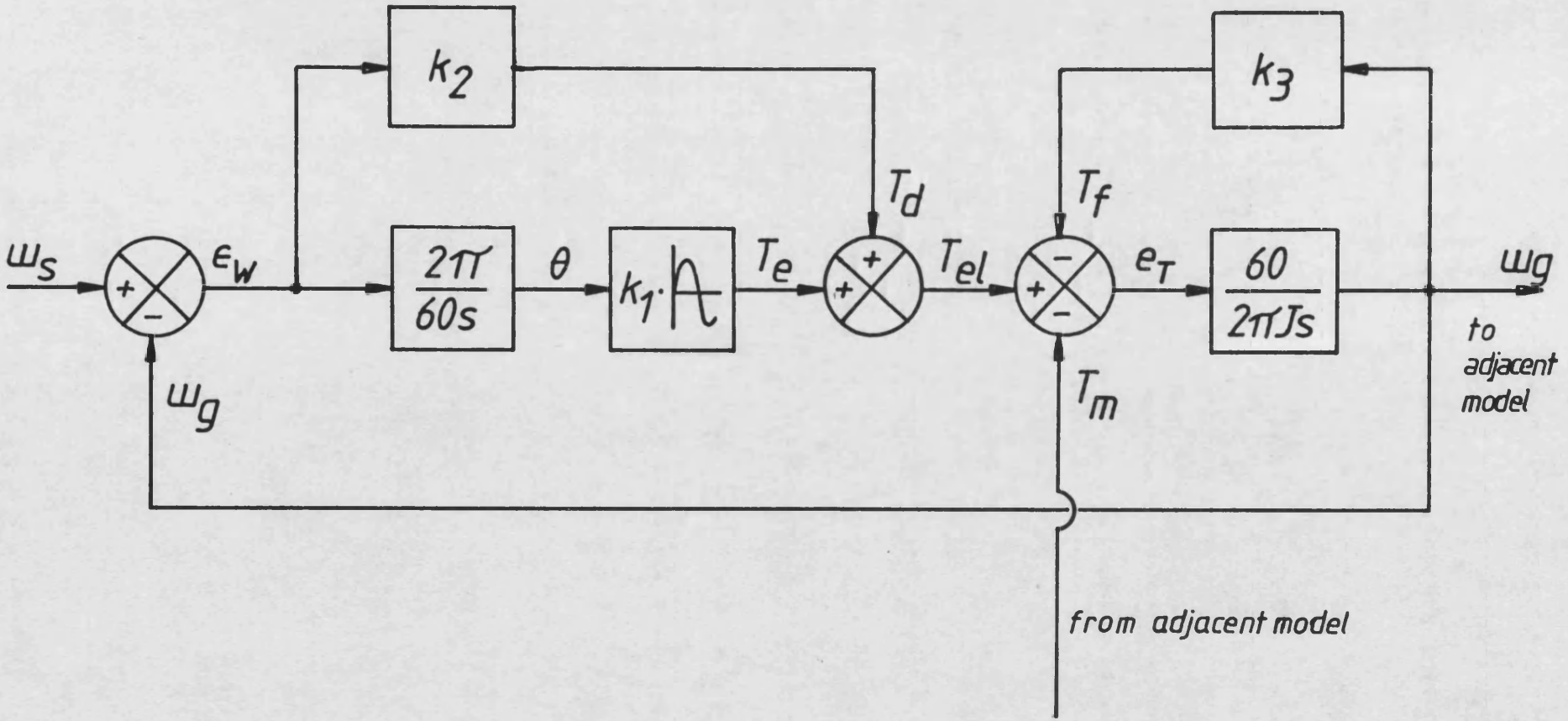


FIG. 4.7 Control block diagram of model GE1Z

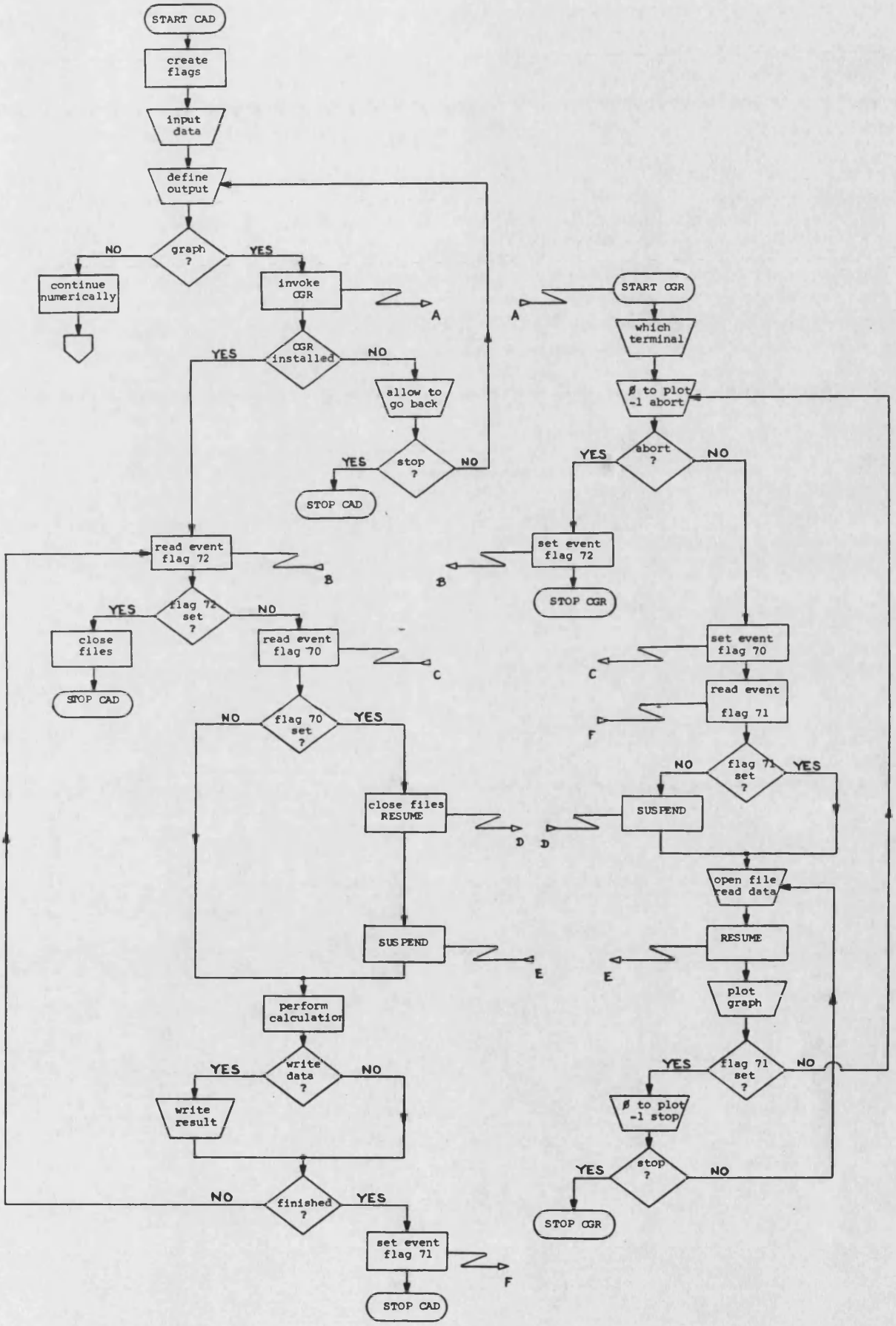


FIG. 4.8 The simulation program with interrupt graphics

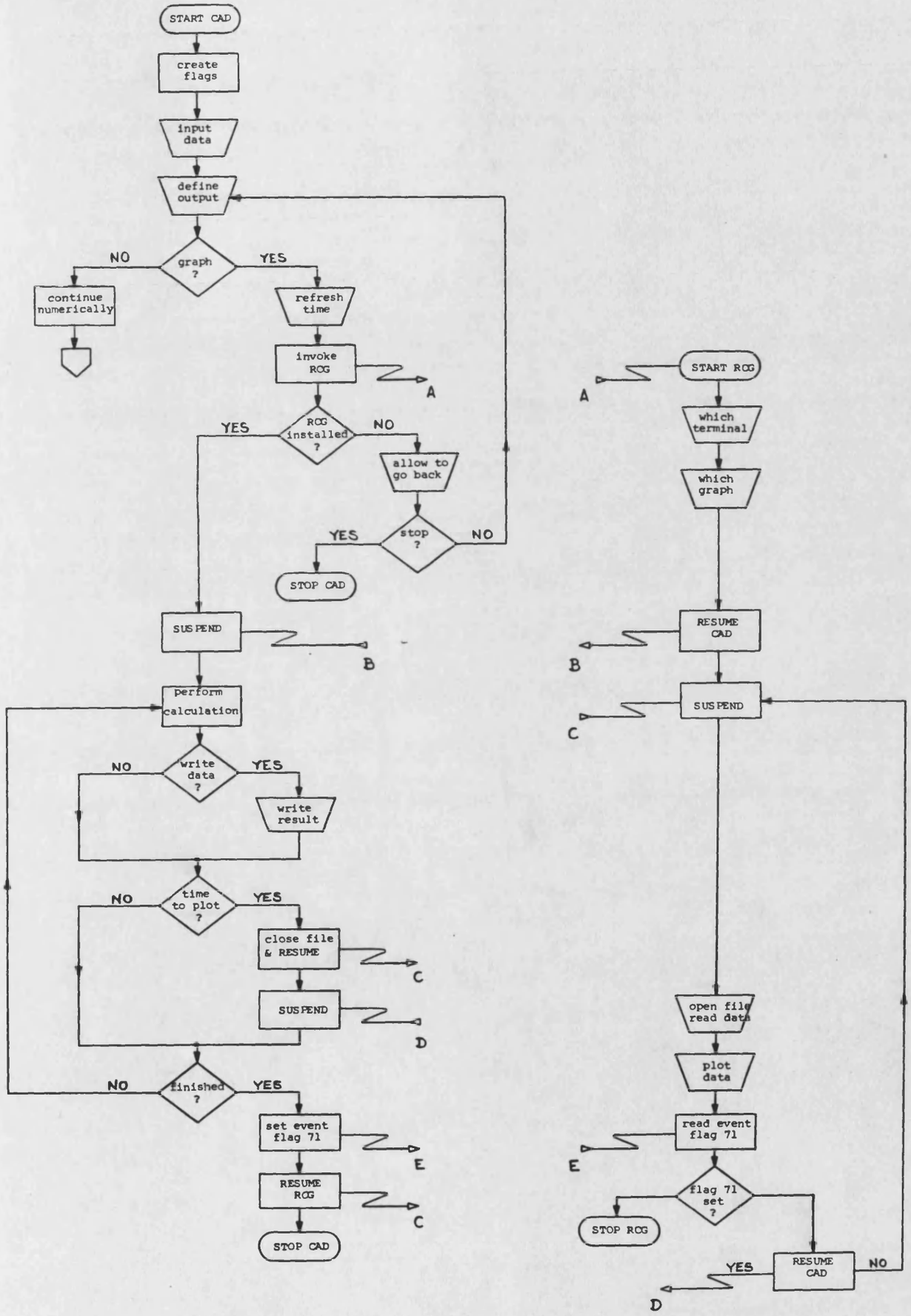


FIG. 4.9 The simulation program with refresh graphics

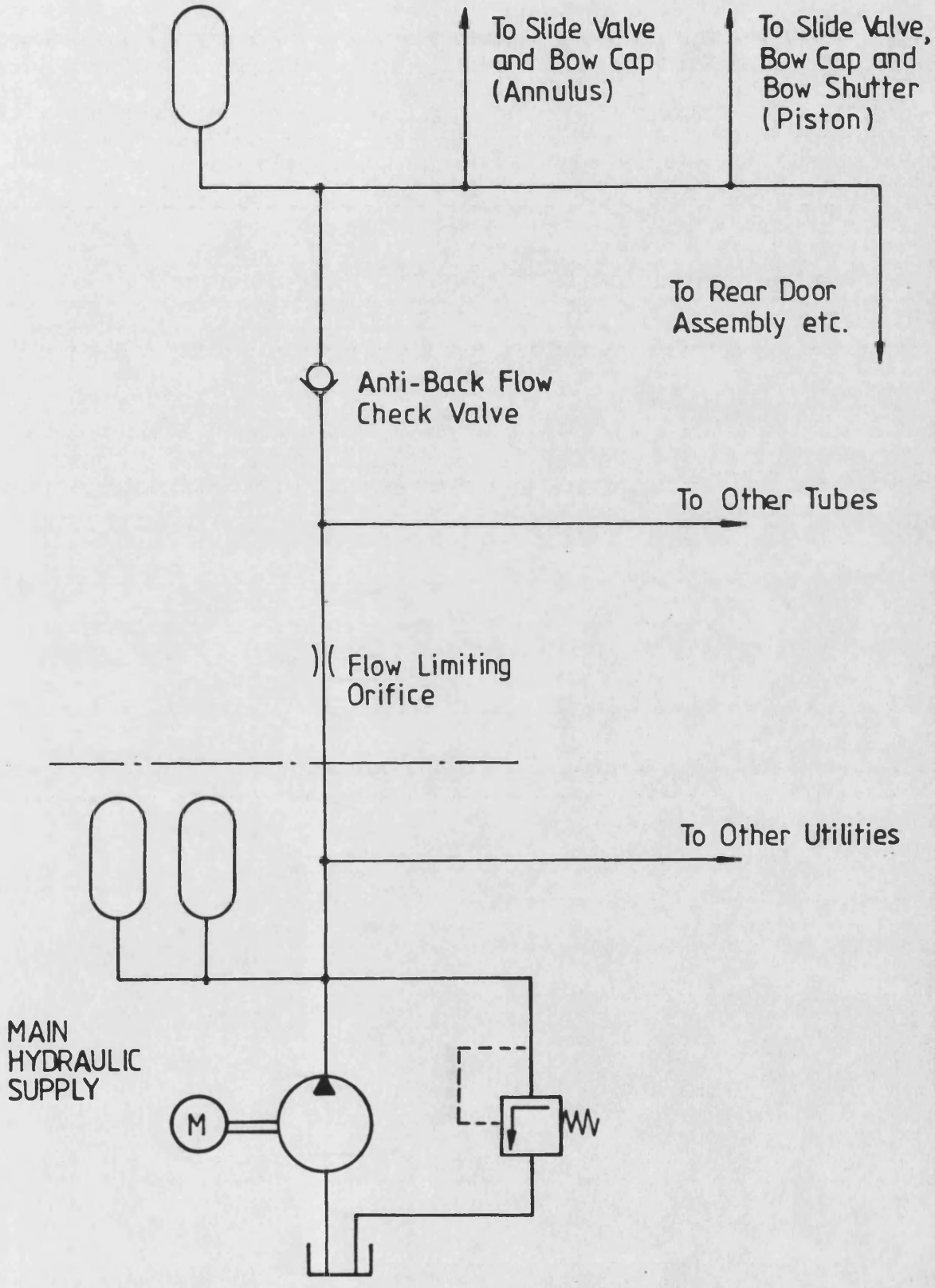


FIG. 5.1 The supply system

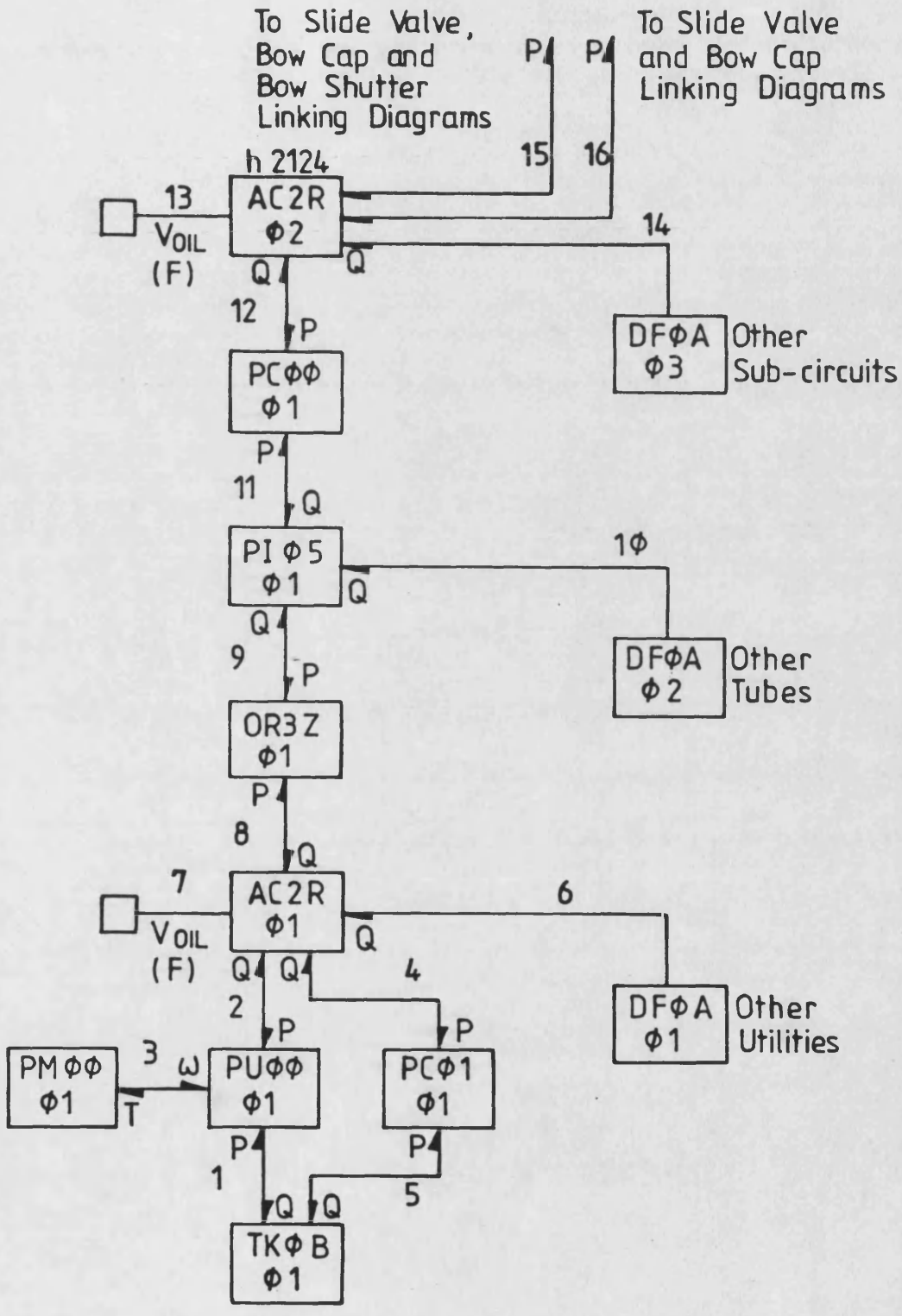


FIG. 5.2 The linking diagram for the supply system

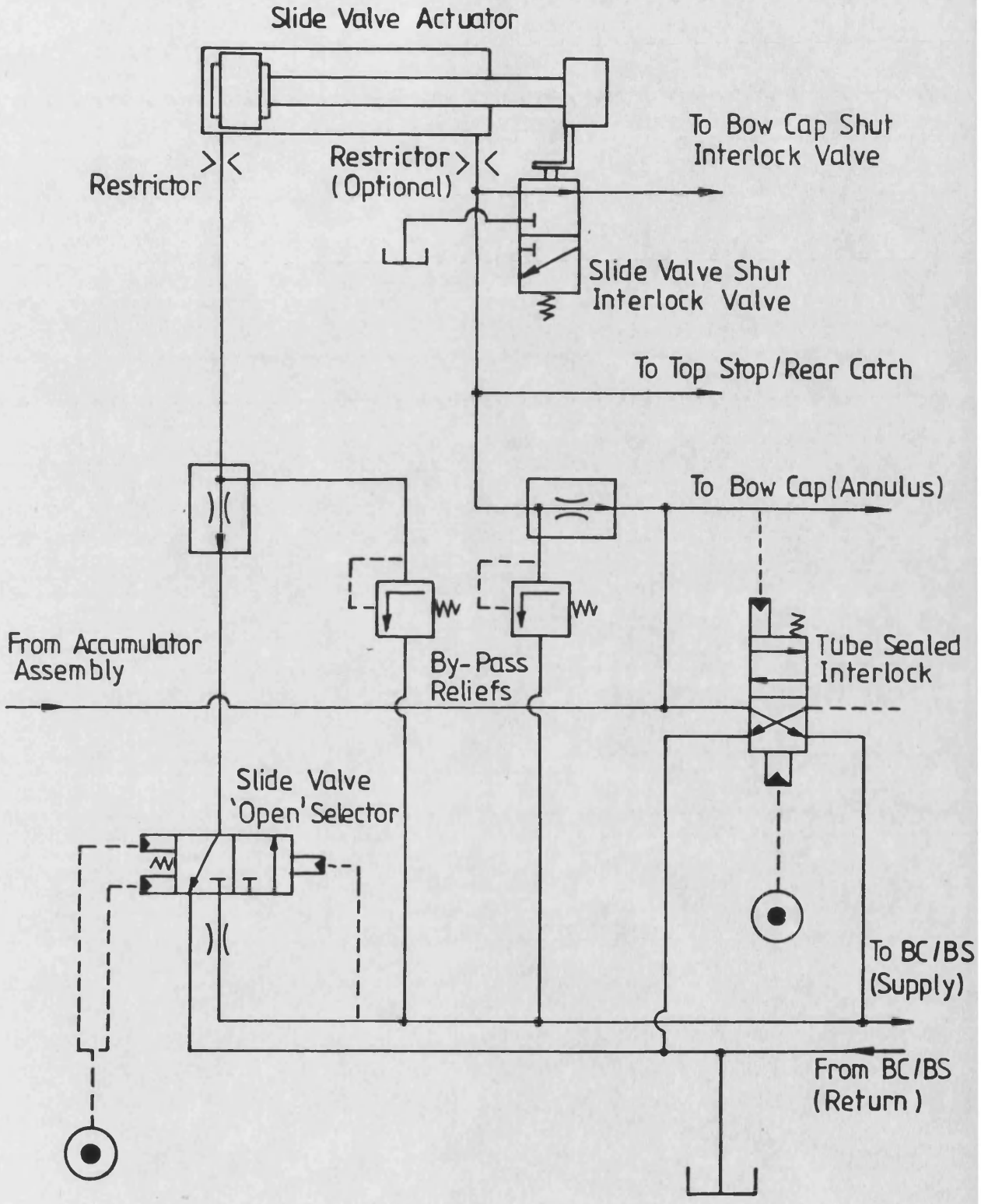


FIG. 5.3 The slide sub-circuit

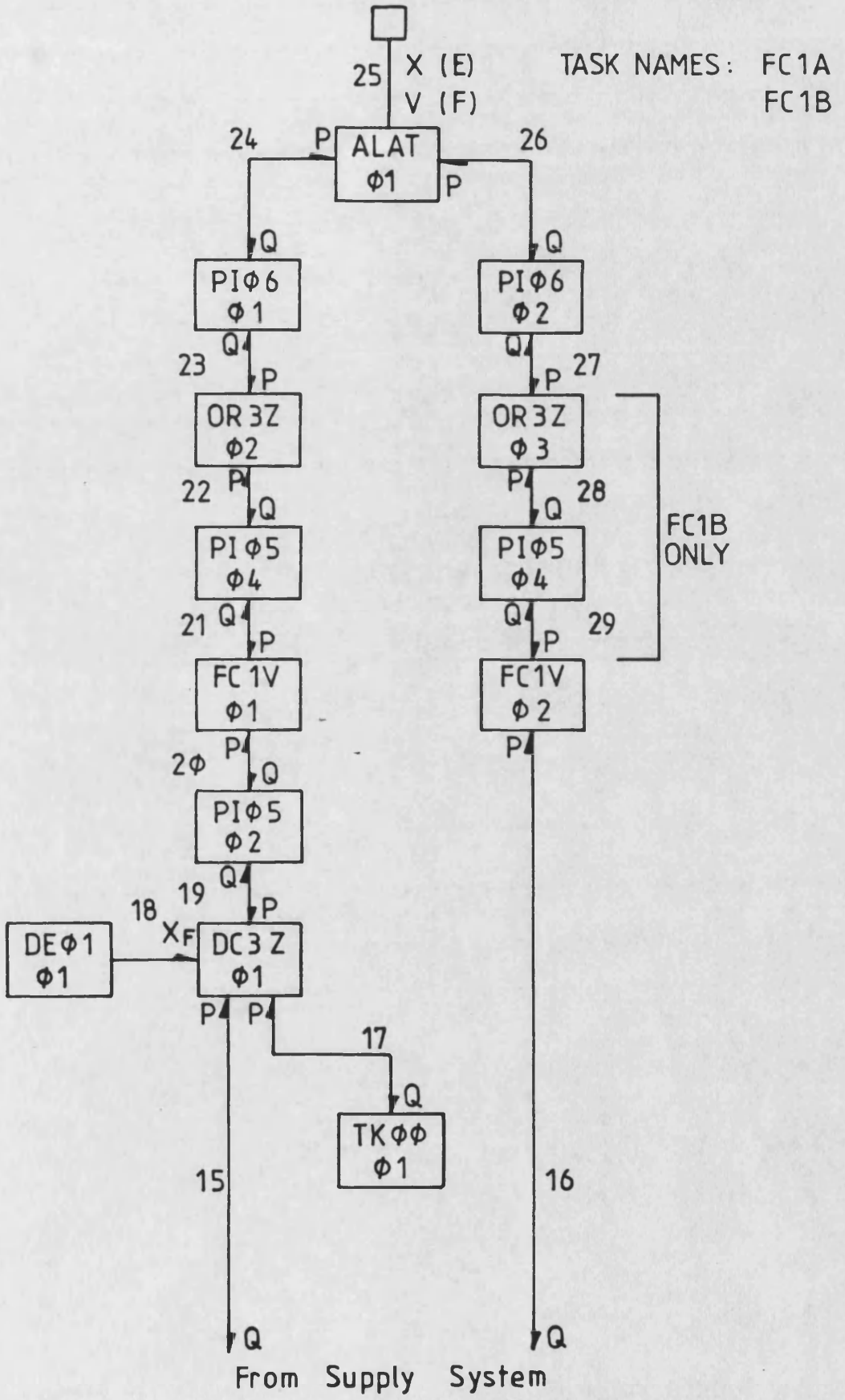


FIG. 5.4 The linking diagram for the slide sub-circuit

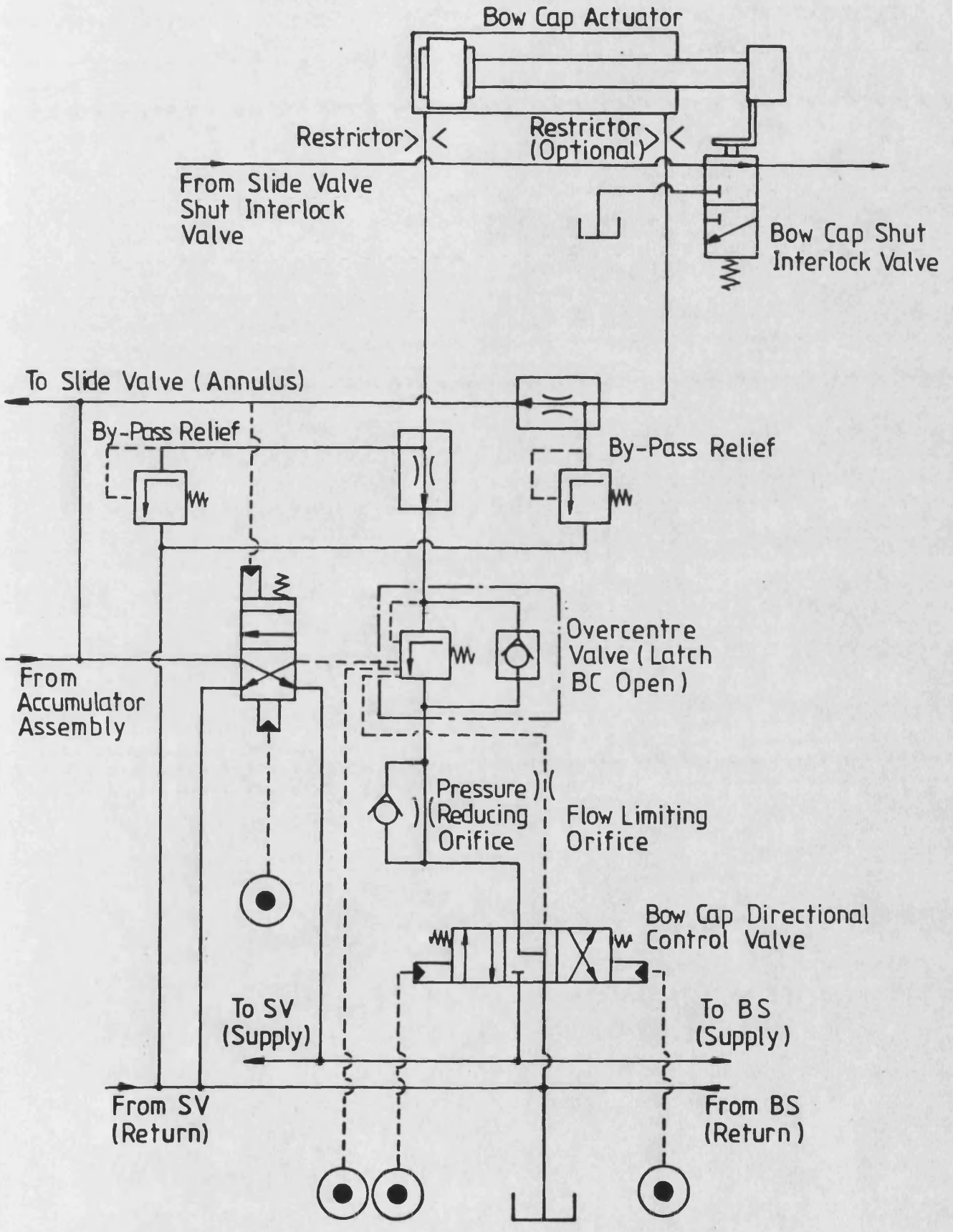


FIG. 5.5 The cap sub-circuit

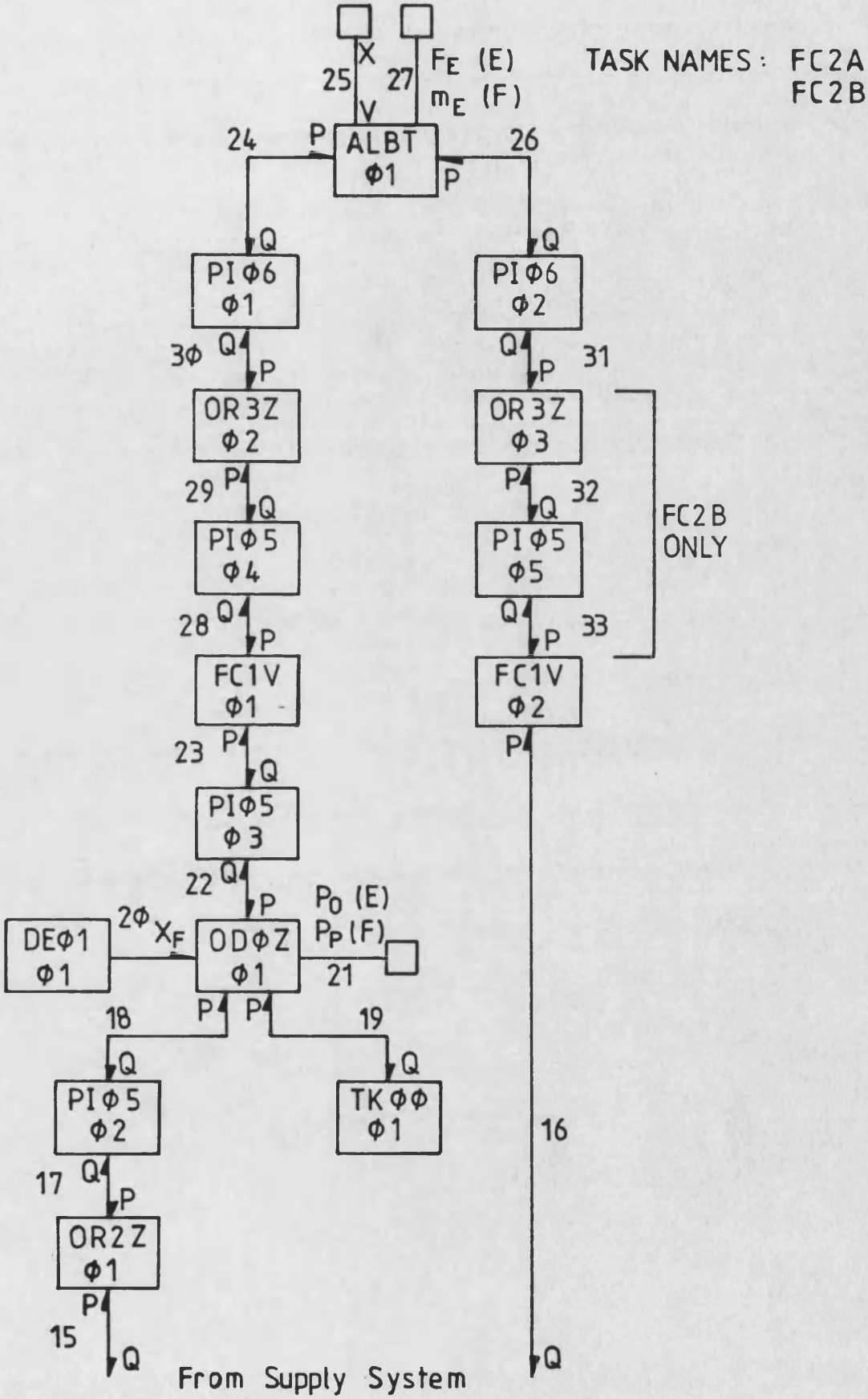


FIG. 5.6 The linking diagram for the cap sub-circuit

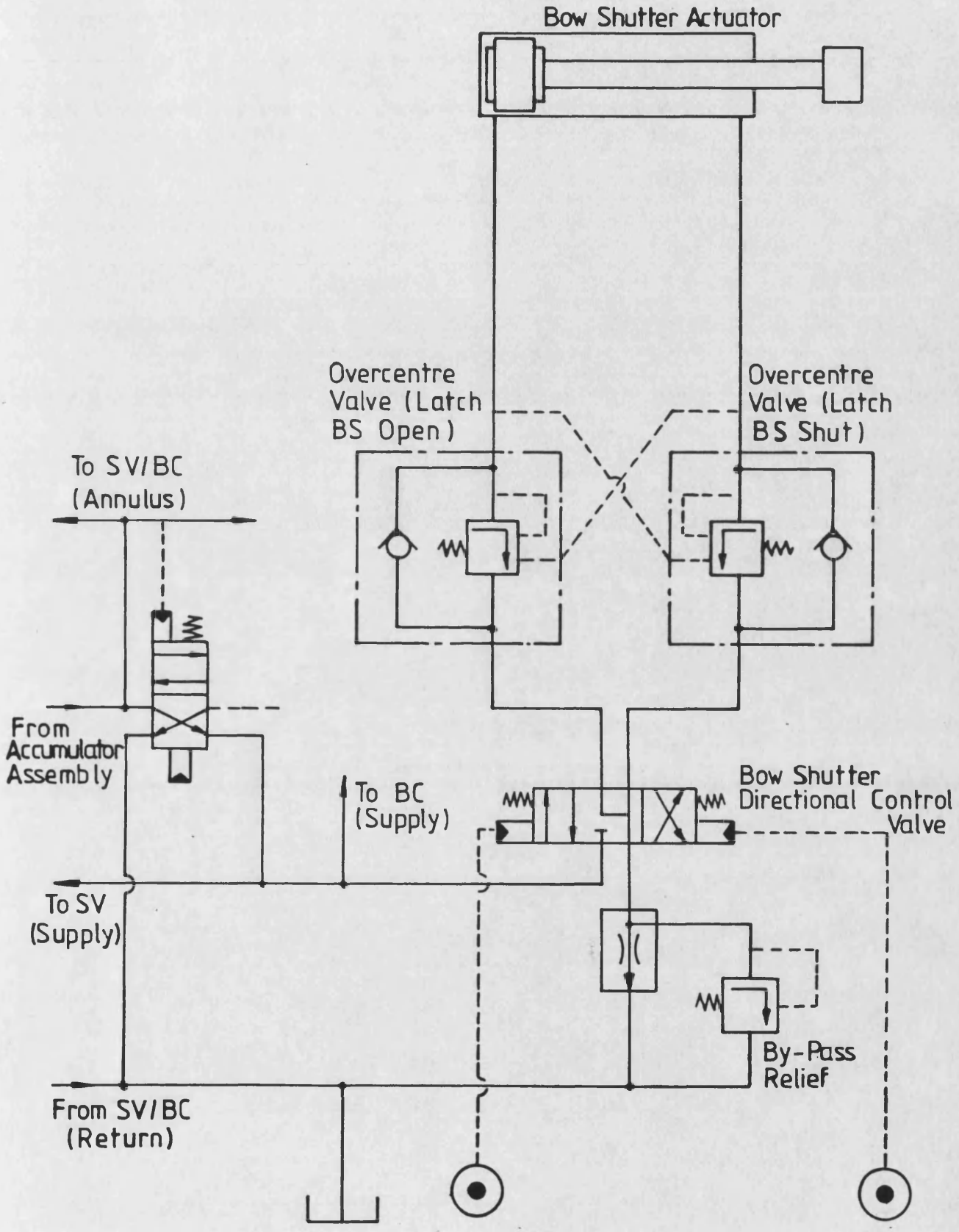


FIG. 5.7 The shutter sub-circuit

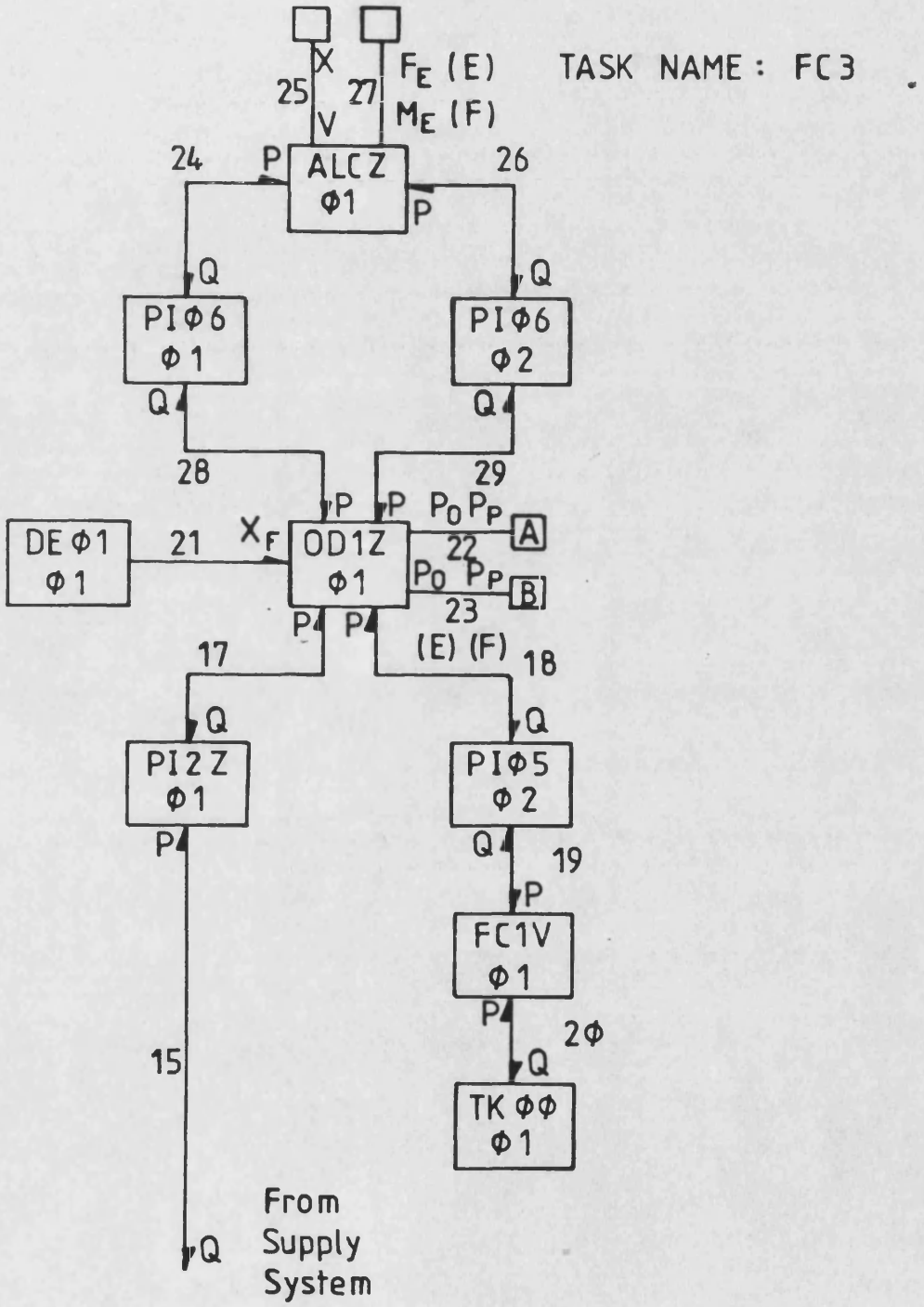


FIG. 5.8 The linking diagram for the shutter sub-circuit

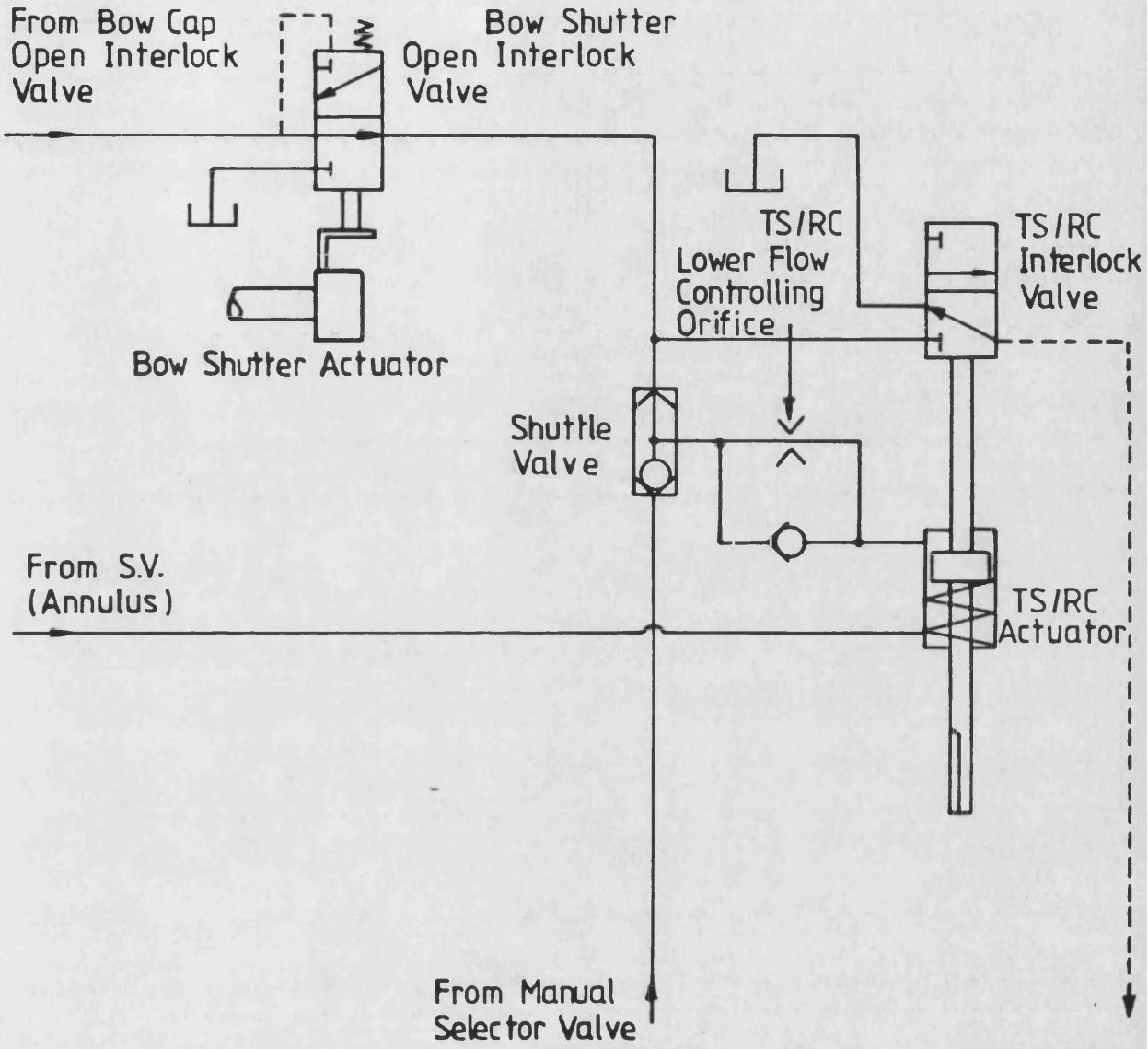


FIG. 5.9 The catch sub-circuit

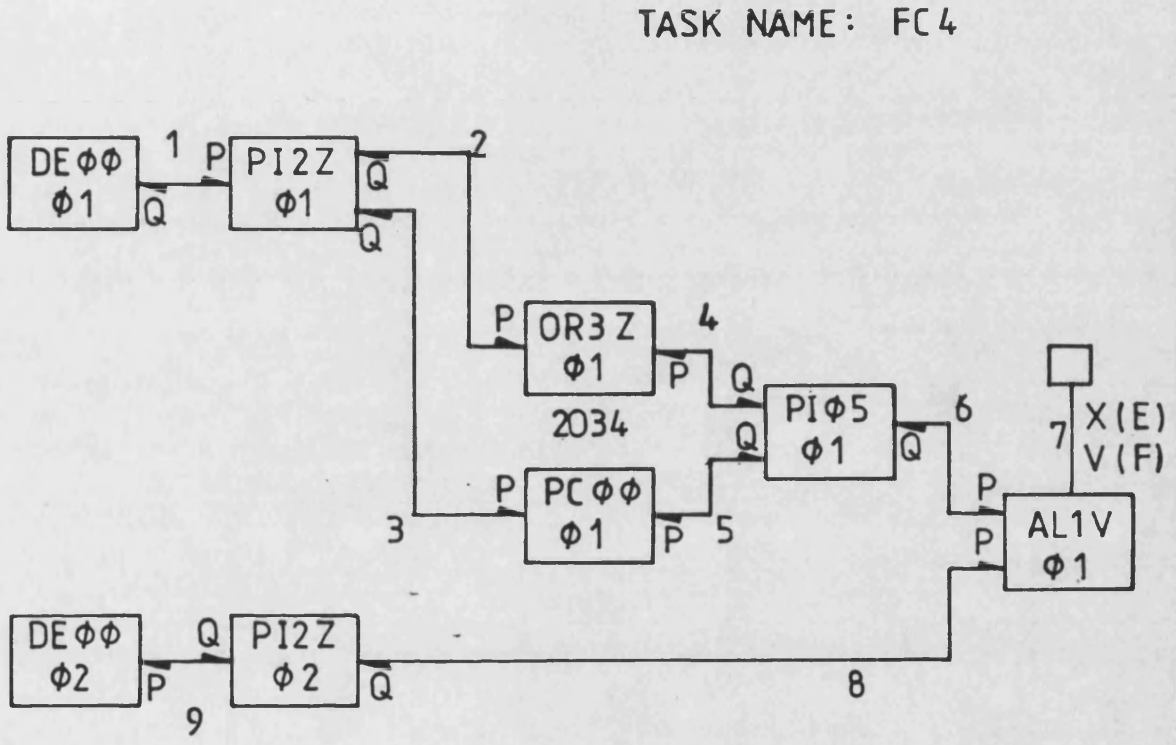


FIG. 5.10 The linking diagram for the catch sub-circuit

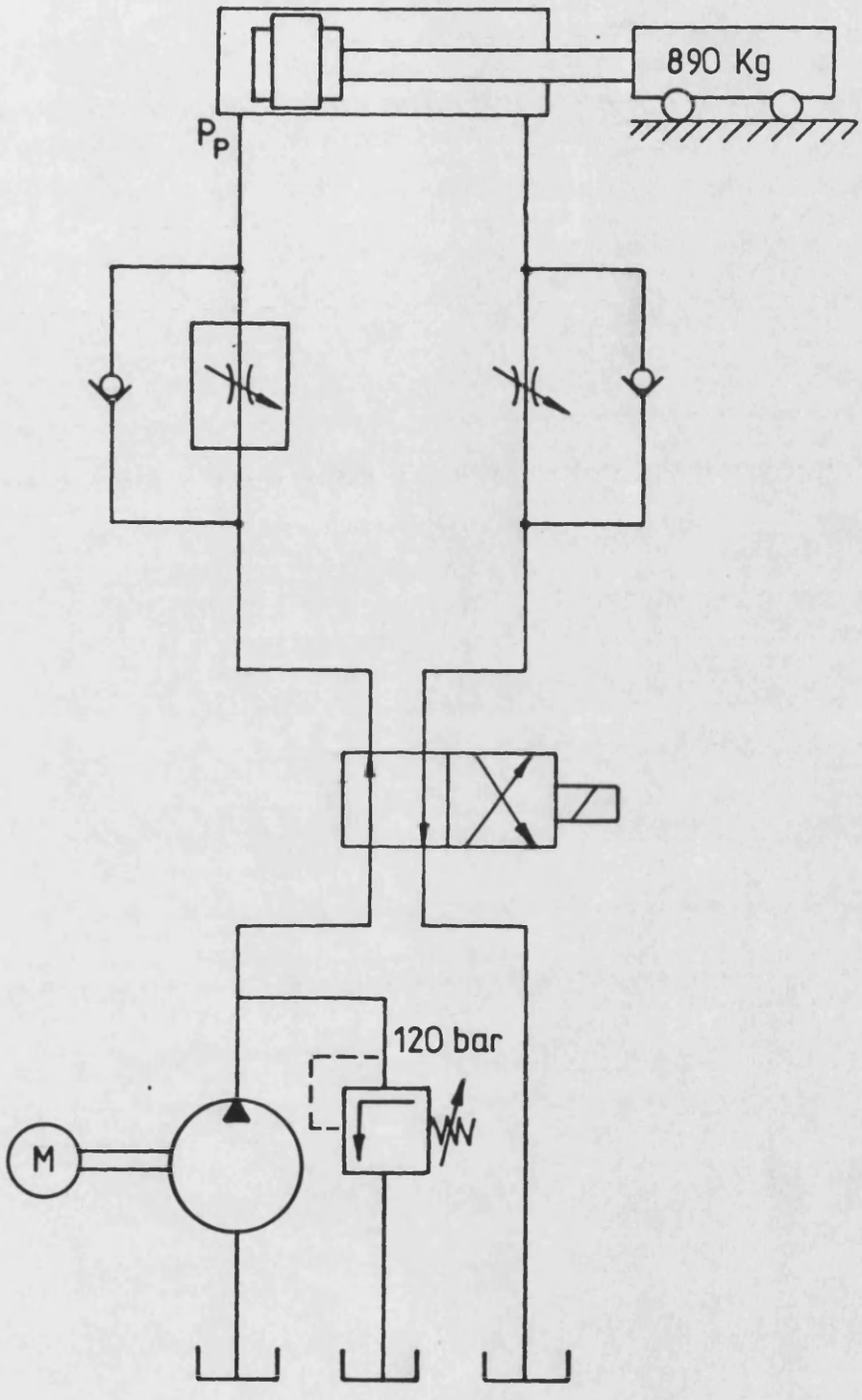


FIG. 5.11 The test rig

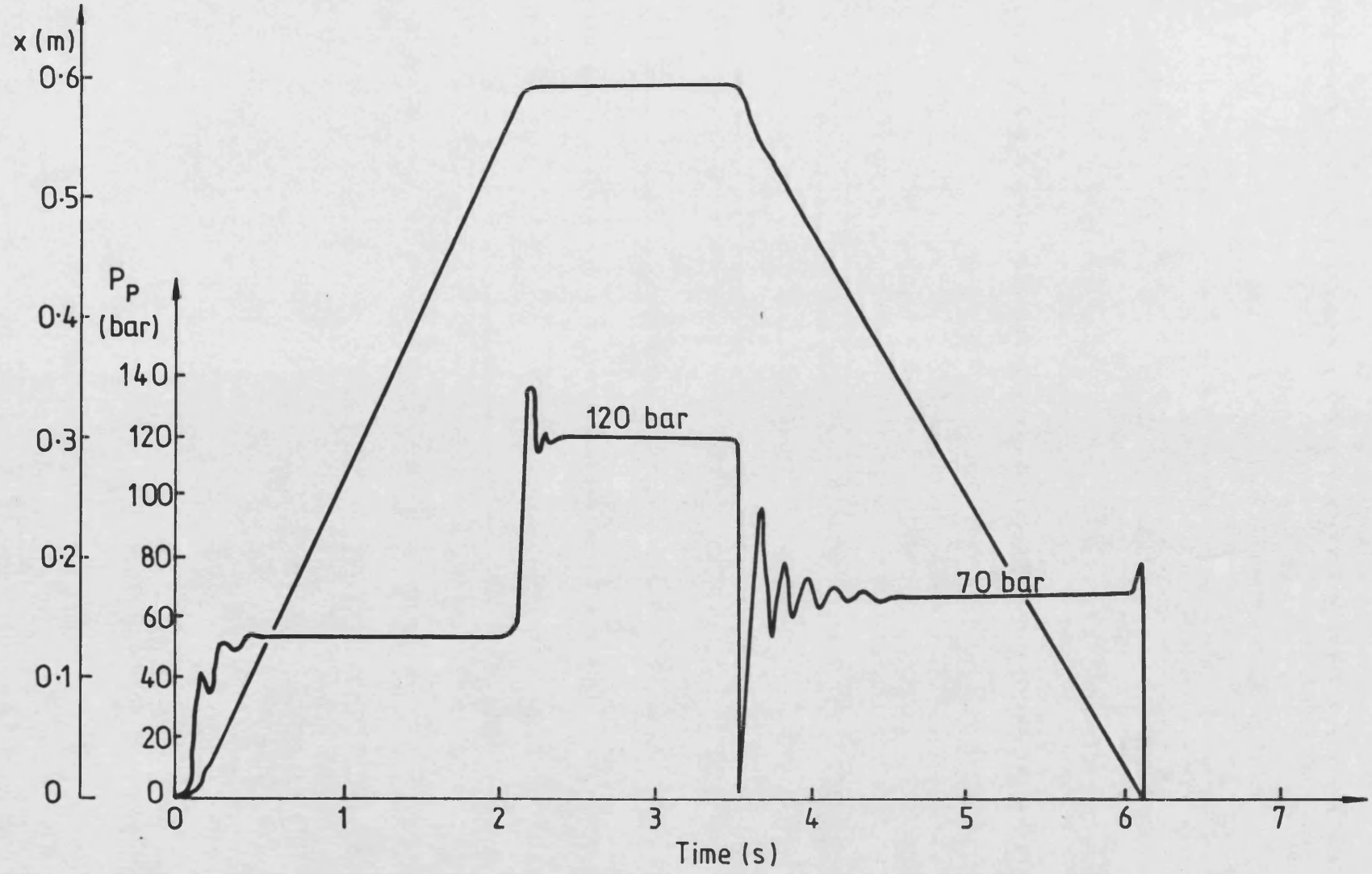
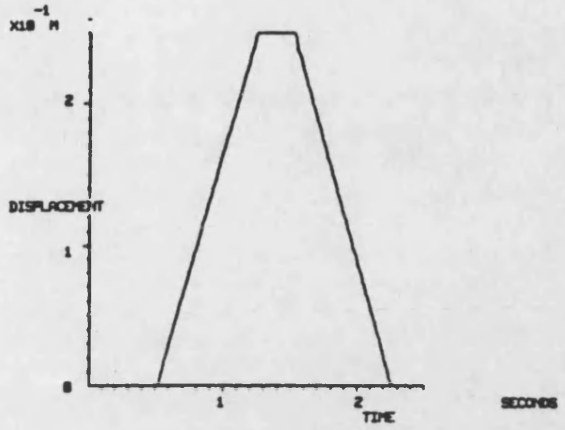
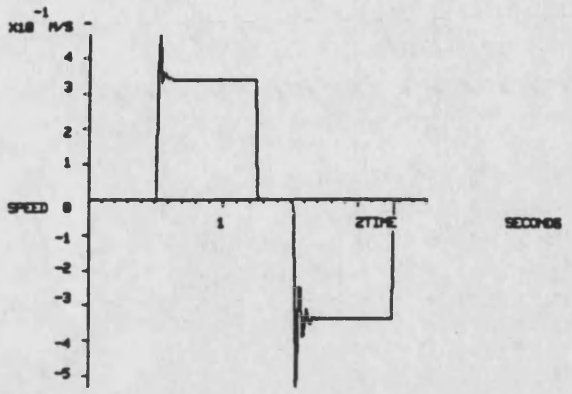


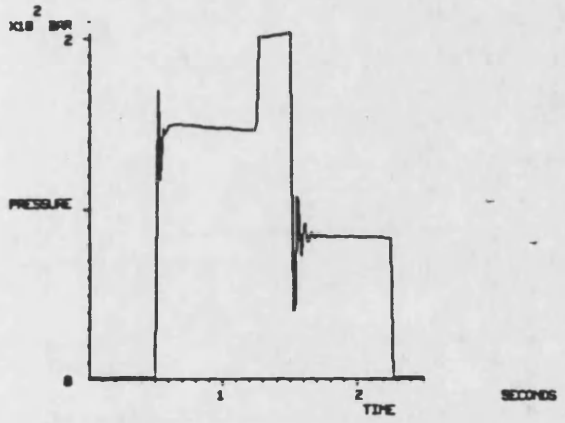
FIG. 5.12 The test results



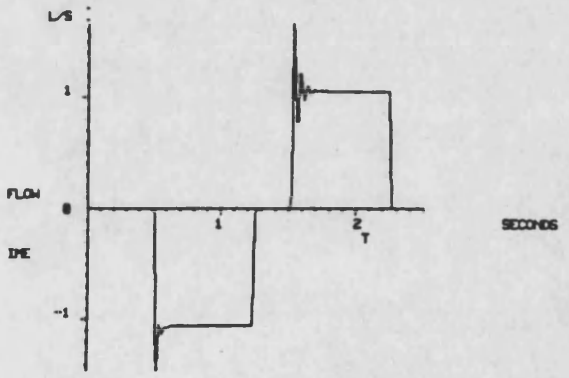
(a) Actuator displacement



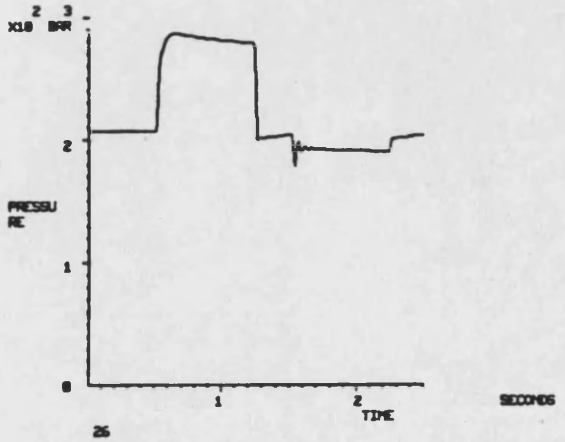
(b) Actuator velocity



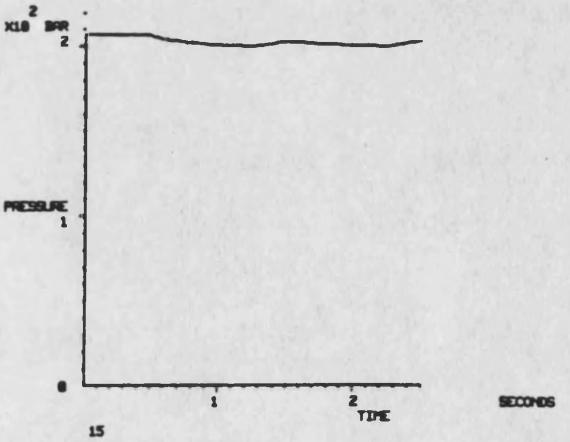
(c) Piston pressure



(d) Piston flowrate

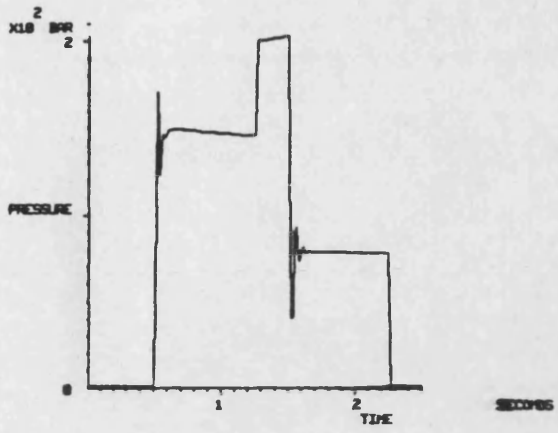


(e) Annulus pressure

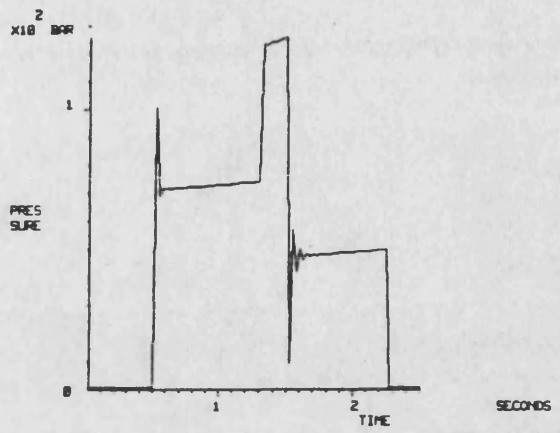


(f) Supply pressure

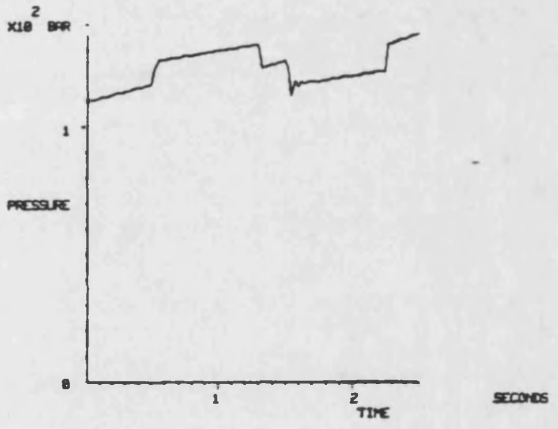
FIG. 5.13 Slide sub-circuit results - Standard run



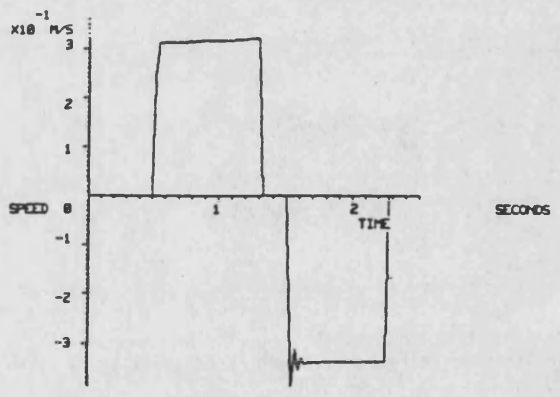
(a) Piston pressure



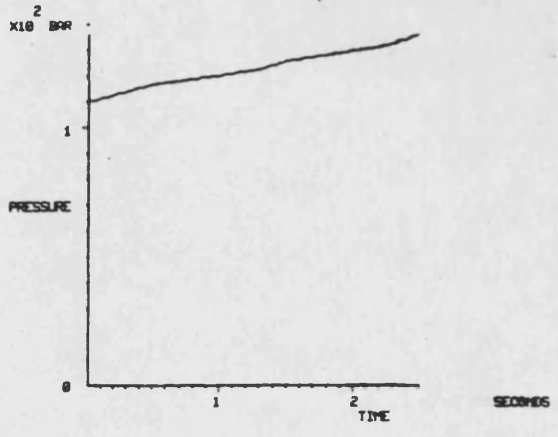
(b) Piston pressure



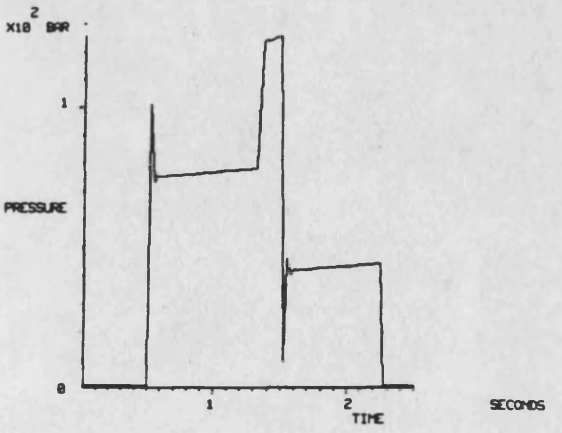
(c) Annulus pressure



(d) Actuator velocity

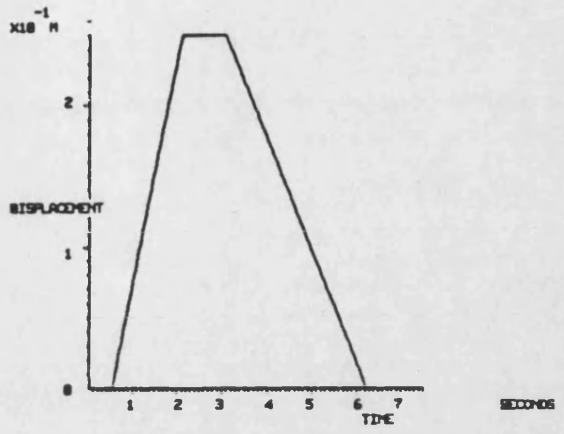


(e) Supply pressure

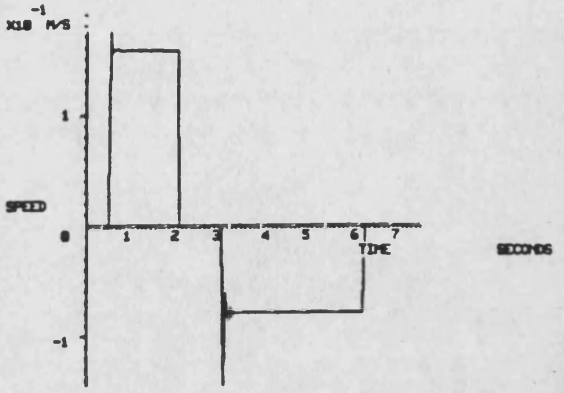


(f) Piston pressure

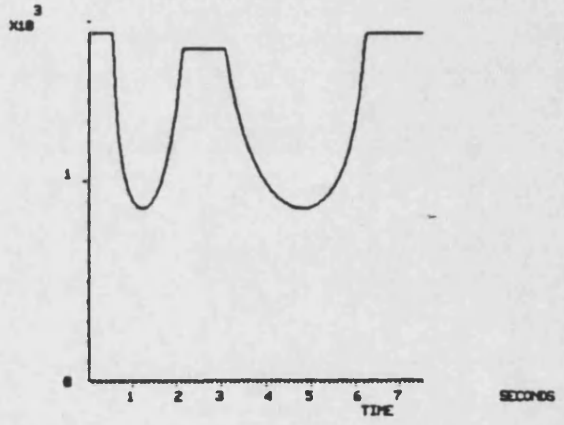
FIG. 5.14 Slide sub-circuit results - Further runs



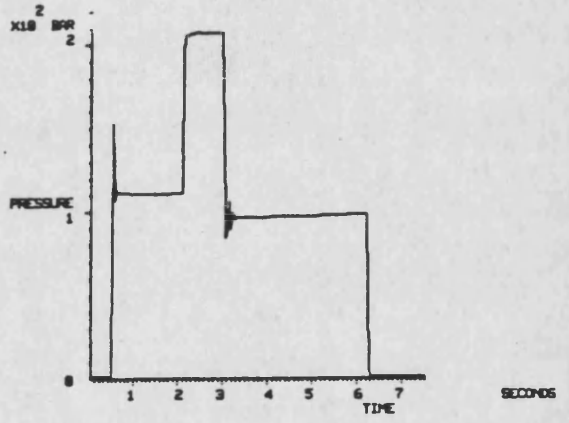
(a) Actuator displacement



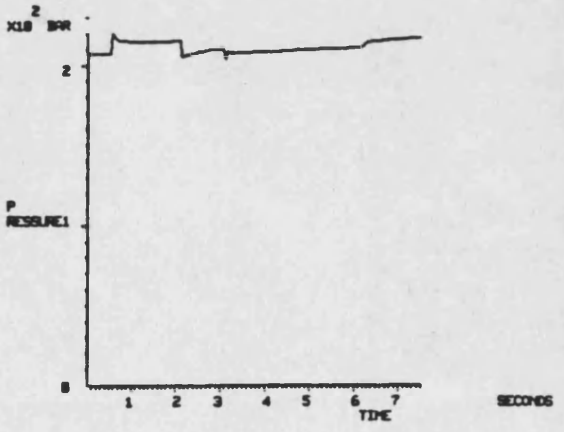
(b) Actuator velocity



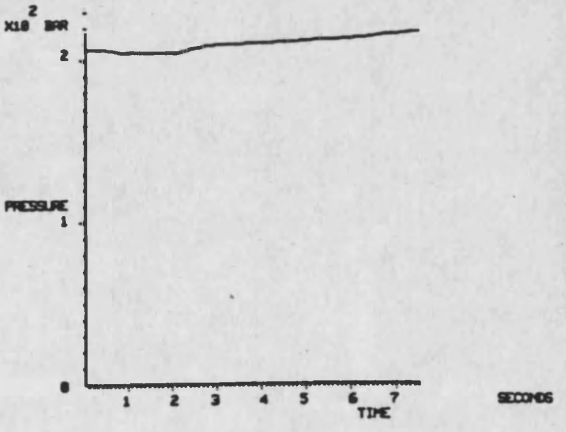
(c) Reflected mass in kg



(d) Piston pressure

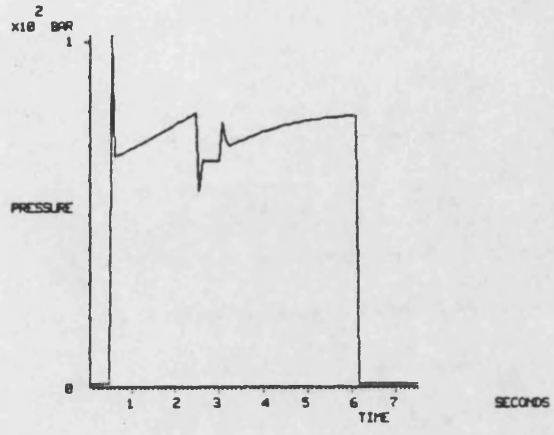


(e) Annulus pressure

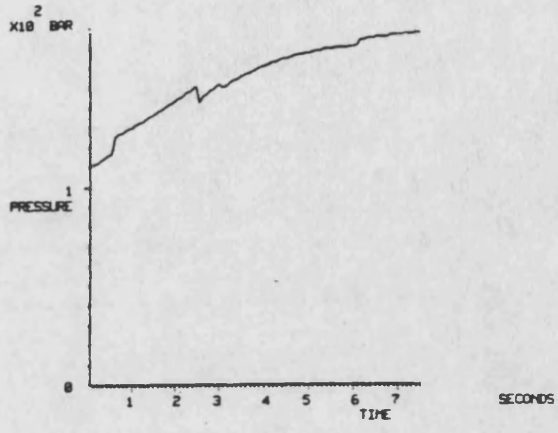


(f) Supply pressure

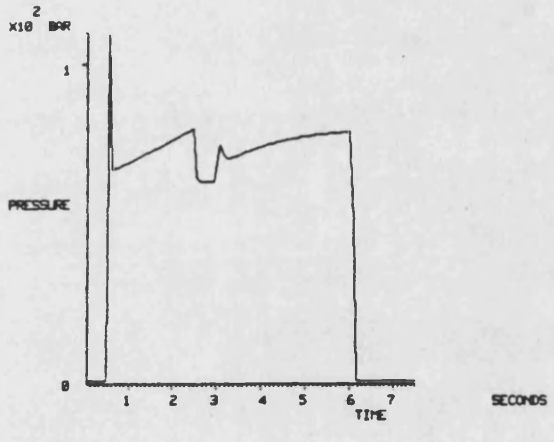
FIG. 5.15 Cap sub-circuit results - Standard run



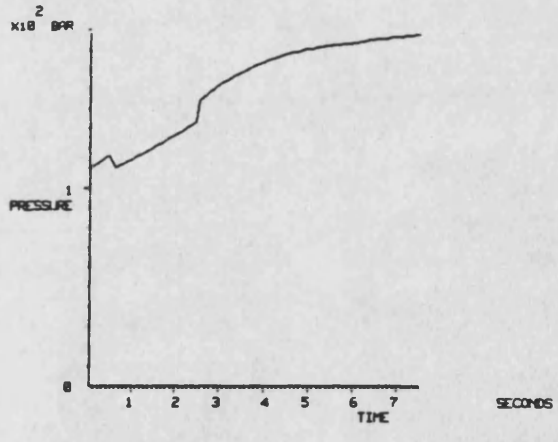
(a) Piston pressure



(b) Annulus pressure

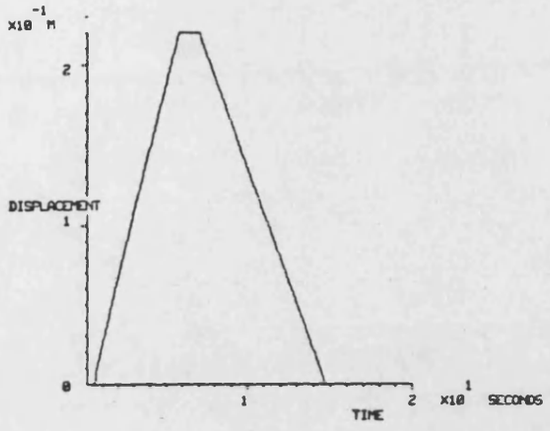


(c) Piston pressure

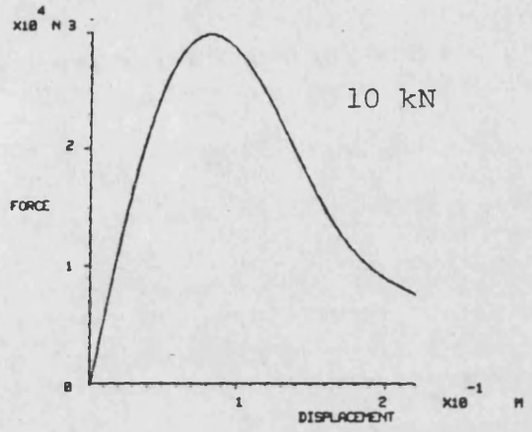


(d) Annulus pressure

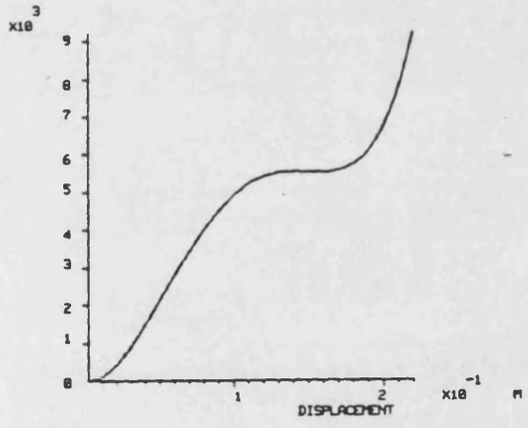
FIG. 5.16 Cap sub-circuit results - Further runs



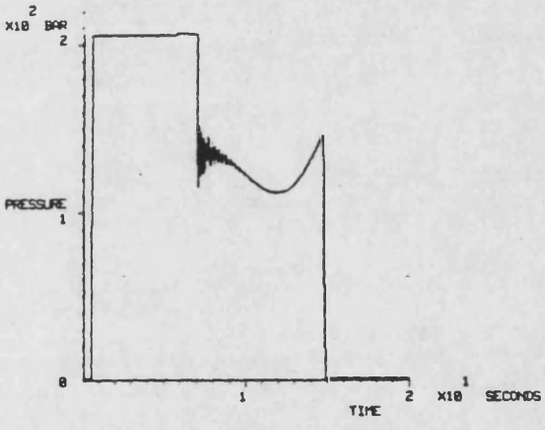
(a) Actuator displacement



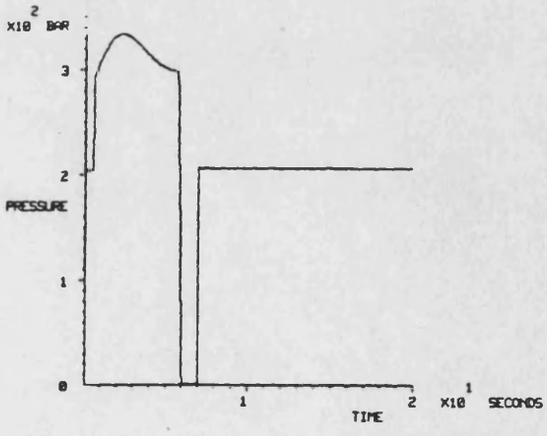
(b) Applied force (vs. x)



(c) Reflected mass in kg

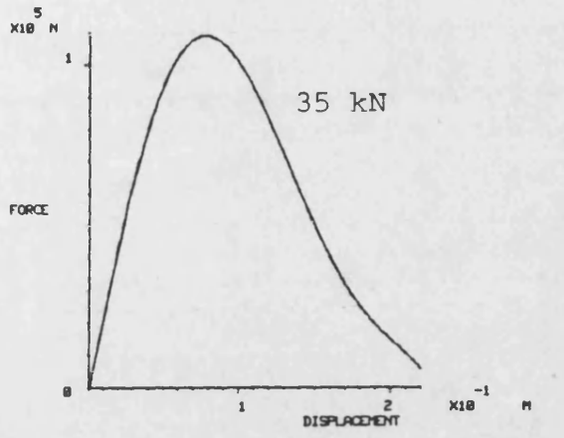


(d) Piston pressure

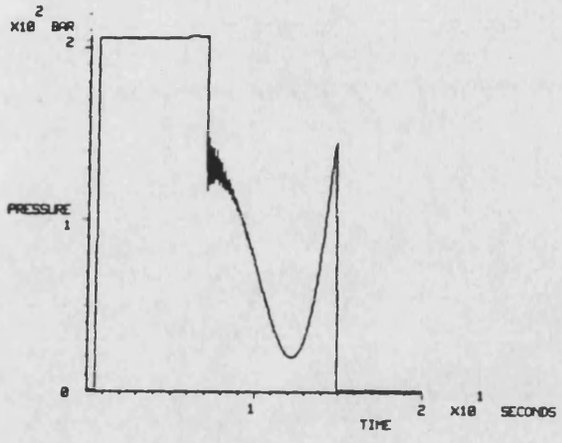


(e) Annulus pressure

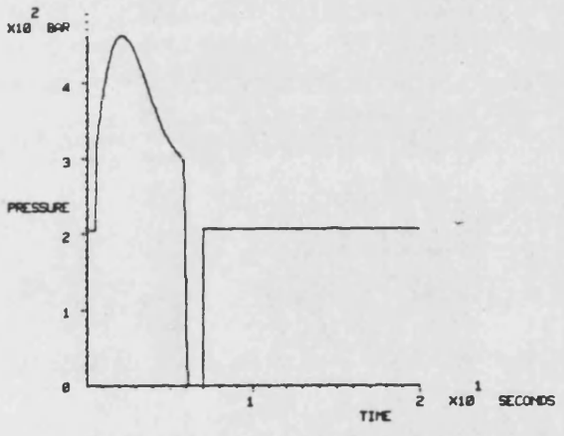
FIG. 5.17 Shutter sub-circuit results - Run 1



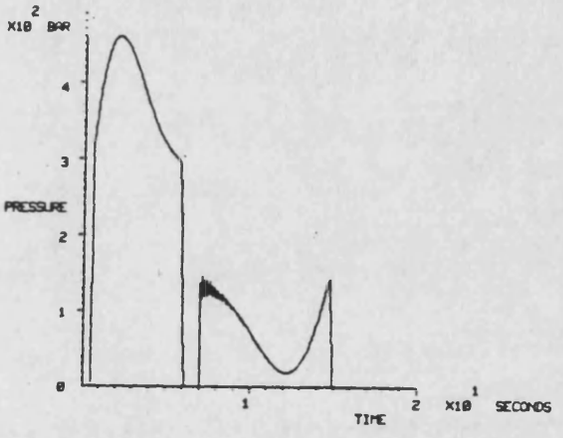
(a) Applied force (vs. x)



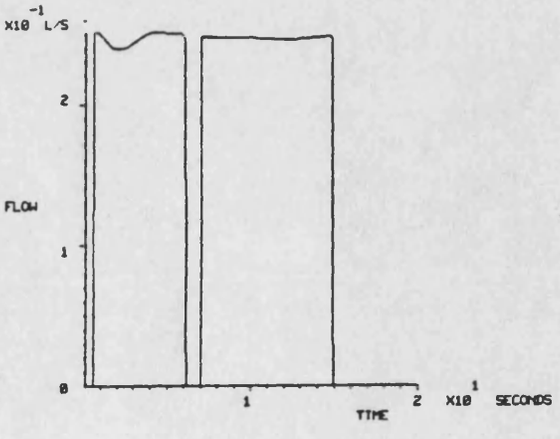
(b) Piston pressure



(c) Annulus pressure

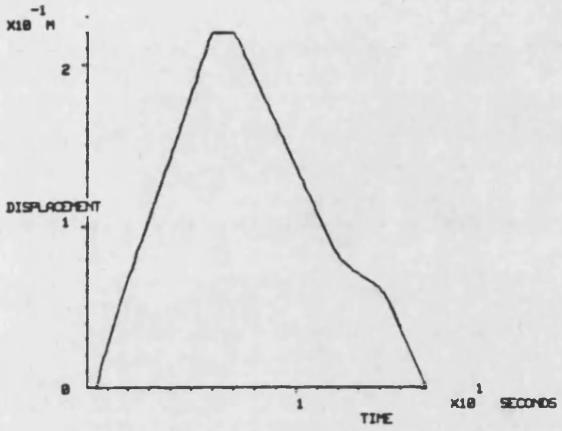


(d) Pressure across F.C.V.

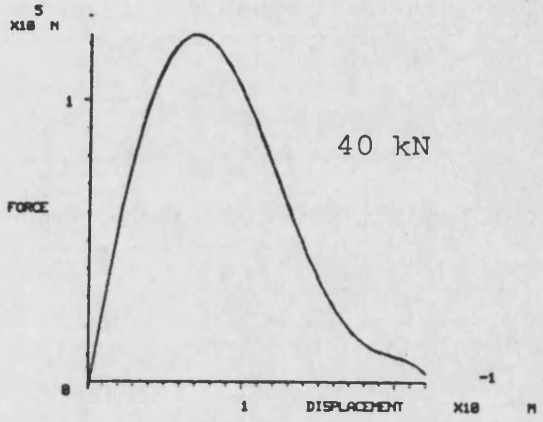


(e) Flowrate through F.C.V.

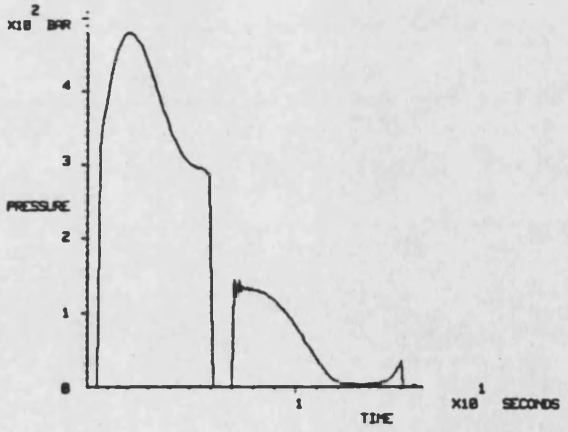
FIG. 5.18 Shutter sub-circuit results - Run 2



(a) Actuator displacement

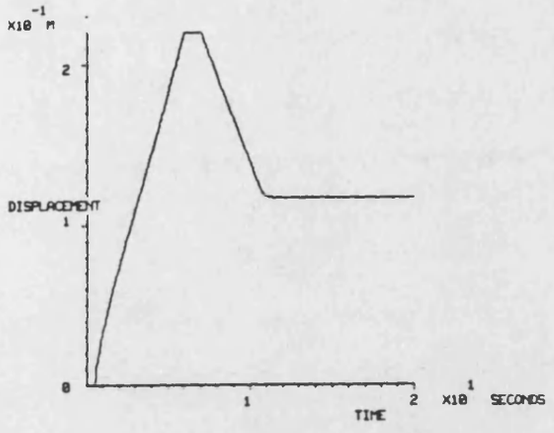


(b) Applied force (vs. x)

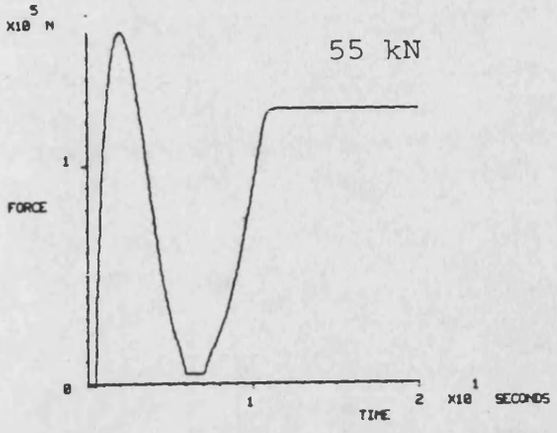


(c) Pressure across F.C.V.

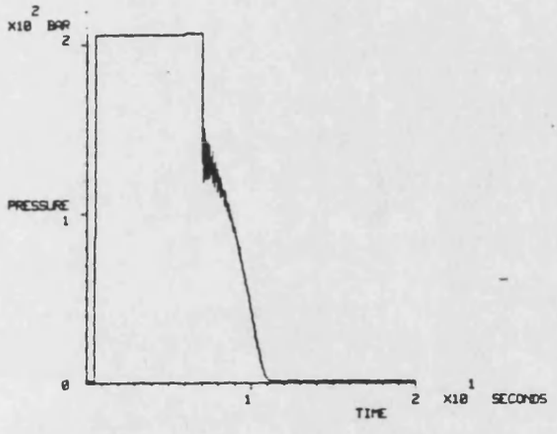
FIG. 5.19 Shutter sub-circuit results - Run 3



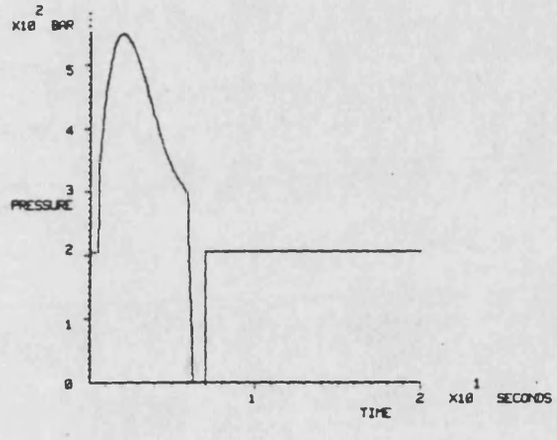
(a) Actuator displacement



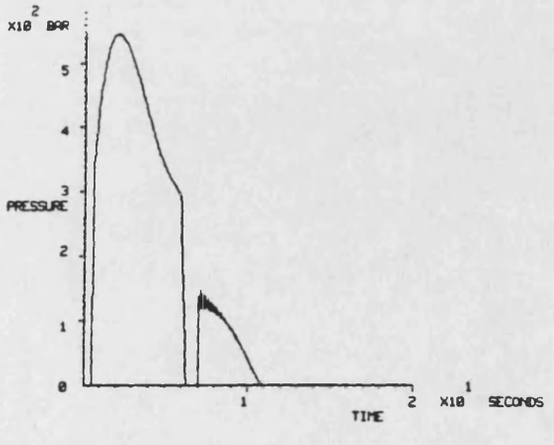
(b) Applied force



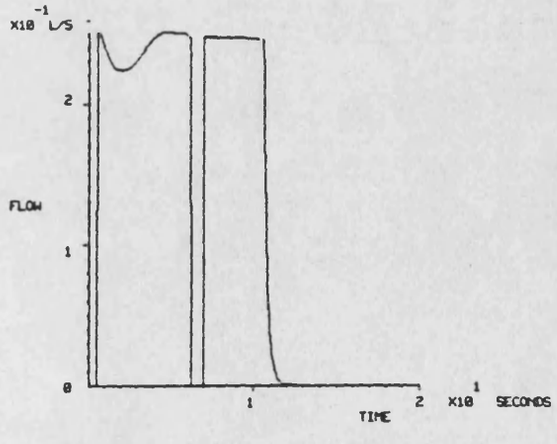
(c) Piston pressure



(d) Annulus pressure

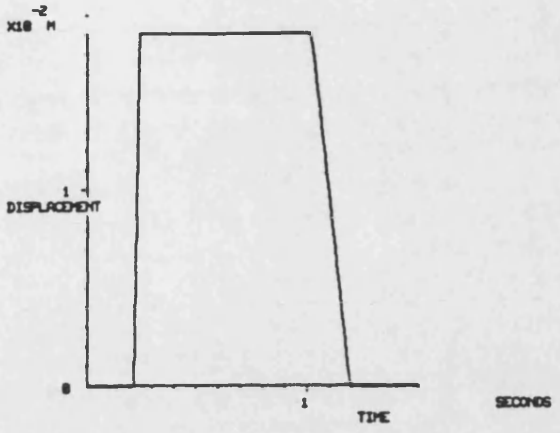


(e) Pressure across F.C.V.

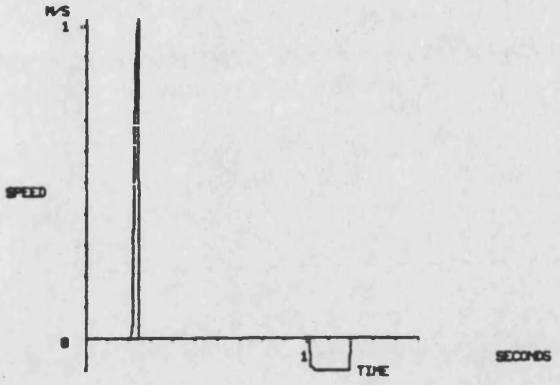


(f) Flowrate through F.C.V.

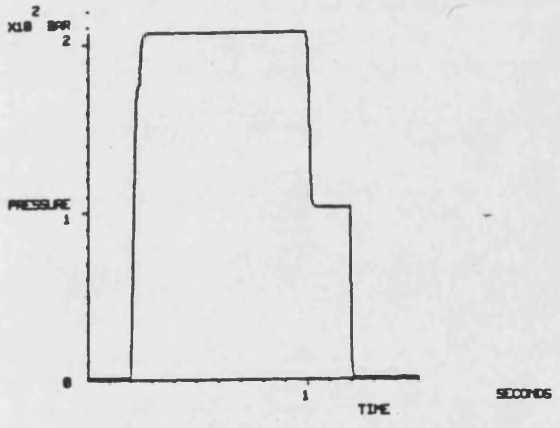
FIG. 5.20 Shutter sub-circuit - Run 4



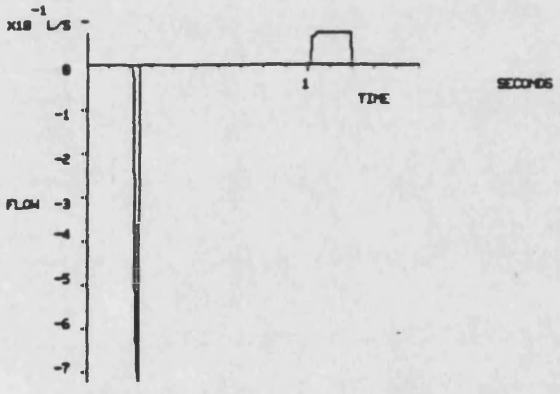
(a) Actuator displacement



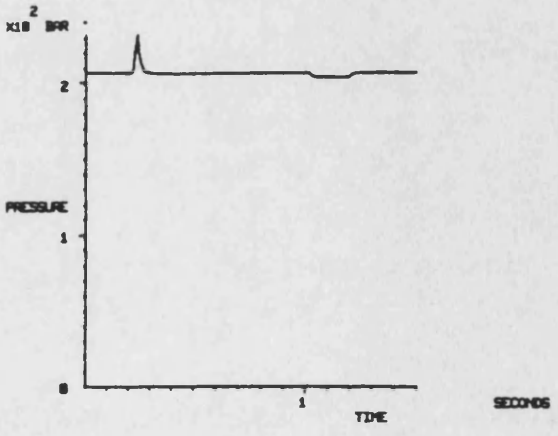
(b) Actuator velocity



(c) Piston pressure

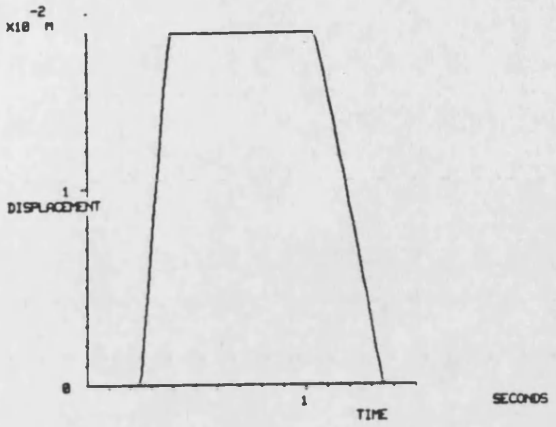


(d) Piston flowrate

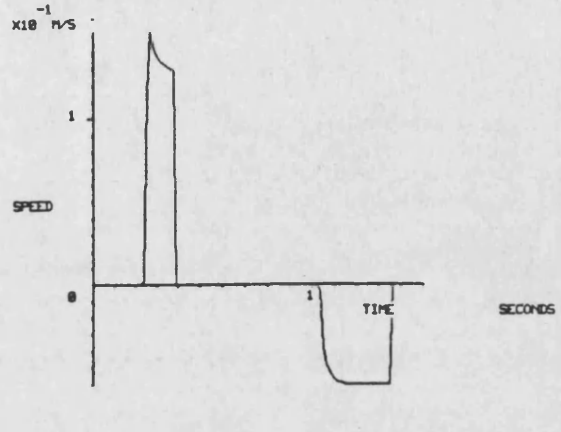


(e) Supply pressure

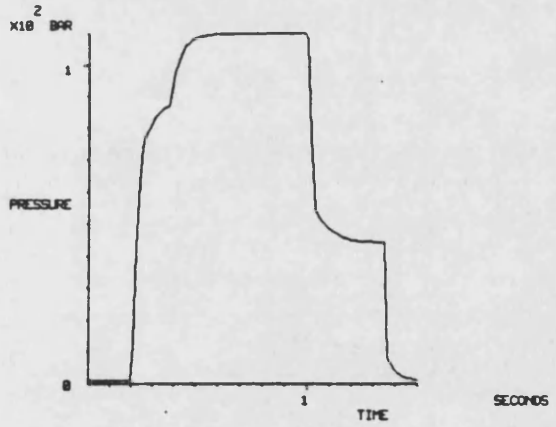
FIG. 5.21 Catch sub-circuit results - Standard run



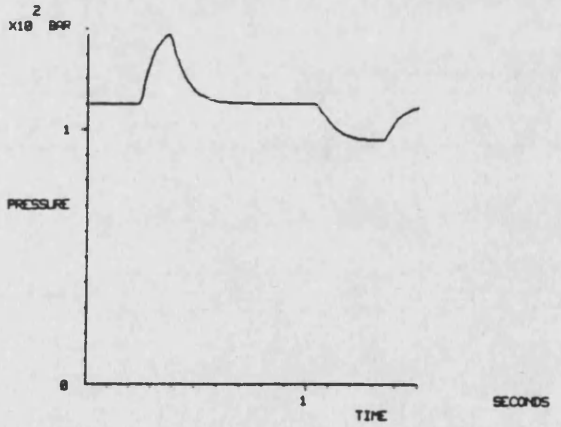
(a) Actuator displacement



(b) Actuator velocity



(c) Piston pressure



(d) Supply pressure

FIG. 5.22 Catch sub-circuit results - Further run

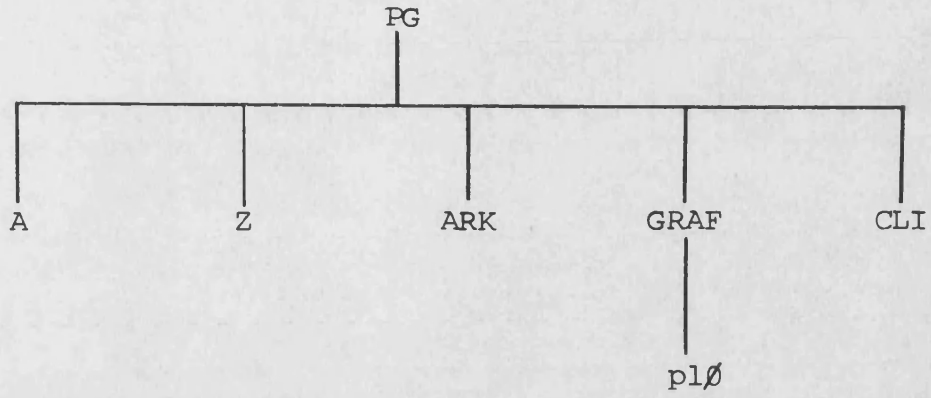


FIG. 6.1 Directory structure of directory PG

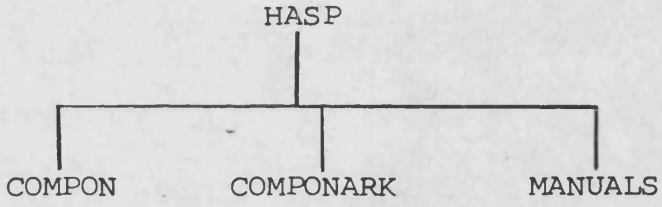


FIG. 6.2 Directory structure of directory HASP

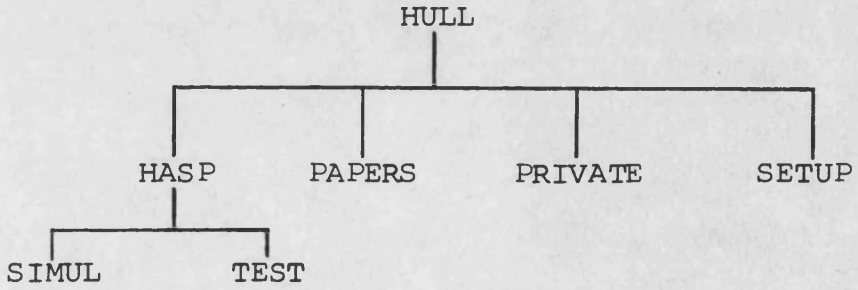


FIG. 6.3 A sample directory structure of a user directory

```
      SUBROUTINE PGHELP

C ***************************************************************************
C ***** SUBROUTINE TO SUPPLY USER WITH INFORMATION ABOUT ANY COMPONENT MODEL
C ***************************************************************************
C
C                                                  SCHOOL OF ENGINEERING
C                                                  UNIVERSITY OF BATH
C                                                  CLAVERTON DOWN
C                                                  BATH
C
C     DEVELOPED:      S. R. HULL                                  02-03-84
C
C ***************************************************************************
C
C     ARRAYS:
C         ELIN   (8)  EXTERNAL LINK INPUTS  (E OR F)                    CH*1
C         ELOUT  (8)  EXTERNAL LINK OUTPUTS (E OR F)                    CH*1
C         INPUT2 (2)  INPUT FROM DATA FILE - 4 CHARACTERS IN 2 ELEMENTS CH*2
C         NAME2  (2)  NAME REQUIRED - 4 CHARACTERS IN 2 ELEMENTS        CH*2
C
C     VARIABLES:
C         DUMMY       DUMMY CHARACTER VARIABLE - USED FOR BREAKS IN     CH*1
C                     OUTPUT OR TO SKIP UNWANTED LINES IN DATA FILES
C         I           LOOP COUNTER                                       I*4
C         ICOUNT      COUNTER TO KEEP TRACK OF NUMBER OF MODELS LISTED   I*4
C         INFO        LINES OF INFORMATION FROM [HASP.COMPON]INFORM.DAT CH*80
C         INPUT4      INPUT FROM DATA FILE                              CH*4
C         IPAGE       COUNTER TO KEEP TRACK OF THE NUMBER OF PAGES OF    I*4
C                     MODELS LISTED
C         LAST        FINAL ENTRY IN INFORM.DAT AND COMPON.DAT          CH*4
C         MAXEL       MAXIMUM NUMBER OF EXTERNAL LINKS                   I*4
C         MINEL       MINIMUM NUMBER OF EXTERNAL LINKS                   I*4
C         MODE        USER MODE                                          I*4
C         NAME4       NAME REQUIRED                                     CH*4
C         NIL         NUMBER OF INTERNAL LINKS                           I*4
C         NLINES      NUMBER OF LINES OF DATA TO BE SKIPPED              I*4
C         NSIG        NUMBER OF SIGNALS                                  I*4
C         NSV         NUMBER OF STATE VARIABLES                          I*4
C         STRING      TITLE OF COMPONENT HELD IN COMPON.DAT             CH*74
C
C ***************************************************************************
                                                    !
      CHARACTER*1  DUMMY,ELIN(8),ELOUT(8)           ! SET UP CHARACTER
      CHARACTER*2  NAME2(2),INPUT2(2)               ! VARIABLES AND ARRAYS.
      CHARACTER*4  NAME4,INPUT4,LAST                !
      CHARACTER*74 STRING                           ! EQUIVALENCE CERTAIN
      CHARACTER*80 INFO                             ! CHARACTER VARIABLES
      EQUIVALENCE  (NAME2,NAME4)                    ! FOR EASE OF REFER-
      EQUIVALENCE  (INPUT2,INPUT4)                  ! ENCE AND COMPARISON.
      DATA         LAST/'LAST'/                     !
                                                    !
C ***************************************************************************
C ***** OPEN THE INFORMATION DATA FILE
C ***************************************************************************
                                                    !
      OPEN (UNIT=6, FILE='[HASP.COMPON]INFORM.DAT',  ! OPEN THE COMPONENT
     *       STATUS='OLD', READONLY)                ! INFORMATION DATA FILE
                                                    !
C ***************************************************************************
C ***** USER OPTION
C ***************************************************************************
```

TABLE 3.1 Listing of Program Generator subroutine PGHELP

```
                                                                      !
  10    WRITE(5,20)                                                   !
  20    FORMAT(//,                                                    !
     *  ' TYPE 1 TO LIST CLASSES OF MODELS',/,                        !
     *  ' TYPE 2 TO LIST INFORMATION ABOUT A PARTICULAR MODEL',/,!
     *  ' PRESS RETURN TO PROCEED WITH GENERATION')                   !
                                                                      ! CHECK USER MODE AND
        READ(5,510,ERR=10)MODE                                        ! CHANGE 0 TO 3 IN
        IF(MODE.NE.1.AND.MODE.NE.2)MODE=3                             ! ORDER TO USE LOGICAL
        GOTO(100,300,500),MODE                                        ! GO TO.
                                                                      !
C ******************************************************************************
C ***** LIST CLASSES OF MODELS IN GROUPS OF UP TO 19
C ******************************************************************************
                                                                      !
 100    WRITE(5,110)                                                  ! EXTRA ENTRIES MAY BE
 110    FORMAT(///,                                                   ! INCLUDED BUT ONLY UP
     *  ' ''AC'' REPRESENTS ACCUMULATORS',/,                          ! TO 19 CONTINUATION
     *  ' ''AL''            ACTUATOR-LOADS',/,                        ! LINES ALLOWED,
     *  ' ''AM''            AMPLIFIERS',/,                            ! (DEFAULT).
     *  ' ''DC''            DIRECTIONAL CONTROL VALVES',/,!
     *  ' ''DE''            DUTY CYCLES - EFFORT SOURCES',/,!
     *  ' ''DF''            DUTY CYCLES - FLOW SOURCES',/,!
     *  ' ''FC''            FLOW CONTROL VALVES',/,                   !
     *  ' ''HE''            HEAT EXCHANGERS',/,                       !
     *  ' ''HU''            PUMP/MOTOR UNITS',/,                      ! ALLOW A BREAK IN THE
     *  ' ''LR''            ROTARY LOADS')                            ! LISTING TO STOP
C       WRITE(5,120)                                                  ! OPTIONS SCROLLING OFF
 120    FORMAT('0PRESS RETURN TO CONTINUE LISTING')                   ! SCREEN.
C       READ(5,520)DUMMY                                              !
        WRITE(5,130)                                                  ! NOT YET REQUIRED
 130    FORMAT(                                                       ! THEREFORE COMMENTISED
     *  ' ''MO''            HYDRAULIC MOTORS',/,                      !
     *  ' ''OR''            ORIFICE RESTRICTORS',/,                   ! FURTHER AMENDMENT:
     *  ' ''PC''            PRESSURE CONTROL VALVES',/,               ! PUT CLASS OPTIONS IN
     *  ' ''PI''            PIPES',/,                                 ! AN EXTERNAL FILE TO
     *  ' ''PM''            PRIME MOVERS',/,                          ! REMOVE NEED TO EDIT
     *  ' ''PU''            HYDRAULIC PUMPS',/,                       ! PROGRAM GENERATOR.
     *  ' ''SC''            SWASH CONTROLLERS',/,                     !
     *  ' ''TK''            HYDRAULIC TANKS',/,                       !
     *  ' ''TS''            TEST SOURCE MODELS')                      !
                                                                      !
        WRITE(5,140)                                                  !
 140    FORMAT('0TYPE REQUIRED CODE')                                 !
        READ(5,530)NAME2(1)                                           !
        WRITE(5,150)                                                  !
 150    FORMAT(///)                                                   !
                                                                      !
C ******************************************************************************
C ***** LIST ALL MODELS IN [HASP.COMPON]COMPON.DAT BEGINNING WITH NAME2(1)
C ******************************************************************************
                                                                      !
        ICOUNT=0                                                      ! SET COUNTERS TO ZERO.
        IPAGE=0                                                       !
                                                                      ! READ ENTRY IN
 200    READ(7,540)INPUT2,NLINES,STRING                               ! COMPON.DAT.
                                                                      !
        IF(INPUT2(1).EQ.NAME2(1))THEN                                 ! IF REQ'D COMPONENT,
              ICOUNT=ICOUNT+1                                         ! UPDATE COUNTER.
              IF(ICOUNT.EQ.21)THEN                                    ! IF 21ST ENTRY, ALLOW
                    WRITE(5,120)                                      ! A BREAK IN LISTING,
                    READ(5,520)DUMMY                                  ! RESET COUNTER ICOUNT,
                    ICOUNT=ICOUNT-20                                  ! AND UPDATE PAGE
```

TABLE 3.1 Listing of Program Generator subroutine PGHELP (continued)

```
                IPAGE=IPAGE+1                                  ! COUNTER BY 1.
              END IF                                           !
              WRITE(5,210)INPUT4,STRING                        ! THEN WRITE MNEMONIC
210           FORMAT(1X,A4,2X,A74)                             ! AND DESCRIPTION.
              DO 220 I=1,NLINES                                ! SKIP REQ'D NUMBER OF
220           READ(7,520)DUMMY                                 ! LINES. GO BACK TO
              GOTO 200                                         ! READ NEXT ENTRY.
        ELSE                                                   !
              IF(INPUT4.EQ.LAST)THEN                           ! IF COMPONENT NOT
                    IF(ICOUNT.EQ.0.AND.IPAGE.EQ.0)THEN         ! REQ'D, CHECK IT'S
                          WRITE(5,230)NAME2(1)                 ! NOT LAST ENTRY. IF IT
230                       FORMAT(                              ! IS, AND NO MODELS
     *                    '0SORRY - NO MODELS IN  CLASS ''',A2,''' EXIST')!
                          REWIND 7                             ! FOUND, WRITE ERROR
                          GOTO 10                              ! MESSAGE AND GO BACK
                    END IF                                     ! TO INITIAL OPTION.
                    GOTO 250                                   ! OTHERWISE GO TO THIS
                                                               ! PARTICULAR COMPONENT
              ELSE                                             ! OPTION.
                    DO 240 I=1,NLINES                          ! IF COMPONENT NOT
240                 READ(7,520)DUMMY                           ! REQ'D AND NOT LAST,
                    GOTO 200                                   ! SKIP REQUIRED
              END IF                                           ! NUMBER OF LINES AND
        END IF                                                 ! READ NEXT ENTRY.
                                                               !
250     REWIND 7                                               !
        WRITE(5,260)NAME2(1)                                   !
260     FORMAT(                                                !
     *  '0COMPLETE COMPONENT NAME FOR HELP ON THAT COMPONENT',/,!
     *  ' PRESS RETURN FOR MAIN HELP OPTIONS',//,              ! COMPLETE CLASS NAME
     *  ' ',A2,$)                                              ! FOR HELP ON PARTIC-
        READ(5,530)NAME2(2)                                    ! ULAR COMPONENT.
        IF(NAME2(2).EQ.'  ')GOTO 10                            ! IF RETURN, GO TO
                                                               ! MAIN HELP OPTION.
                                                               !
C ***************************************************************************
C ***** LIST INFORMATION ABOUT A PARTICULAR MODEL CONTAINED IN INFORM.DAT
C ***************************************************************************
                                                               !
300     IF(MODE.EQ.2)THEN                                      ! IF MODE IS 2, FIND
310           WRITE(5,320)                                     ! OUT COMPONENT NAME.
320           FORMAT(///,                                      !
     *        ' TYPE 4 CHARACTER MNEMONIC REPRESENTING COMPONENT',/,!
     *        ' PRESS RETURN FOR MAIN HELP OPTIONS')           !
              READ(5,550)NAME4                                 ! IF RETURN, GO TO
              IF(NAME4.EQ.'    ')GOTO 10                       ! MAIN HELP OPTION.
        END IF                                                 !
330     READ(7,560)INPUT4,NLINES                               ! LOCATE ENTRY IN
        IF(INPUT4.EQ.NAME4)THEN                                ! COMPON.DAT AND READ
              READ(7,570)MINEL,MAXEL,NIL,NSIG,NSV              ! COMPONENT ATTRIBUTES.
              READ(7,580)ELIN                                  !
              READ(7,580)ELOUT                                 !
              DO 340 I=4,NLINES                                !
340           READ(7,520)DUMMY                                 !
        ELSE                                                   ! IF ENTRY IS NOT REQ'D
              IF(INPUT4.EQ.LAST)THEN                           ! AND IS LAST, THEN
                    WRITE(5,350)INPUT4                         ! WRITE ERROR MESSAGE.
350                 FORMAT(                                    !
     *              '0SORRY - NO INFORMATION ON MODEL ',A4)!
                    REWIND 7                                   !
                    GOTO 250                                   !
              ELSE                                             ! IF ENTRY IS NOT REQ'D
                    DO 360 I=1,NLINES                          ! BUT IS NOT LAST, THEN
```

TABLE 3.1 Listing of Program Generator subroutine PGHELP (continued)

```
360               READ(7,520)DUMMY                        ! SKIP LINES AND READ
                  GOTO 330                                ! NEXT ENTRY.
            END IF                                        !
      END IF                                              !
370   READ(6,560)INPUT4,NLINES                            ! DO THE SAME SORT OF
      IF(INPUT4.EQ.NAME4)THEN                             ! THING IN THE INFORM-
            WRITE(5,380)NAME4,STRING                      ! ATION DATA FILE. IF
380         FORMAT('1',A4,2X,A74)                         ! ENTRY FOUND, WRITE
            DO 390 I=1,NLINES                             ! THIS INFORMATION TO
            READ(6,590)INFO                               ! THE TERMINAL
390         WRITE(5,400)INFO                              ! TOGETHER WITH THE
400         FORMAT(1X,A80)                                ! ATTRIBUTES READ FROM
            WRITE(5,410)MINEL,MAXEL,ELIN,ELOUT,NIL,NSIG,NSV!   COMPON.DAT
410         FORMAT(                                       !
     *      '0EXTERNAL LINKS:  MINIMUM NUMBER = ',I1,/,!
     *      '                  MAXIMUM NUMBER = ',I1,/,!
     *      '                  INPUTS         = ',8A1,/,!
     *      '                  OUTPUTS        = ',8A1,/,!
     *      ' NUMBER OF INTERNAL LINKS  = ',I1,/,       !
     *      ' NUMBER OF SIGNALS         = ',I1,/,       !
     *      ' NUMBER OF STATE VARIABLES = ',I1,/,       !
     *      'OPRESS RETURN TO CONTINUE',$)              !
            READ(5,520)DUMMY                              !
      ELSE                                                ! IF NO ENTRY IN INFO-
            IF(INPUT4.EQ.LAST)THEN                        ! RMATION DATA FILE,
                  WRITE(5,410)MINEL,MAXEL,ELIN,ELOUT,NIL,NSIG,NSV!
                  READ(5,520)DUMMY                        ! THEN JUST WRITE THE
                                                          ! ATTRIBUTES READ IN
                                                          ! COMPON.DAT.
            ELSE                                          ! IF ENTRY NOT YET
                  DO 420 I=1,NLINES                       ! FOUND, SKIP REQUIRED
420               READ(6,520)DUMMY                        ! NUMBER OF LINES AND
                  GOTO 370                                ! READ NEXT ENTRY.
            END IF                                        !
      END IF                                              !
      REWIND 6                                            ! GO TO MAIN HELP
      REWIND 7                                            ! OPTION.
      GOTO 10                                             !
                                                          !
C *****************************************************************************
C ***** PROCEED WITH SIMULATION
C *****************************************************************************
                                                          !
500   CLOSE (6)                                           ! CLOSE THE INFORMATION
      REWIND 7                                            ! DATA FILE - SAME LUN
      RETURN                                              ! USED IN PGIN FOR
                                                          ! SYSTEM DATA FILE.
                                                          ! REWIND COMPON.DAT AND
                                                          ! RETURN.
                                                          !
C *****************************************************************************
C ***** ALL INPUT FORMATS BUT NONE OF THE OUTPUT FORMATS
C *****************************************************************************
                                                          !
510   FORMAT(I1)                                          !
520   FORMAT(A1)                                          !
530   FORMAT(A2)                                          !
540   FORMAT(2A2,I2,A74)                                  !
550   FORMAT(A4)                                          !
560   FORMAT(A4,I2)                                       !
570   FORMAT(5I1)                                         !
580   FORMAT(8A1)                                         !
590   FORMAT(A80)                                         !

      END
```

TABLE 3.1 Listing of Program Generator subroutine PGHELP (continued)

Type of model	Advantages	Disadvantages
Instantaneous first principle model	1. For accuarate parametric data, likely to give reasonably accurate results over all operating ranges	1. Often very sensitive to accuracy of input information 2. Relies upon theory which is not always definitive 3. Difficult to program 4. Slow to execute 5. Assumes very fast response
Instantaneous characteristic model	1. Input information readily available 2. Easy to program 3. Fast to execute	1. May not fully describe the behaviour of the model in all its modes of operation 2. Assumes very fast response
Dynamic first principle model	As for instantaneous first principle model	As for instantaneous first principle model (1-4) but to a greater degree
Dynamic characteristic model	As for instantaneous characteristic model	As for instantaneous characteristic model (1); also it may be necessary to perform a test simulation to define the coefficients of the differential eqn.

TABLE 3.2 A comparison of modelling methods

APPENDIX A - THE INTERPRETER STRUCTURE

A.1 INTRODUCTION

100. The purpose of the HASP command interpreter has been discussed in Chapter 2. This Appendix gives a more detailed account of the structure of the interpreter. Under the VMS operating system, the interpreter takes the form of a command procedure, i.e., a procedure defined by the Digital Command Language (DCL). A complete listing of the procedure is given in table A1. The information contained in this Appendix should be sufficient to produce a similar interpreter for an operating system other than VMS. In fact, the interpreter can be written in any language which supports logical if statements, character manipulation and the issuing of system commands. Special attention is given to the use of intrinsic system functions.

A.2 PRIMARY LOGIC

200. Figure A1 gives an outline of the interpreter structure. Initially, an introductory message is displayed as shown in figure A2. A prompt is displayed inviting the user to type a command such as "GENERATE" or "SIMULATE" etc. A series of logical if statements direct control to the relevant program section. If none of the if statements are satisfied, then the user has typed an unacceptable command. In this case, a list of acceptable commands is displayed followed by the prompt.

201. A facility of the interpreter is that the user may opt to be

taken through a standard sequence of tasks without specifying the commands explicitly. This sequence is GENERATE, LINK, SIMULATE, DRAW, EXIT i.e. generate a new circuit, produce the simulation program, run the simulation program, look at the results then stop. The user is informed of the task that would automatically follow at each stage. This facility can of course be overridden by simply typing an acceptable command.

202. This facility is achieved by using a simple character flag. The flag is initially set to "1". The command GENERATE transfers control to the program section STAGE_1. At the end of the generation process, the flag is set to "2". Similarly, the command SIMULATE transfers control to the program section STAGE_2, the flag is subsequently set to "3" and so on. If the user simply presses <RETURN> rather than typing a command, then the statement

```
$ IF COMMAND .EQS. "" THEN GOTO STAGE_'FLAG'
```

will transfer control to the next command in the automatic sequence.

203. In detail, the statement compares the contents of the variable COMMAND to the string contained between the double quotes. If they are the same (i.e. the user has not typed a command),then control is transferred to the program section whose name is made up of the characters STAGE followed by the contents of the variable FLAG (i.e. 1, 2, 3 etc.).

204. A further facility of the interpreter is that the user need

only type sufficient characters to make the command unambiguous. For example, in the case of GENERATE, then the command G is sufficient. However, the procedure compares all of the characters defined by the user to the relevant part of the acceptable commands. Therefore, commands such as GEN or GENER would be acceptable but GRAPH would not.

205. When command procedures are invoked, the operating system assumes that the input required by programs is contained in the procedure. However, when a program such as the program generator is being run, then the input must obviously be provided by the user. Therefore, the input stream must be redefined. This is achieved in DCL by the command

```
$ DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
```

This command applies only to the following executable task.

206. Three of the options (GENERATE, SIMULATE and EXIT) are simple and require no further detail. For example, the command GENERATE causes the procedure to issue the DEFINE command shown above followed by the commands RUN [PG.A]PGA and FLAG="2". The commands which do require further description are LINK, DRAW, HELP, BATCH and VMS.

The LINK program section (STAGE 2)

207. The LINK command carries out four tasks. Firstly it deletes generated object code and simulation programs remaining from previous simulations. Secondly, it purges all related data files and

generated simulation source. Thirdly, it compiles the newly generated source and finally, it links all the necessary object modules to form the new simulation program. In simple terms, it is the link between the program generator and the simulation program. The commands related to this process are contained in a separate command file called CAD.COM, a listing of which is given in table A2.

208. The feature which requires explanation is concerned with the reassignment of the two streams SYS$ERROR and SYS$OUTPUT. Whenever an attempt is made to delete files which do not exist, the system considers that a non-fatal error has occurred and duly transmits messages to this effect down the two streams mentioned above. However, during the deleting and purging process carried out under the LINK command, it is likely that a large number of the files catered for, will not actually exist. The resulting error messages are prolifically produced and are at best, of little interest, and at worst, annoying. Therefore, the messages are dumped in two temporary files. The stream SYS$ERROR is assigned to the file SYSERR.TMP (the extension TMP to denote temporary) and the stream SYS$OUTPUT is assigned to the file SYSOUT.TMP. As soon as the deleting and purging process is complete, these two temporary files are deleted and the two streams are deassigned.

The DRAW program section (STAGE 4)

209. Figure A3 shows a flow diagram for the program sections employed to produce graphs of simulation results. Four graphics programs are currently available:

GRF plots any item of information against time (default)

XYP plots any item of information against any other item

P2P plots two items of information against time

UPD plots the same item of information from two consecutive simulations against time

In addition, the command EXIT may be typed.

210. The logic of this part of the procedure is similar to that employed to transfer control based upon the initial options as described above. However, the prompt now displayed is GRAPHICS: rather than HASP: indicating to the user the fact that control has passed to another part of the procedure. Again, the first character of the command is sufficient to be unambiguous.

The HELP program section

211. The logic of the HELP preogram section is similar to the primary logic described above. However, a different prompt is displayed (HELP:) and only the first character of the command need be typed.

The BATCH program section

212. The VMS operating system does not allow more than one command issued from a given terminal to be active at any one time. Therefore, if a user wishes to run more than one simulation at once, then he must submit them to the batch queue. The batch processor will allow a number of jobs to be active but they will be run with a lower priority than the foreground jobs i.e. those run from a

terminal. When a job is submitted to batch, all four streams normally taken as the terminal must now be reassigned to various files. These streams are given below.

SYS$COMMAND	commands sent to the operating system
SYS$INPUT	interactive input required by a program
SYS$ERROR	messages from the operating system to the user
SYS$OUTPUT	interactive output from a program

213. If a user of the HASP command interpreter issues the BATCH command, then the auxiliary command file CADBATCH.COM will be invoked, a listing of which is given in Table A3. This command file determines the nature of the job to be submitted to the batch queue. It is also a form of coding generator since it writes the two files required to satisfy the input streams SYS$COMMAND and SYS$INPUT. The commands are almost entirely sequential. Therefore, a step by step literary description is all that is required.

(i) The procedure determines the directory or sub-directory from which the job is to be run. When a job is submitted to batch, the user effectively logs in again (though the user is unaware of it). Therefore, a sub-directory from which the command interpreter is being run will not be the sub-directory from which the batch job will be run unless it is specifically stated as being so.

(ii) The procedure determines the name of the simulation program to be submitted. The default is CAD.EXE which resides in the directory or sub-directory defined above.

(iii) The procedure determines the name of the data file containing the parametric data required by the simulation program. This is normally PARAM.DAT but if more than one job is to be submitted, then the data files must be named differently to ensure that each job uses the correct data file.

(iv) The procedure determines the name of the job to be submitted. This name is attached not only to the job entered in the batch queue, but also to all the four files satisfying the four streams. Again, if more than one job is to be submitted, then they must each have a unique job name.

(v) If the procedure is being used before 1700 hrs, then the user is given the option to hold the job until after 1700 hrs in order to avoid peak computer usage times.

(vi) The user is also given the option of attaching a CPU time limit of one hour to the job. This is useful in the case of an experimental simulation program where the integrity of the coding is uncertain. If a job is terminated due to this CPU limit being exceeded, the simulation results up to that point are saved.

(vii) The procedure then determines the simulation time and the print interval in seconds. This information is normally supplied by the user at run time but as stated previously, the SYS$INPUT stream must be satisfied by a file.

(viii) The files satisfying the SYS$COMMAND and SYS$INPUT are then written using the information gained above. An example of these two files are given in Table A4.

(ix) Finally, an appropriate SUBMIT command is issued, again based

upon the information gained above.

214. The two output streams are satisfied by the files jobnameOUT.TMP and jobname.LOG. The first contains all the simulation output normally seen at the terminal. The second contains all system messages including information about CPU time used etc. The two files satisfying the command and input streams, jobname.COM and jobnameIN.TMP are both deleted upon completion of the batch job.

The VMS program section

215. VMS is perhaps the most unusual of the commands, its purpose being to allow users to issue standard VMS instructions whilst operating in the procedure. In practise, VMS has proved to be a very useful command allowing the user to quickly investigate without losing continuity in terms of his work within HASP.

216. The program section concerned with VMS performs three functions. Firstly, it gives a VMS: prompt and waits for a VMS command. The command is stored in the variable called VMS_COMMAND. If the user presses <RETURN>, control is returned to the main option list. Secondly, a check is made to ensure the user has not typed the command MONITOR (or any part of the command). MONITOR is a VMS utility which requires the use of an abort control character in order to stop its operation. The abort control character also aborts the command interpreter and is therefore considered illegal within HASP. Finally, if the command is acceptable, the procedure simply issues the command to the system taking care to redefine the input stream in

case the ensuing VMS operation requires user interaction.

217. Experienced users of both HASP and VMS will almost certainly find this feature a little cumbersome, preferring simply to leave the interpreter. However, it must be remembered that one of the objectives of this package is to appeal to engineers who are not experienced with VMS. To these users, it must seem as if the computer system is HASP with VMS simply providing a subset of commands.

Intrinsic functions

218. A small number of intrinsic functions are used by the interpreter. Although these may not be available under other operating systems in the form shown, there should normally be some equivalent.

219. The first intrinsic function used is F$TIME. This function simply returns the current time and date. It is used for two reasons. Firstly, it is used in order to give a greeting when the procedure is first invoked. Secondly, it is used in batch mode in order to decide whether or not the question concerning holding jobs until after 1700 hrs should be asked (para.213). This function is obviously superfluous to the basic running of the interpreter.

220. The second function used is F$LENGTH which returns the number of characters which constitute a given string. An example of the use of this function is given below.

```
$ SIZE = F$LENGTH(COMMAND)
```

This line finds the number of characters in the string variable COMMAND and assigns it to the integer variable SIZE.

221. Finally, the third intrinsic function used is F$EXTRACT which extracts certain characters from a given string. An example of the use of this function is given below.

```
$ IF F$EXTRACT(0,SIZE,"GENERATE") .EQS. COMMAND THEN GOTO STAGE_1
```

This line extracts a number of characters from the string "GENERATE" and compares the resulting string to the string contained in the variable COMMAND. The first argument defines that the offset should be zero characters. The second argument defines the number of characters to be extracted from the string, i.e. the integer contained in the variable SIZE. In simple terms, the command checks to find if the contents of COMMAND is any portion of the string "GENERATE" and if it is, control is transferred to the program section which carries out the generation process.

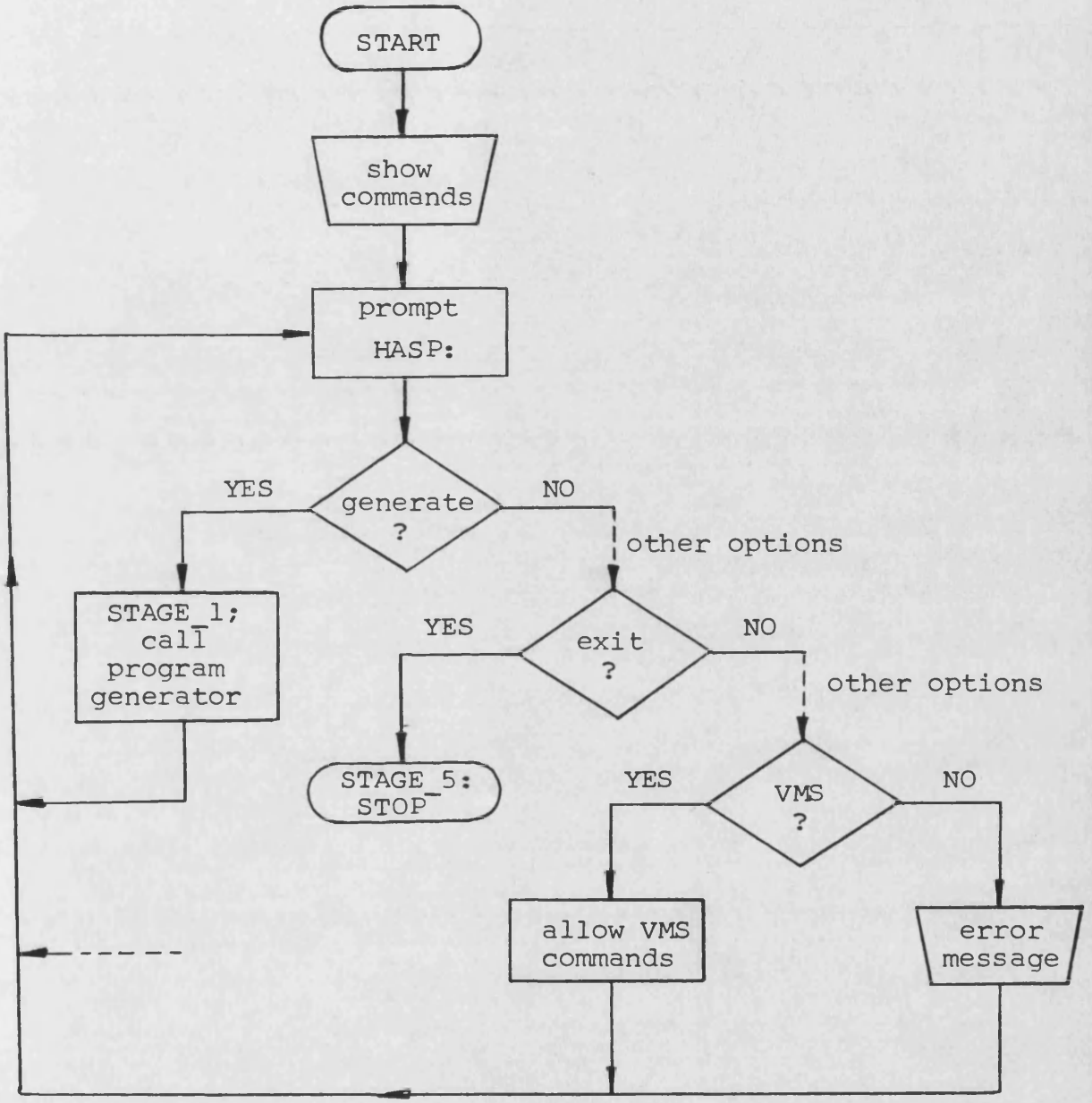


FIG. A.1 Flow diagram for the HASP Command Interpreter

```
HASP Command Interpreter Version 1.0                University of Bath

Good morning

The commands you can enter are:

 GENERATE        Generate the source of a simulation program
 LINK            Produce the simulation task
 SIMULATE        Run the simulation program
 DRAW            View the simulation results (Tektronix only)
 EXIT            Exit from HASP Command Interpreter
 BATCH           Submit a simulation program to run on batch
 HELP            Obtain more information about these options
 VMS             Have the ability to type VMS commands

HASP:_
```

FIG. A.2 The introductory message

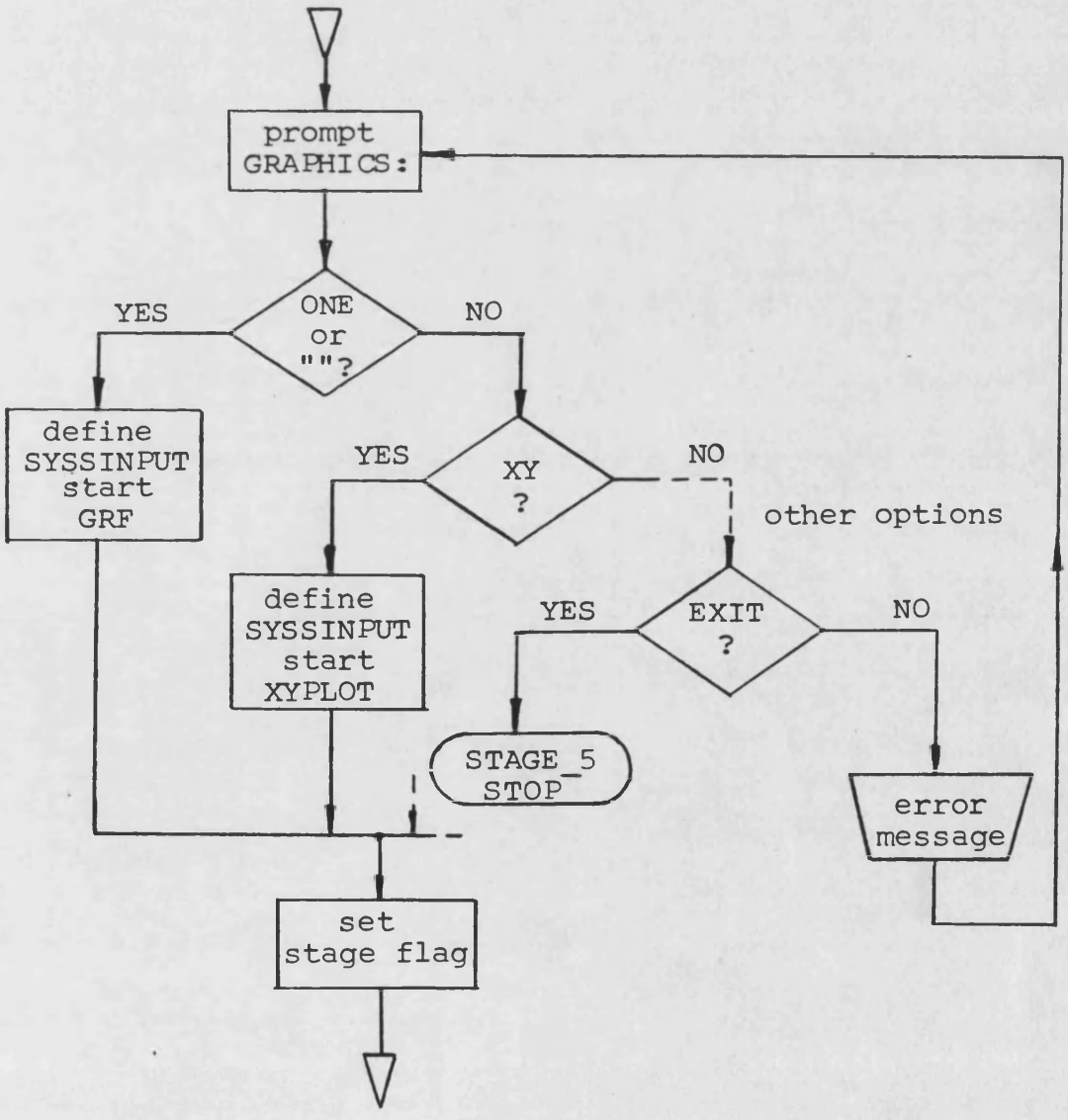


FIG. A.3 Flow diagram for the GRAPHICS facility

```
$ !*****************************************************************************
$ !     COMMAND PROCEDURE FOR USERS OF HASP - a command line interpreter
$ !*****************************************************************************
$ !                                           Fluid Power Group
$ !                                           School of Engineering
$ !                                           University of Bath
$ !                                           England
$ !     Developed:      S.R. Hull
$ !     Date:           05-MAY-84
$ !     Amended:
$ !*****************************************************************************
$ !     String Variable Names:
$ !             COMMAND          User defined command in HASP mode
$ !             COMMAND_SIZE     Number of characters in COMMAND
$ !             DUMMY            Dummy variable to allow a halt in write
$ !             FLAG             Counter to show next stage by default
$ !             GRAPH_TYPE       User defined command in GRAPHICS mode
$ !             TYPE_SIZE        Number of characters in GRAPH_TYPE
$ !             TOPIC            User defined command in HELP mode
$ !             TOPIC_SIZE       Number of characters in TOPIC
$ !             VMS_COMMAND      User defined command in VMS mode
$ !*****************************************************************************
$ !     Program sections:
$ !             AFTERNOON        Writes Good afternoon if between 1200 and 1700
$ !             BATCH            Submits a simulation run to batch
$ !             CONTINUE         Continue with remainder of introduction
$ !             END_GRAPHICS     End of STAGE_4
$ !             EVENING          Writes Good evening if between 1700 and 2400
$ !             GRF              Runs [PG.GRAF]GRF.EXE
$ !             HELP_BATCH       List information about BATCH command
$ !             HELP_DRAW        List information about DRAW command
$ !             HELP_EXIT        List information about EXIT command
$ !             HELP_GENERATE    List information about GENERATE command
$ !             HELP_LINK        List information about LINK command
$ !             HELP_LONG        Control section for major help routines
$ !             HELP_SHORT       Gives a short list of acceptable HASP commands
$ !             HELP_SIMULATE    List information about SIMULATE command
$ !             HELP_VMS         List information about VMS command
$ !             ILLEGAL_VMS      Variable VMS_COMMAND is unacceptable
$ !             PROMPT           Gives HASP prompt
$ !             P2P              Runs [PG.GRAF]P2P.EXE
$ !             STAGE_1          Runs [PG.A]PGA.EXE
$ !             STAGE_2          Invokes [PG.HCI]CAD.COM
$ !             STAGE_3          Runs [default]CAD.EXE
$ !             STAGE_4          Controls which GRAPHICS program is to be used
$ !             STAGE_5          Exits from HASP1.HCI
$ !             UPD              Runs [PG.GRAF]UPD.EXE
$ !             VMS_EXECUTE      Carries out user defined VMS command
$ !             VMS_MODE         Gives VMS prompt
$ !             XYP              Runs [PG.GRAF]XYP.EXE
$ !*****************************************************************************
$ !     Associated system routines:
$ !             F$EXTRACT(a,b,c) Extracts the string b characters long starting
$ !                              with an offset a from input string c
$ !             F$LENGTH (a)     Finds the length of string a
$ !             F$TIME   ()      Returns the time
$ !*****************************************************************************
$ !     Associated tasks:                                        Location:
$ !             PGA      Standard program generator              [PG.A]
$ !             CAD      Generated simulation program            [default]
$ !             GRF      Graphics (y     vs. T)                  [PG.GRAF]
$ !             XYP      Graphics (y     vs. x)                  [PG.GRAF]
$ !             P2P      Graphics (ya,yb vs. T)                  [PG.GRAF]
```

TABLE A1(a) Listing of the command interpreter

```
$ !             UPD     Graphics (y1,y2 vs. T)                  [PG.GRAF]
$ !*************************************************************************
$ !     Associated command files:                               Location:
$ !             CAD.COM         Produces CAD.EXE                [HASP.COMPON]
$ !             CADBATCH.COM    Submits a job to batch          [PG.HC]]
$ !*************************************************************************
$ !     Associated data files:                                  Loc'n and type:
$ !             system.DAT      User named file containing a    [default]
$ !                             description of the circuit      (sequential)
$ !             COMPON.DAT      File containing the attributes  [HASP.COMPON]
$ !                             of all components               (sequential)
$ !             PARAM.DAT       File containing a parametric    [default]
$ !                             description of the components   (sequential)
$ !             CADRES.DAT      File containing the simulation  [default]
$ !                             results                         (direct access)
$ !*************************************************************************
$ !
$ !**** Initialise FLAG
$ !
$       FLAG = "1"
$ !
$ !**** Type Introduction
$ !
$       TYPE SYS$INPUT

       HASP Command Interpreter Version 1.0            University of Bath

$ !
$ !**** Find out if it is morning, afternoon or evening
$ !
$ MORNING:
$       IF F$EXTRACT(12,2,F$TIME()) .GES. 12 THEN GOTO AFTERNOON
$       WRITE SYS$OUTPUT "        Good morning"
$       GOTO CONTINUE
$ AFTERNOON:
$       IF F$EXTRACT(12,2,F$TIME()) .GES. 17 THEN GOTO EVENING
$       WRITE SYS$OUTPUT "        Good afternoon"
$       GOTO CONTINUE
$ EVENING:
$       WRITE SYS$OUTPUT "        Good evening"
$ !
$ !**** Give a short summary of the HASP commands
$ !
$ CONTINUE:
$       GOTO HELP_SHORT
$ !
$ !**** Give a HASP prompt
$ !
$ PROMPT:
$       INQUIRE COMMAND "HASP"
$ !
$ !**** Continue to next stage by default
$ !
$       IF COMMAND .EQS. "" THEN GOTO STAGE_'FLAG'
$ !
$ !**** Find out length of command
$ !
$       COMMAND_SIZE = F$LENGTH(COMMAND)
$ !
$ !**** Check that the command typed in is acceptable and if so, go to the
$ !**** corresponding section
$ !
$       IF F$EXTRACT(0,COMMAND_SIZE,"GENERATE") .EQS. COMMAND -
                                                  THEN GOTO STAGE_1
$       IF F$EXTRACT(0,COMMAND_SIZE,"LINK")     .EQS. COMMAND -
                                                  THEN GOTO STAGE_2
$       IF F$EXTRACT(0,COMMAND_SIZE,"SIMULATE") .EQS. COMMAND -
                                                  THEN GOTO STAGE_3
$       IF F$EXTRACT(0,COMMAND_SIZE,"DRAW")     .EQS. COMMAND -
                                                  THEN GOTO STAGE_4
```

TABLE A1(b) Listing of the command interpreter (continued)

```
$       IF F$EXTRACT(0,COMMAND_SIZE,'EXIT')     .EQS. COMMAND -
                                                  THEN GOTO STAGE_5
$       IF F$EXTRACT(0,COMMAND_SIZE,'BATCH')    .EQS. COMMAND -
                                                  THEN GOTO BATCH_MODE
$       IF F$EXTRACT(0,COMMAND_SIZE,'HELP')     .EQS. COMMAND -
                                                  THEN GOTO HELP_LONG
$       IF F$EXTRACT(0,COMMAND_SIZE,'VMS')      .EQS. COMMAND -
                                                  THEN GOTO VMS_MODE
$ !
$ !**** Command is unacceptable - Write error message and give a short
$ !**** list of acceptable commands
$ !
$       WRITE SYS$OUTPUT '        Invalid command'
$       GOTO HELP_SHORT
$ !
$ !*************************************************************************
$ !     Stage 1:  Run the program generator.
$ !               Do not allow user to interrupt the program.
$ !               Update FLAG to allow user to continue with LINK by default.
$ !*************************************************************************
$ !
$ STAGE_1:
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       RUN [PG.A]PGA
$       FLAG = '2'
$       TYPE SYS$INPUT

Press <RETURN> to proceed with compilation and linking
$       GOTO PROMPT
$ !
$ !*************************************************************************
$ !     Stage 2:  Tidy up default directory, compile new source and link
$ !               the simulation program.
$ !               Update FLAG to allow user to continue with SIMULATE
$ !               by default.
$ !*************************************************************************
$ !
$ STAGE_2:
$       @[PG.HC]JCAD
$       FLAG = '3'
$       TYPE SYS$INPUT

Press <RETURN> to proceed with simulation
$       GOTO PROMPT
$ !
$ !*************************************************************************
$ !     Stage 3:  Run the simulation program.
$ !               Update FLAG to allow user to continue with DRAW by default.
$ !*************************************************************************
$ !
$ STAGE_3:
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       RUN CAD
$       FLAG = '4'
$       TYPE SYS$INPUT

Press <RETURN> to produce graphical output (ONLY ON TEKTRONIX TERMINALS)
$       GOTO PROMPT
$ !
$ !*************************************************************************
$ !     Stage 4:  Run one of the graphics programs.
$ !               Update FLAG to allow user to continue with EXIT by default.
$ !*************************************************************************
```

TABLE A1(c) Listing of the command interpreter (continued)

```
$ !
$ STAGE_4:
$       TYPE SYS$INPUT

        There is a choice of four graphics programs

        GRF plots any item of information against time (default)
        XYP plots any item of information against any other item
        P2P plots any two items of information against time
        UPD plots the same item of information from two consecutive
            simulations against time

        Type name of program required (or EXIT)

$ !
$ !**** Give GRAPHICS prompt and allow choice of GRF by default
$ !
$       INQUIRE GRAPH_TYPE 'GRAPHICS'
$       IF GRAPH_TYPE .EQS. "" THEN GOTO GRF
$       TYPE_SIZE = F$LENGTH(GRAPH_TYPE)
$ !
$ !**** Go to correct section if command is valid
$ !
$       IF F$EXTRACT(0,TYPE_SIZE,'GRF')  .EQS. GRAPH_TYPE THEN GOTO GRF
$       IF F$EXTRACT(0,TYPE_SIZE,'XYP')  .EQS. GRAPH_TYPE THEN GOTO XYP
$       IF F$EXTRACT(0,TYPE_SIZE,'P2P')  .EQS. GRAPH_TYPE THEN GOTO P2P
$       IF F$EXTRACT(0,TYPE_SIZE,'UPD')  .EQS. GRAPH_TYPE THEN GOTO UPD
$       IF F$EXTRACT(0,TYPE_SIZE,'EXIT') .EQS. GRAPH_TYPE THEN GOTO STAGE_5
$ !
$ !**** Command is unacceptable - Write error message and go back to
$ !**** beginning of Stage 4.
$ !
$       WRITE SYS$OUTPUT "         Invalid program name"
$       GOTO STAGE_4
$ !
$ !**** RUN GRF
$ !
$ GRF:
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       RUN [PG.GRAF]GRF
$       GOTO END_GRAPHICS
$ !
$ !**** RUN GRF
$ !
$ XYP:
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       RUN [PG.GRAF]XYP
$       GOTO END_GRAPHICS
$ !
$ !**** RUN GRF
$ !
$ P2P:
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       RUN [PG.GRAF]P2P
$       GOTO END_GRAPHICS
$ !
$ !**** RUN GRF
$ !
$ UPD:
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       RUN [PG.GRAF]UPD
$ END_GRAPHICS:
$       FLAG = '5'
```

TABLE A1(d) Listing of the command interpreter (continued)

```
$       TYPE SYS$INPUT

Press <RETURN> to exit
$       GOTO PROMPT
$ !
$ !*************************************************************************
$ !     Stage 5:  Exit form HASP Command Interpreter
$ !*************************************************************************
$ !
$ STAGE_5:
$       EXIT
$ !
$ !*************************************************************************
$ !     Submit a simulation program to run on batch
$ !*************************************************************************
$ !
$ BATCH_MODE:
$       @[PG.HCIJCADBATCH.COM
$       FLAG = "5"
$       TYPE SYS$INPUT

Press <RETURN> to exit
$       GOTO PROMPT
$ !
$ !*************************************************************************
$ !     Give a short list of acceptable commands under HASP
$ !*************************************************************************
$ !
$ HELP_SHORT:
$       TYPE SYS$INPUT

        The commands you can enter are:

         GENERATE       Generate the source of a simulation program
         LINK           Produce the simulation task
         SIMULATE       Run the simulation program
         DRAW           View the simulation results (Tektronix only)
         EXIT           Exit from HASP Command Interpreter
         BATCH          Submit a simulation program to run on batch
         HELP           Obtain more information about these options
         VMS            Have the ability to type VMS commands

$       GOTO PROMPT
$ !
$ !*************************************************************************
$ !     Give more information about a user selected command
$ !*************************************************************************
$ !
$ HELP_LONG:
$       TYPE SYS$INPUT

        GENERATE        Generate the source of a simulation program
        LINK            Produce the simulation task
        SIMULATE        Run the simulation program
        DRAW            View the simulation results (Tektronix only)
        EXIT            Exit from HASP Command Interpreter
        BATCH           Submit a simulation program to run on batch
        VMS             Have the ability to type VMS commands

        Press <RETURN> to continue

$ !
$ !**** Display TOPIC prompt and return to HASP if no command on line
```

TABLE A1(e) Listing of the command interpreter (continued)

```
$ !
$       INQUIRE TOPIC 'HELP'
$       IF TOPIC .EQS. '' THEN GOTO PROMPT
$       TOPIC_SIZE = F$LENGTH(TOPIC)
$ !
$ !**** Go to correct section if topic is valid
$ !
$       IF F$EXTRACT(0,TOPIC_SIZE,'GENERATE') .EQS. TOPIC -
                                                    THEN GOTO HELP_GENERATE
$       IF F$EXTRACT(0,TOPIC_SIZE,'LINK')     .EQS. TOPIC -
                                                    THEN GOTO HELP_LINK
$       IF F$EXTRACT(0,TOPIC_SIZE,'SIMULATE') .EQS. TOPIC -
                                                    THEN GOTO HELP_SIMULATE
$       IF F$EXTRACT(0,TOPIC_SIZE,'DRAW')     .EQS. TOPIC -
                                                    THEN GOTO HELP_DRAW
$       IF F$EXTRACT(0,TOPIC_SIZE,'EXIT')     .EQS. TOPIC -
                                                    THEN GOTO HELP_EXIT
$       IF F$EXTRACT(0,TOPIC_SIZE,'BATCH')    .EQS. TOPIC -
                                                    THEN GOTO HELP_BATCH
$       IF F$EXTRACT(0,TOPIC_SIZE,'VMS')      .EQS. TOPIC -
                                                    THEN GOTO HELP_VMS
$ !
$ !**** Command is unacceptable - Write error message and go back to
$ !**** beginning of HELP_LONG.
$ !
$       WRITE SYS$OUTPUT '         Invalid topic'
$       GOTO HELP_LONG
$ !
$ !**** Information on GENERATE command
$ !
$ HELP_GENERATE:
$       TYPE SYS$INPUT

GENERATE:

        The command GENERATE invokes the standard  HASP program generator.
        This task image is called  PGA.EXE  and  it resides  in  directory
        [PG.A] on the RA80 disc.

        The input required by the program is a complete description of the
        hydraulic circuit to be simulated.  This takes  the form of a list
        of HASP component names together with the  links  associated  with
        each model.

        The program generator produces several files which will, in  part,
        form a program to simulate a hydraulic  circuit.  The names of the
        files are  MAIN.FOR,  AUX.FOR,  OUT.FOR,  CONTRL.FOR  and CAD.OPT.
        These files will reside in the default directory.

        The command G is sufficient.

        For more information, see HASP USERS GUIDE.

                                                          Press <RETURN>

$       INQUIRE DUMMY 'HELP'
$       GOTO HELP_LONG
$ !
$ !**** Information on LINK command
$ !
$ HELP_LINK:
$       TYPE SYS$INPUT
```

TABLE A1(f) Listing of the command interpreter (continued)

```
LINK:

        The command LINK invokes the standard HASP command procedure
        CAD.COM which takes the generated segments and creates a simulation
        program from them.  CAD.COM resides in directory [PG.HCI] on
        the RA80 disc.

        The file carries out the following tasks:
        1.  It deletes files created by previous program generations in the
            default directory (data files are only purged).
        2.  It compiles the four FORTRAN files for the current simulation.
        3.  It links these files with the integrator, standard utility
            routines and model routines to form the simulation program
            called CAD.EXE.

        The command L is sufficient.

        For more information, see HASP USERS GUIDE.

                                                          Press <RETURN>

$       INQUIRE DUMMY "HELP"
$       GOTO HELP_LONG
$ !
$ !**** Information on SIMULATE command
$ !
$ HELP_SIMULATE:
$       TYPE SYS$INPUT

SIMULATE:

        The command SIMULATE runs the most recently produced simulation
        program.  This program is called CAD.EXE and is resident in the
        default directory.

        If it is the first time a simulation program has been run, then
        the user will be required to interactively define the parametric
        data for every component in the simulation.  On completion of the
        first run, these results are stored in a file called PARAM.DAT
        which is resident in the default directory.  On subsequent runs,
        this data is retrieved and may be amended by the user as reuired.

        The results produced by the simulation program are stored in a
        binary data file called CADRES.DAT which is resident in the
        default directory.

        The command S is sufficient.

        For more information, see HASP USERS GUIDE.
                                                          Press <RETURN>

$       INQUIRE DUMMY "HELP"
$       GOTO HELP_LONG
$ !
$ !**** Information on DRAW command
$ !
$ HELP_DRAW:
$       TYPE SYS$INPUT

DRAW:

        The command DRAW runs one of the four graphics programs which are
        are resident in directory [PG.GRAF] on the RA80 disc.  The programs
```

TABLE A1(g) Listing of the command interpreter (continued)

```
        access the results data files  CADRES.DAT  which must be resident in
        the default directory.

        The four programs which may be used are:
        1. GRF which plots any item of information from the  latest  results
              file against time;
        2. XYP which plots any item of information against any other item of
              information, both from the latest results file;
        3. P2P which  plots  any two  items  of  information from the latest
              results file against time;
        4. UPD which plots the same item of information from two consecutive
              simulations against time. In this case, the two results files
              must be numbered versions 1 and 2.

        The command D is sufficient.

        For more information, see HASP USERS GUIDE
                                                                Press <RETURN>

$       INQUIRE DUMMY 'HELP'
$       GOTO HELP_LONG
$ !
$ !**** Information on EXIT command
$ !
$ HELP_EXIT:
$       TYPE SYS$INPUT

EXIT:

        The command EXIT exits from the HASP command Interpreter and returns
        the user to DCL level of VMS.

        The command E is sufficient.

                                                                Press <RETURN>

$       INQUIRE DUMMY 'HELP'
$       GOTO HELP_LONG
$ !
$ !**** Information on GENERATE command
$ !
$ HELP_BATCH:
$       TYPE SYS$INPUT

BATCH:

        The command BATCH invokes the command procedure CADBATCH.COM which
        is resident in directory [PG.HCI] on the RA80 disc.

        The procedure requires the user to define the job to be  submitted
        to batch.  This entails the definition of the command  file  which
        contains the RUN command and the definition of  the job name.  The
        user is also given the option to set a CPU time  limit on  the job
        and also to request that the job be run after 1700 hrs.

        The submitted job temporarily creates a file called  (jobname).LOG
        which is resident in the default  directory.  This  file  contains
        all system and error messages (i.e. it is temporarily  SYS$ERROR).
        Upon  completion  of the job, the file is resident in the  default
        directory.

        The command B is sufficient.
```

TABLE A1(h) Listing of the command interpreter (continued)

```
                                                                Press <RETURN>

$       INQUIRE DUMMY "HELP"
$       GOTO HELP_LONG
$ !
$ !**** Information on VMS command
$ !
$ HELP_VMS:
$       TYPE SYS$INPUT

VMS:

        The command VMS allows the user to type VMS commands without the need
        to exit from the HASP Command Interpreter.

        All commands except MONITOR (and therfore ACT) are allowed.  However,
        any  VMS command that requires user interaction will be unsuccessful.
        It should also be noted that if ^Y is typed to abort the VMS command,
        then the command interpreter will also be aborted.

        The command V is sufficient.

                                                                Press <RETURN>

$       INQUIRE DUMMY "HELP"
$       GOTO HELP_LONG
$ !
$ !*************************************************************************
$ !     VMS mode:  Allow the user to type VMS commands
$ !*************************************************************************
$ !
$ VMS_MODE:
$ !
$ !**** Display VMS prompt
$ !
$       TYPE SYS$INPUT

Press <RETURN> to return to HASP Command Interpreter
$       INQUIRE VMS_COMMAND "VMS"
$       IF VMS_COMMAND .EQS. "" THEN GOTO PROMPT
$ !
$ !**** Check that command was not MON
$ !
$       IF F$EXTRACT(0,3,VMS_COMMAND) .NES. "MON" THEN GOTO VMS_EXECUTE
$ INVALID_VMS:
$       TYPE SYS$INPUT

        Invalid command - MONITOR is not allowed under HASP

$       GOTO VMS_MODE
$ !
$ !**** Carry out command as required
$ !
$ VMS_EXECUTE:
$       DEFINE/USER_MODE SYS$INPUT SYS$COMMAND:
$       VMS_COMMAND
$       GOTO VMS_MODE
$ !
$ !**** End of HASP Command Interpreter
$ !
```

TABLE A1(i) Listing of the command interpreter (continued)

```
$ !*****************************************************************************
$ !     COMMAND PROCEDURE TO PRODUCE CAD.EXE
$ !*****************************************************************************
$ !                                                 Fluid Power Group
$ !                                                 School of Engineering
$ !                                                 University of Bath
$ !                                                 England
$ !     Developed:      S.R. Hull
$ !     Date:           07-MAY-84
$ !     Amended:
$ !*****************************************************************************
$ !     Associated files:                                          Location:
$ !             CAD.OPT         Linker options file created by  [default]
$ !                             the program generator
$ !*****************************************************************************
$       TYPE SYS$INPUT

DELETING OR PURGING FILES REMAINING FROM PREVIOUS SIMULATIONS

$ !
$ !**** Redefine SYS$ERROR and SYS$OUTPUT so that system messages do not
$ !**** appear on the screen
$ !
$       DEFINE SYS$ERROR  SYSERR.TMP
$       DEFINE SYS$OUTPUT SYSOUT.TMP
$ !
$ !**** Delete various simulation files
$ !
$       DELETE MAIN.OBJ;*,AUX.OBJ;*,OUT.OBJ;*,CONTRL.OBJ;*,CAD.EXE;*
$ !
$ !**** Purge various simulation files
$ !
$       PURGE PARAM.DAT/K=3,CADRES.DAT/K=3,-
              MAIN.FOR/K=1,AUX.FOR/K=1,OUT.FOR/K=1,CONTRL.FOR/K=1,CAD.OPT/K=1
$ !
$ !**** Deassign SYS$ERROR and SYS$OUTPUT and delete the temporary files
$ !
$       DEASSIGN SYS$ERROR
$       DEASSIGN SYS$OUTPUT
$       DELETE SYSERR.TMP;*
$       DELETE SYSOUT.TMP;*
$ !
$ !**** Compile generated simulation source
$ !
$       TYPE SYS$INPUT

COMPILING THE NEWLY GENERATED SIMULATION SOURCE FILES
$       FORTRAN MAIN,AUX,OUT,CONTRL
$ !
$ !**** Link CAD.EXE with the aid of the linker options file CAD.OPT
$ !
$       TYPE SYS$INPUT

LINKING ALL THE ROUTINES REQUIRED TO FORM THE SIMULATION PROGRAM
$       LINK CAD/OPT
```

TABLE A2 Listing of the LINK command procedure

```
$ !*****************************************************************************
$ !     COMMAND PROCEDURE TO RUN A BATCH SIMULATION JOB
$ !*****************************************************************************
$ !                                                 Fluid Power Group
$ !                                                 School of Engineering
$ !                                                 University of Bath
$ !                                                 England
$ !     Developed:      S.R. Hull
$ !     Date:           06-MAY-84
$ !     Amended:
$ !*****************************************************************************
$ !     String variable names:
$ !             AFTER_TIME      Flag to show if job to be held until 1700 [Y/N]
$ !             CPU_LIMIT       Flag to show if job has 1 hour CPU limit  [Y/N]
$ !             DIRECTORY       User defined directory
$ !             DIRECTORY_LENGTH     Number of characters in DIRECTORY
$ !             FILE_NAME       Name of parametric data file
$ !             JOB_NAME        Name of job to be inserted into SYS$BATCH
$ !             LENGTH_MINUS_1  Number of characters in DIRECTORY minus one
$ !             PRINT_INTERVAL  Simulation print interval in seconds
$ !             TASK_NAME       Name of simulation program to be submitted
$ !             TOTAL_TIME      Total time to be simulated in seconds
$ !             USERNAME        Corrected form of DIRECTORY
$ !*****************************************************************************
$ !     Program sections:
$ !             AFTER_TIME      Find out if job is to be held until 1700
$ !             CHECK_LENGTH    Check that JOB_NAME is 6 characeters or less
$ !             CONTINUE_n      Dummy label
$ !             CPU_LIMIT       Find out if job has 1 hour CPU limit
$ !             DATA_NAME       Find out name of parametric data file
$ !             DETERMINE_USERNAME   Correct user defined DIRECTORY
$ !             DIRECTORY       Find out name of directory to be used
$ !             EXIT            Exit from procedure
$ !             FILE_NAME       Find out name of parametric data file
$ !             JOB_NAME        Find out name of job to be inserted into queue
$ !             LOGIN_DIRECTORY Set USERNAME to SYS$LOGIN
$ !             PRINT_INTERVAL  Find out the simulation print interval
$ !             SUBMIT_GOTO     Go to the correct SUBMIT command
$ !             SUBMIT_1        Submit with /AFTER and /CPU_LIMIT
$ !             SUBMIT_2        Submit with /AFTER but without /CPU_LIMIT
$ !             SUBMIT_3        Submit without /AFTER but with /CPU_LIMIT
$ !             SUBMIT_4        Submit without /AFTER and without /CPU_LIMIT
$ !             TASK_NAME       Find out name of simulation program
$ !             TOO_LATE        It is already after 1700 so no /AFTER
$ !             TOTAL_TIME      Find out total time to be simulated
$ !             WRITE_BATCH     Write the command procedure to be submitted
$ !             WRITE_INPUT     Write the temporary data file [jobname]IN.TMP
$ !*****************************************************************************
$ !     Associated system routines:
$ !             F$EXTRACT(a,b,c) Extracts the string b characters long starting
$ !                              with an offset a from input string c
$ !             F$LENGTH (a)     Returns the number of characters in string a
$ !             F$TIME   ()      Returns the time
$ !*****************************************************************************
$ !     Associated command file:                                  Location:
$ !           [jobname].COM     The batch job procedure           SYS$LOGIN
$ !                             (SYS$COMMAND)
$ !*****************************************************************************
$ !     Associated data files:                                    Loc'n and type:
$ !           [jobname]IN.TMP   File containing the input         SYS$LOGIN
$ !                             noramlly defined by user          (sequential)
$ !                             (SYS$INPUT)
$ !           [jobname]OUT.TMP  File containing the output        SYS$LOGIN
```

TABLE A3(a) Listing of the BATCH command procedure

```
$ !                             normally sent to the terminal   (sequential)
$ !                             (SYS$OUTPUT)
$ !*****************************************************************************
$ !     Associated log file:                                        Location:
$ !           [jobname].LOG     Log of job                          SYS$LOGIN
$ !                             (SYS$ERROR)
$ !*****************************************************************************
$ !
$ !**** Find out name of directory from which job is to be run
$ !
$ DIRECTORY:
$       TYPE SYS$INPUT

        Type directory or sub-directory from which the job is to be run
        [default = SYS$LOGIN]

$       INQUIRE DIRECTORY "BATCH"
$ !
$ !**** Find out the name of the root directory
$ !
$ DETERMINE_USERNAME:
$       IF DIRECTORY .EQS. "" THEN GOTO LOGIN_DIRECTORY
$       IF F$EXTRACT(0,1,DIRECTORY) .EQS. "[" THEN GOTO CONTINUE_1
$       DIRECTORY = "[" + DIRECTORY
$ CONTINUE_1:
$       DIRECTORY_LENGTH = F$LENGTH(DIRECTORY)
$       LENGTH_MINUS_1 = DIRECTORY_LENGTH - 1
$       IF F$EXTRACT(LENGTH_MINUS_1,1,DIRECTORY) .EQS. "]" THEN GOTO CONTINUE_2
$       DIRECTORY = DIRECTORY + "]"
$ CONTINUE_2:
$       USERNAME = DIRECTORY
$       GOTO TASK_NAME
$ !
$ !**** No directory has been defined - Username is the default (SYS$LOGIN)
$ !
$ LOGIN_DIRECTORY:
$       USERNAME = "SYS$LOGIN"
$ !
$ !**** Find out name of simulation program to be submitted
$ !
$ TASK_NAME:
$       TYPE SYS$INPUT

        Type name of simulation program to be run  [default = CAD.EXE]

$       INQUIRE TASK_NAME "BATCH"
$       IF TASK_NAME .NES. "" THEN GOTO FILE_NAME
$       TASK_NAME = "CAD"
$ !
$ !**** Find out name of file containing parametric data
$ !
$ FILE_NAME:
$       TYPE SYS$INPUT

        Type name of the parametric data file [default = PARAM.DAT]

                                CAUTION
        If more than one job is being submitted, the files should be
        named differently

$       INQUIRE FILE_NAME "BATCH"
$ !
$ !**** Allow a default of PARAM.DAT
```

TABLE A3(b) Listing of the BATCH command procedure (continued

```
$ !
$       IF FILE_NAME .NES. "" THEN GOTO JOB_NAME
$       FILE_NAME = 'PARAM.DAT'
$ !
$ !**** Find out name of job to be inserted into queue SYS$BATCH
$ !
$ JOB_NAME:
$       TYPE SYS$INPUT

        Type name of the batch job [maximum 6 characters, default = BATCH]

                                CAUTION
        A user should never submit more than one job of the same name

$       INQUIRE JOB_NAME "BATCH"
$ !
$ !**** Allow a default of BATCH
$ !
$       IF JOB_NAME .NES. "" THEN GOTO CHECK_LENGTH
$       JOB_NAME = "BATCH"
$ !
$ !**** Check that JOB_NAME is not more than 6 characters long
$ !
$ CHECK_LENGTH:
$       IF F$LENGTH(JOB_NAME) .LE. 6 THEN GOTO AFTER_TIME
$       WRITE SYS$INPUT

        ERROR: Jobname must be 6 characters or less

$       GOTO JOB_NAME
$ !
$ !**** Allow job to be held in queue until after 1700 provided it is not
$ !**** already after 1700
$ !
$ AFTER_TIME:
$       IF F$EXTRACT(12,2,F$TIME()) .GES. 17 THEN GOTO TOO_LATE
$       TYPE SYS$INPUT

        Do you want the job to be held until after 1700 hrs ?  [Y/N]

$       INQUIRE AFTER_TIME "BATCH"
$       GOTO CPU_LIMIT
$ !
$ !**** Already after 1700
$ !
$ TOO_LATE:
$       AFTER_TIME = "N"
$ !
$ !**** Allow a CPU time limit of 1 hour to be attached to the job
$ !
$ CPU_LIMIT:
$       TYPE SYS$INPUT

        Do you want to attach a 1 hour CPU time limit to the job ?   [Y/N]

$       INQUIRE CPU_LIMIT "BATCH"
$ !
$ !**** Find out the total simulation time - This information is required
$ !**** by the simulation program but is normally supplied interactively
$ !**** by the user
$ !
$ TOTAL_TIME:
$       TYPE SYS$INPUT
```

TABLE A3(c) Listing of the BATCH command procedure (continued)

```
        Type total simulation time in seconds

$       INQUIRE TOTAL_TIME "BATCH"
$ !
$ !**** Do not allow a negative value
$ !
$       IF F$EXTRACT(0,1,TOTAL_TIME) .EQS. "-" THEN GOTO TOTAL_TIME
$ !
$ !**** Similarly, find out the print interval required
$ !
$ PRINT_INTERVAL:
$       TYPE SYS$INPUT

        Type print interval in seconds

$       INQUIRE PRINT_INTERVAL "BATCH"
$       IF F$EXTRACT(0,1,PRINT_INTERVAL) .EQS. "-" THEN GOTO PRINT_INTERVAL
$       IF F$EXTRACT(0,1,TOTAL_TIME) .EQS. "0" THEN GOTO WRITE_BATCH
$       IF F$EXTRACT(0,1,PRINT_INTERVAL) .EQS. "0" THEN GOTO PRINT_INTERVAL
$ !
$ !**** Write the command procedure to be submitted to batch [jobname.COM]
$ !
$ WRITE_BATCH:
$       OPEN/WRITE OUTFILE 'JOB_NAME'.COM
$       WRITE OUTFILE "$ SET DEFAULT ",USERNAME
$       WRITE OUTFILE "$ DEFINE SYS$INPUT ",JOB_NAME,"IN.TMP"
$       WRITE OUTFILE "$ DEFINE SYS$OUTPUT ",JOB_NAME,"OUT.TMP"
$       IF FILE_NAME .EQS. "PARAM.DAT" THEN GOTO CONTINUE_3
$       WRITE OUTFILE "$ COPY ",FILE_NAME," PARAM.DAT"
$ CONTINUE_3:
$       WRITE OUTFILE "$ RUN ",TASK_NAME
$       WRITE OUTFILE "$ DEASSIGN SYS$OUTPUT"
$       WRITE OUTFILE "$ DEASSIGN SYS$INPUT"
$       WRITE OUTFILE "$ DELETE ",JOB_NAME,".COM;*"
$       WRITE OUTFILE "$ DELETE ",JOB_NAME,"IN.TMP;*"
$       CLOSE OUTFILE
$ !
$ !**** Write the temporary data file [jobname]IN.TMP which will contain the
$ !**** information normally defined interactively
$ !
$ WRITE_INPUT:
$       OPEN/WRITE OUTFILE 'JOB_NAME'IN.TMP
$       WRITE OUTFILE "1"
$       WRITE OUTFILE "0"
$       WRITE OUTFILE TOTAL_TIME
$       WRITE OUTFILE PRINT_INTERVAL
$       WRITE OUTFILE "2"
$       WRITE OUTFILE "0"
$       WRITE OUTFILE "3"
$       CLOSE OUTFILE
$ !
$ !**** Write a warning onto the screen
$ !
$ WARNING:
$       TYPE SYS$INPUT

        The files [jobname]OUT.TMP and [jobname].LOG  will remain after
        completion of the job and will contain all output normally seen
        by the terminal

        WARNING:   Do not tamper with the following files until the job
                   is complete:
```

TABLE A3(d) Listing of the BATCH command procedure (continued)

```
        [jobname].COM  [jobname]IN.TMP  [jobname]OUT.TMP  [jobname].LOG

$ !
$ !**** Go to the appropriate SUBMIT command
$ !**** AFTER_TIME = 'Y' and CPU_LIMIT = 'Y' are the defaults
$ !
$ SUBMIT_GOTO:
$       IF AFTER_TIME .EQS. 'N' .AND. CPU_LIMIT .EQS. 'N' THEN GOTO SUBMIT_4
$       IF AFTER_TIME .EQS. 'N' .AND. CPU_LIMIT .NES. 'N' THEN GOTO SUBMIT_3
$       IF AFTER_TIME .NES. 'N' .AND. CPU_LIMIT .EQS. 'N' THEN GOTO SUBMIT_2
$ !
$ !**** Submit job after 1700 with a CPU limit of 1 hour
$ !
$ SUBMIT_1:
$       SUBMIT/NOPRINTER/AFTER=17:00:00.00/CPUTIME=01:00:00.00-
                        /NAME='JOB_NAME' 'JOB_NAME'.COM
$       GOTO EXIT
$ !
$ !**** Submit the job after 1700 with no CPU limit
$ !
$ SUBMIT_2:
$       SUBMIT/NOPRINTER/AFTER=17:00:00.00/NAME='JOB_NAME' 'JOB_NAME'.COM
$       GOTO EXIT
$ !
$ !**** Submit the job now with a CPU limit of 1 hour
$ !
$ SUBMIT_3:
$       SUBMIT/NOPRINTER/CPUTIME=01:00:00.00/NAME='JOB_NAME' 'JOB_NAME'.COM
$       GOTO EXIT
$ !
$ !**** Submit the job now with no CPU limit
$ !
$ SUBMIT_4:
$       SUBMIT/NOPRINTER/NAME='JOB_NAME' 'JOB_NAME'.COM
$ EXIT:
```

TABLE A3(e) Listing of the BATCH command procedure (continued)

```
$ SET DEFAULT [.HASP.SIMUL]
$ DEFINE SYS$INPUT BATCHIN.TMP
$ DEFINE SYS$OUTPUT BATCH1OUT.TMP
$ COPY NEWDATA.DAT PARAM.DAT
$ RUN SIMPROG
$ DEASSIGN SYS$OUTPUT
$ DEASSIGN SYS$INPUT
$ DELETE BATCH1.COM;*
$ DELETE BATCH1IN.TMP;*
```

(a) Command file (jobname.COM)

```
1
0
2.5
0.01
2
0
3
```

(b) Input data (jobnameIN.TMP)

TABLE A.4 The two files generated by the interpreter (user defined variables underlined)

APPENDIX B - A SELECTION OF COMPONENT MODELS

B.1 AL2Z AND AL3Z ACTUATORS/MECHANICAL LINKAGES

Actuator with cap mechanism: AL2Z

100. In this section, the external mass will be termed "the cap". During extension and retraction of this actuator, the only significant external load is inertial provided the plane of motion of the mechanism is horizontal. However, because the motion of the cap is angular, its inertia will be experienced as a variable effective mass by the linear actuator. In addition, the actuator will experience a variable force due to the weight of the cap if the plane of the mechanism is inclined to the horizontal (see figure B1.1). This model assumes that the friction effect of the hinges is negligible and that the mass of the connecting link is negligible compared to the combined mass of the cap and the attached bellcrank. Variation of the frictional force due to varying side load on the piston is also neglected.

Model linking

101. AL2Z has two external links, two internal links and two signals and its linking diagram is shown in figure B1.2.

SUBROUTINE INPUTS:

P_P Piston pressure in bar

P_A Annulus pressure in bar

SUBROUTINE OUTPUTS:

Q_P Piston flow in l/s

Q_A Annulus flow in l/s

V_P Piston fluid volume in l

V_A Annulus fluid volume in l

x Displacement of the actuator rod in m

v Velocity of the actuator rod in m/s

F External force in N

m_e Effective mass in kg

User defined parameters

102. The following user defined parameters are common to both the actuator and load models AL2Z and AL3Z.

Piston diameter in cm

Rod diameter in cm

Stiction in N

Coulomb friction as a fraction of the stiction level

Viscous friction coefficient in N/(m/s)

Windage loss coefficient in $N/(m/s)^2$

Actuator stroke in m

Initial displacement of the actuator rod in m

Initial velocity of the actuator rod in m/s

Spring stiffness in N/m

Piston leakage coefficient in (l/s)/bar

103. For the cap actuator AL2Z, the following parameters are also

required.

Mass of the cap in kg

Angle of inclination of the actuator to the horizontal

Inertia of the cap in kg m^2

Length of the hinge link in m

Length of the actuator link in m

Initial longitudinal distance from the end of the actuator rod the the hinge centre in m

Longitudinal distance between the hinge centre and the centre of mass of the cap when the actuator is fully retracted in m

Transverse distance between the hinge centre and the centre of mass of the cap when the actuator is fully retracted in m

104. Nomenclature

F_{EXT} External force (positive if assisting extension) in N

h Transverse distance - hinge to the line of action of the actuator in m

J Inertia of the cap about its hinge in kg m^2

L_{CG} Distance from the hinge to the centre of mass of the cap in m

M Mass of the cap and its associated linkage in kg

M_e Effective mass of the cap as experienced by the actuator in kg

R Radius of the hinge arm in m

V Linear velocity of the actuator in m/s

x Displacement of the actuator rod in m

x_o Initial distance - hinge to the rod end in m

x_{CG} Initial longitudinal distance - hinge to the centre of mass of the cap in m

y Transverse distance - hinge to the centre of mass of the cap im m

y_{CG} Initial transverse distance - hinge to the centre of mass of the cap in m

γ Included angle of the bellcrank in radians

θ Angle of the bellcrank arm (actuator link) to the horizontal in radians

ω Angular velocity of the cap in rad/s

Model equations

105. The mathematical model incorporates two differential equations. The time derivative of the piston displacement is given by

$$\frac{dx}{dt} = V$$

The time derivative of the piston velocity is given by

(i) for $V \neq 0$

$$\frac{dV}{dt} = \frac{P_P A_P - P_A A_A - f_V V - f_\omega V^2 - k_S x - f_D (SIGN(V)) + F_{EXT}}{m_e}$$

(ii) for $V=0$ and $|P_P A_P - P_A A_A - k_S x + F_{EXT}| > |F_S|$

$$\frac{dV}{dt} = \frac{P_P A_P - P_A A_A - k_S x - F_D (SIGN(P_P A_P - P_A A_A - k_S x + F_{EXT})) + F_{EXT}}{m_e}$$

(iii) for $V=0$ and $|P_P A_P - P_A A_A - k_S x + F_{EXT}| \leq |F_S|$

$$\frac{dV}{dt} = 0$$

106. The flows into and out of the actuator are then calculated as shown below

$$Q_P = -VA_P - K_L(P_P - P_A) \text{ by convention}$$

and

$$Q_A = VA_A + K_L(P_P - P_A)$$

107. When the piston reaches an end stop, special modelling techniques are required. The integrator may fail if the time derivative of any state variable is discontinuous. Therefore, it is assumed that the piston on reaching an end stop is brought smoothly to rest over some finite but extremely small distance. In this region, the reaction force of the end stop on the piston is assumed to increase with respect to its displacement, thus allowing the force system to reach static equilibrium. Additionally, in this region the viscous friction coefficient is increased smoothly in order to dampen the motion of the piston.

108. The terms of external force and mass must be defined relative to the line of motion of the actuator rod. The external force on the actuator rod is calculated by performing a force balance on the bellcrank and the connecting rod. The calculated mass is variable with respect to piston displacement and is termed the referred or effective mass.

Calculation of the external force on the actuator

109. The external force experienced by the actuator is caused by the gravitational force of the cap and its associated mechanism. This is only relevant if the actuator is inclined to the horizontal. From figure B1.3 it can be seen that the gravitational force is given by

110. With reference to figure B1.4, if the centre of the cap and its mechanism is forward of the hinge, x_{CG} is defined as being positive, otherwise x_{CG} is negative.

111. The included angle of the bellcrank, γ , is given by,

$$\gamma = \sin^{-1}\left(\frac{x_o - L}{R}\right) + \tan^{-1}\left(\frac{x_{CG}}{y_{CG}}\right)$$

The angle between the axis of the piston and the smaller bellcrank arm, θ , is calculated as follows.

$$\theta_1 = \tan^{-1}\left(\frac{h}{x_o - x}\right)$$

$$L_{AC} = \frac{h}{\sin\theta_1}$$

Using the cosine rule,

$$\theta_2 = \cos^{-1}\left(\frac{L_{AC}^2 + R^2 - L^2}{2 L_{AC} R}\right)$$

$$\theta = \theta_1 + \theta_2$$

112. The radius of the centre of mass of the cap about the bellcrank

hinge, L_{CG} , is given by,

$$L_{CG} = \left(x_{CG}^2 + y_{CG}^2\right)^{\frac{1}{2}}$$

113. The transverse distance from the bellcrank hinge to the centre of mass of the cap, y, can now be calculated as

$$y = L_{CG} \sin\left(\pi - \theta - \gamma\right)$$

The distance h is constant and is given by

$$h = \left(R^2 - \left(x_o - L\right)^2\right)^{\frac{1}{2}}$$

114. Finally, taking moments about the bellcrank hinge and rearranging gives

$$F_{EXT} = -mg \sin\beta\left(\frac{y}{h}\right)$$

Calculation of the effective mass of the cap

115. The effective mass is calculated using the following energy equation

$$\frac{1}{2} m_e v^2 = \frac{1}{2} J \omega^2$$

or rearranging

$$m_e = J\left(\frac{\omega}{v}\right)^2 \qquad \text{(B1.1)}$$

with reference to figure B1.4,

$$R\sin\theta - L\sin\alpha = h$$

$$R\cos\theta + L\cos\alpha = x_o - x$$

Rearranging and squaring both sides gives,

$$L^2 \sin^2\alpha = (R\sin\theta - h)^2$$

and

$$L^2 \cos^2\alpha = ((x_o - x) - R\cos\theta)^2$$

Adding, noting $\cos^2\alpha + \sin^2\alpha = 1$ gives

$$L^2 = h^2 - 2Rh\sin\theta + (x_o - x)^2 - 2R(x_o - x)\cos\theta + R^2 \quad (B1.2)$$

Using the chain rule

i.e. $$\frac{dL^2}{dx} \cdot \frac{dx}{dt} + \frac{dL^2}{d\theta} \cdot \frac{d\theta}{dt} = 0$$

and rearranging

$$\frac{\omega}{v} = \frac{-\frac{dL^2}{dx}}{\frac{dL^2}{d\theta}} \quad (B1.3)$$

Differentiate equation B1.2 with respect to θ ,

$$\frac{dL^2}{d\theta} = -2Rh\cos\theta + 2R(x_o - x)\sin\theta \quad (B1.4)$$

Similarly,differentiate equation B1.2 with respect to x,

$$\frac{dL^2}{dx} = -2(x_o - x) + 2R\cos\theta \quad (B1.5)$$

Substituting equations B1.4 and B1.5 into equation B1.3 and then into B1.1 gives

$$m_e = J\left(\frac{2(x_o - x) - 2R\cos\theta}{2R(x_o - x)\sin\theta - 2Rh\cos\theta}\right)$$

Actuator with shutter mechanism: AL3Z

116. In this section, the external mass will be termed "the shutter". As in the case of the cap actuator described in paras.100 to 115, the shutter actuator experiences the inertia of the shutter as a variable effective mass. The force exerted on the actuator rod by the linkage is due to both a component of the weight of the shutter and an external load on the shutter. For ease of understanding, the initial theory is developed for an actuator mounted horizontally with the plane of the mechanism being vertical. The more complex cases of the actuator being mounted at an angle to the horizontal and/or the plane of the mechanism being at some angle to the horizontal are considered in paragraphs 130 to 135. Since the plane on motion of the shutter mechanism is not horizontal, a component of the weight of the shutter will be experienced by the actuator at some displacements. The external load must be defined as a function type (e.g. linear, sinusoidal) together with the end conditions of that function. Figure B1.5 shows the actuator, the shutter and its associated linkage. In writing this model, the same assumptions have been made as were made for AL2Z. Additionally, it is assumed that the distributed force on the shutter may be represented by a point load at a user defined centre of pressure.

Model linking

117. AL3Z has two external links, two internal links and two signals and its linking diagram is shown in figure B1.6.

SUBROUTINE INPUTS:

P_P Piston pressure in bar

P_A Annulus pressure in bar

SUBROUTINE OUTPUTS:

Q_P Piston flow in l/s

Q_A Annulus flow in l/s

V_P Piston fluid volume in l

V_A Annulus fluid volume in l

x Displacement of the actuator rod in m

v Velocity of the actuator rod in m/s

F_{EXT} External force in N

m_e Effective mass in kg

User defined parameters

118. These parameters are required in addition to those mentioned in 102.

Mass of the shutter in kg

Inertia of the shutter about its hinge in kg m^2

Length of the link connecting the rod to the bellcrank in m (AB)

Length of the lower bellcrank arm in m (BC)

Length of the upper bellcrank arm in m (CD)

Length of the mechanism/shutter connecting pin to the shutter hinge in m (EF)

Initial longitudinal distance - bellcrank hinge to the rod end in m

Longitudinal distance - bellcrank hinge to the shutter hinge in m

Vertical distance - bellcrank hinge to the shutter hinge in m

Included angle of the bellcrank in degrees

Initial longitudinal distance - shutter hinge to the centre of mass of the shutter in m

Initial vertical distance - shutter hinge to the centre of mass of the shutter in m

Angle of inclination of the pressure force to the horizontal in degrees

Initial longitudinal distance - shutter hinge to the centre of pressure in m

Initial vertical distance - shutter hinge to the centre of pressure in m

Form of the function defining the external pressure force with respect to the angular position of the shutter e.g. linear, sinusoidal, cubic.

External force on the shutter when horizontal in N

External force on the shutter when closed in N

Angle of the shutter to the horizontal when closed in degrees

Angle of the line EF to the horizontal when the shutter is closed in degrees

Orientation of the axis of the actuator and of the plane of the mechanism relative to the horizontal and vertical planes

Note: For sign convention refer to figure B1.9. Positive as shown.

119. Nomenclature

F_p External force on the shutter in N

F_n Force in N (see figures B1.11 to B1.23)

h_n, L_n, x_n Distances and lengths in mm (see figures B1.7 to B1.15)

J_c Effective inertia of the shutter about the bellcrank hinge in kgm^2

J_F Inertia of the shutter about its hinge in kg m

m_e Effective mass of the shutter relative to the actuator in kg

v Velocity of the actuator piston in m/s

x Displacement of the actuator piston in m

α Inclination of the external force to the horizontal in radians (positive upwards)

β Inclination of the actuator mounting plate to the horizontal in radians (positive upwards)

$\theta_n \psi$ Mechanism angles in radians (see figures B1.7 to B1.19)

ϕ Included angle of the bellcrank in radians

ω_c Angular velocity of the bellcrank in rad/s

ω_F Angular velocity of the shutter in rad/s

(n is a general subscript reference)

Model equations

120. As for the model AL2Z, the force applied to the piston rod by the external load and the effective mass of the shutter referred to

the piston rod must be calculated for all displacements of the rod.

(a) Calculation of lengths and angles required in the following analysis

121. (i) The angle between the axis of the piston and the smaller bellcrank arm, θ_{CB}, is calculated as follows:

Refer to figure B1.7

The distance h is constant and is given by

$$h = (L_2^2 - (x_o - L_1)^2)^{\frac{1}{2}}$$

$$\theta_{CB1} = \tan^{-1}\left(\frac{h}{x_o - x}\right)$$

$$L_{AC} = \frac{h}{\sin\theta_{CB1}}$$

Using the cosine rule,

$$\theta_{CB2} = \cos^{-1}\left(\frac{L_{AC}^2 + L_2^2 - L_1^2}{2L_{AC}L_2}\right)$$

$$\theta_{CB} = \theta_{CB1} + \theta_{CB2} \qquad (B1.6)$$

(ii) The angles θ_F, θ_C and θ_D are derived as follows:

With reference to figures B1.8 and B1.9,

$$x_{CF} = (L_{CF}^2 + h_{CF}^2)^{\frac{1}{2}}$$

$$\gamma_{CF} = \tan^{-1}\left(\frac{h_{CF}}{L_{CF}}\right)$$

Using the cosine rule

$$x_{DF} = (L_3^2 + x_{CF}^2 - 2L_3 x_{CF}\cos(\theta_C - \gamma_{CF}))^{\frac{1}{2}}$$

$$\gamma_{FE} = \cos^{-1}\left(\frac{x_{DF}^2 + L_5^2 - L_4^2}{2L_5 x_{DF}}\right)$$

$$\gamma_{FD} = \cos^{-1}\left(\frac{x_{DF}^2 + x_{CF}^2 - L_3^2}{2x_{DF}x_{CF}}\right)$$

$$\theta_F = \gamma_{FE} + \gamma_{FD} + \gamma_{CF}$$

Using the definition of θ_{cB} given in equation B1.6,

$$\theta_c = \phi - \theta_{cB}$$

$$h_{CF} = L_3 \sin\theta_c + L_4 \sin\theta_D - L_5 \sin\theta_F$$

$$\therefore \quad \theta_D = \sin^{-1}\left(\frac{h_{CF} - L_3 \sin\theta_c + L_5 \sin\theta_F}{L_4}\right)$$

(iii) The dimensions X_{CA} and X_{CP} are calculated as follows :

With reference to figure B1.10,

$$x_{CG} = (h_{GI}^2 + L_{GI}^2)^{\frac{1}{2}}$$

$$x_{CP} = (h_{PI}^2 + L_{PI}^2)^{\frac{1}{2}}$$

(iv) The angles θ_{GI} and θ_{PI} are calculated as follows:

With reference to figure B1.10,

$$\theta_{GI} = \tan^{-1}\left(\frac{h_{GI}}{L_{GI}}\right)$$

$$\theta_{PI} = \tan^{-1}\left(\frac{h_{PI}}{L_{PI}}\right)$$

(v) Figure B1.11 shows the shutter at some intermediate position. In order to calculate the reaction F_E on rod ED, it is necessary to calculate three moment arms, X_{FE} , X_{FP} , and X_{mg}.

With reference to figure B1.12,

$$\theta_{ED} = \frac{\pi}{2} - \theta_D$$

$$\theta_{EF} = \frac{\pi}{2} - \theta_F$$

$$\theta_E = \theta_{ED} - \theta_{EF}$$

$$x_{FE} = L_5 \sin\theta_E$$

With reference to figure B1.13,

$$\theta_{FP} = \theta_{PI} - (\theta_{FI} - \theta_F)$$

$$\psi_{FP} = \alpha + \theta_{FP}$$

$$x_{FP} = x_{cp} \sin\psi_{FP}$$

With reference to figure B1.14,

$$\theta_{CG} = \theta_{GI} - (\theta_{FI} - \theta_F)$$

$$\psi_{CG} = \left(\frac{\pi}{2} - \theta_{CG}\right) + \beta$$

$$x_{mg} = x_{CG} \sin\psi_{CG}$$

(vi) In order to calculate F_{EXT}, the moment arm of F_E about C must be known.

With reference to figure B1.15,

$$\psi_{CD} = \theta_D - \theta_C$$

$$x_{CF} = L_3 \sin(\theta_D - \theta_C)$$

(b)Calculation of the external force on the acutuator

122. The force on the shutter, F_p, is calculated using a function defined by the user. Four options are available:

(i) Linear (figure B1.16)

$$F_p = \left(\frac{F_2 - F_1}{\theta_{SI}}\right)\theta_S + F_1$$

(ii) Square Law (figure B1.17)

$$F_p = \left(\frac{F_2 - F_1}{\theta_{SI}}\right)\theta_S^2 + F_1$$

(iii) Cubic Law (figure B1.18)

$$F_p = \left(\frac{F_2 - F_1}{\theta_{sI}}\right)\theta_s^3 + F_1$$

(iv) Sinusoidal (figure B1.19)

$$F_p = \left(\frac{F_2 - F_1}{\sin\theta_{sI}}\right)\sin\theta_s + F_1$$

123. Referring to figure B1.11 and taking moments about F,

$$F_E = F_p x_{FP} + mg x_{mg}$$

124. Referring to figure B1.15 and taking moments about C,

$$F_{EXT} = \frac{x_{CF} F_E}{h}$$

(c) Calculation of the effective mass of the shutter referred to the actuator piston

125. The effective mass of the shutter relative to the line of action of the actuator must be calculated. This effective mass is a function of the inertia of the shutter about its hinge (F) and the geometry of the mechanism which changes with respect to actuator piston displacement, x.

126. In order to break the calculation down into equations of manageable proportions, the problem is considered in two sections. The effective inertia of the shutter about the bellcrank hinge (C) may be calculated using the energy relationship

$$\frac{1}{2} J_c \omega_c^2 = \frac{1}{2} J_F \omega_F^2$$

i.e. $$J_c = J_F\left(\frac{\omega_F}{\omega_c}\right)^2 \qquad (B1.7)$$

127. Similarly, the effective mass of the shutter relative to the actuator may be calculated by,

$$\frac{1}{2} M_A V^2 = \frac{1}{2} J_c \omega_c^2$$

i.e. $$M_A = J_c\left(\frac{\omega_c}{v}\right)^2$$

Calculation of M_A as a function of J_c

128. With reference to the calculation of the effective mass of the cap (paragraph 115) and figure B1.7,

$$m_A = J_c\left(\frac{2(x_o-x)-2L_2\cos\theta_{CB}}{2L_2(x_o-x)\sin\theta_{CB}-2L_2h\cos\theta_{CB}}\right)$$

Calculation of J_c as a function of J_F

129. With reference to figure B1.8,

$$L_{CF} = L_3\cos\theta_C + L_4\cos\theta_D + L_5\cos\theta_F$$

$$\cos\theta_D = \frac{L_{CF} - L_3\cos\theta_C - L_5\cos\theta_F}{L_4}$$

Similarly,

$$h_{CF} = L_3\sin\theta_C + L_4\sin\theta_D - L_5\sin\theta_F$$

$$\sin\theta_D = \frac{h_{CF} - L_3\sin\theta_C + L_5\sin\theta_F}{L_4}$$

Using the identity $\cos^2\theta_D + \sin^2\theta_D = 1$, and rearranging gives,

$$L_4^2 = \left(L_{CF} - L_3\cos\theta_C - L_5\cos\theta_F\right)^2 + \left(h_{CF} - L_3\sin\theta_C + L_5\sin\theta_F\right)^2 \qquad (B1.8)$$

Differentiating equation B1.8 with respect to time,

$$2\left(L_{CF}-L_3\cos\theta_c-L_5\cos\theta_F\right)\left(L_3\sin\theta_c\frac{d\theta_c}{dt}+L_5\sin\theta_F\frac{d\theta_F}{dt}\right)$$
$$+\,2\left(h_{CF}-L_3\sin\theta_c+L_5\sin\theta_F\right)\left(-L_3\cos\theta_c\frac{d\theta_c}{dt}+L_5\cos\theta_F\frac{d\theta_F}{dt}\right)=0$$

or rearranging,

$$\frac{d\theta_F}{dt}\,\frac{dt}{d\theta_c}=\frac{-L_3}{L_5}\left[\frac{\sin\theta_c\left(L_{CF}-L_5\cos\theta_F\right)-\cos\theta_c\left(h_{CF}+L_5\sin\theta_F\right)}{\sin\theta_F\left(L_{CF}-L_3\cos\theta_c\right)+\cos\theta_F\left(h_{CF}-L_3\sin\theta_c\right)}\right] \qquad (B1.9)$$

Combining equations B1.7 and B1.9 gives,

$$J_c=J_F\left(\frac{-L_3}{L_5}\left[\frac{\sin\theta_c\left(L_{CF}-L_5\cos\theta_F\right)-\cos\theta_c\left(h_{CF}+L_5\sin\theta_F\right)}{\sin\theta_F\left(L_{CF}-L_3\cos\theta_c\right)+\cos\theta_F\left(h_{CF}+L_3\sin\theta_c\right)}\right]\right)^2$$

130. Local to global system transformation

As previously stated, the theory covered in paragraphs 120 to 129 applies only to a shutter mechanism which is mounted horizontally (i.e. Its plane of motion is vertical).

131. In practice, these mechanisms may be divided into two categories. Those mounted horizontally and those mounted vertically. In both cases it is necessary to calculate the component of weight relevant to the local system described in paragraphs 120 to 129 and to modify the angle β.

(a) Mechanisms mounted horizontally

132. Suppose, the plane upon which the mechanism is mounted is at

some angle to the horizontal, β, and also that the actuator is mounted at some angle to the line of steepest descent of the plane upon which it is mounted, ξ, then β must be corrected as described below.

133. β' is the modified version of the angle β.

With reference to figures B1.20 and B1.21,

$$OC = r\cos\xi$$
$$AA' = r\sin\beta$$

By similar triangles,

$$\frac{CC'}{AA'} = \frac{OC}{OA} = \frac{r\cos\xi}{r}$$
$$BB' = CC' = r\sin\beta\cos\xi$$
$$\sin\beta' = \frac{BB'}{r} = \cos\xi\sin\beta$$
$$\beta' = \sin^{-1}(\cos\xi\sin\beta) \qquad (B1.10)$$

The angle β in the theory described in paragraphs 120 to 129 should be replaced by β' for the general case.

134. The weight acting in the plane of motion of the shutter mechanism is no longer simply mg but is a component of the total weight.

135. With reference to figure B1.22 and comparison with equation B1.10,

$$\gamma = \sin^{-1}\left(\cos\left(\frac{\pi}{2}-\xi\right)\sin\beta\right.$$

i.e.

$$\gamma = \sin^{-1}(\sin\xi \sin\beta)$$

Therefore

$$(mg)_{eff} = mg(\sin^{-1}(\sin\xi \sin\beta))$$

Similarly, mg in the theory described previously should be replaced by $(mg)_{eff}$ in the general case.

(b) Mechanisms mounted vertically

136. The angle β' in this case is simply given by,

$$\beta' = \frac{\pi}{2}$$

With reference to figure B1.23, the effective weight is given by,

$$(mg)_{eff} = mg \sin(\psi+\beta)$$

B.2 DC4Z DIRECTIONAL CONTROL VALVE

Introduction

200. DC4Z is an instantaneous model of a four way three position directional control valve. The configuration of the centre position is defined by the user during the parametric data input section. The user has thirteen configurations to choose from and these options are shown in figure B2.1 along with their code number.

201. The model does not account for dynamic behaviour of the valve. It is assumed that the valve is zero lapped.

Model linking

202. DC4Z has five external links and its linking diagram is shown in figure B2.2.

SUBROUTINE INPUTS:

P_S	Supply pressure in bar
P_A	Service A pressure in bar
P_B	Service B pressure in bar
P_R	Return pressure in bar
X_F	Fractional displacement of valve

SUBROUTINE OUTPUTS:

Q_S	Supply flow in l/s
Q_A	Service A flow in l/s
Q_B	Service B flow in l/s

Q_R Return flow in l/s

User defined parameters

203. Two modes of input exist for this model. These are,

a) To define 4 to 6 restriction constants for the 4 to 6 possible flow paths in (l/s)/bar$^{1/2}$

b) To define flows in l/s and corresponding pressure drops in bar for each of the flow paths.

204. Data for paths A to B and S to R will only be required if these paths exist in the particular configuration chosen. It should be noted that there is no direct path from S to R in a valve such as No.1 though the fluid may pass from S to A and from A to R. The flow path in a valve such as No.10 is achieved by passing the fluid through the centre of the valve spool.

205. Nomenclature

C Linear restriction constant in (l/s)/bar

C Restriction constant in (l/s)/bar$^{1/2}$

K_1, K_2, K_3 Flow factors

Q_A, Q_B, Q_R, Q_S Flows through the A,B,S and R ports in l/s

Q_{XY} Flow across path XY in l/s

Q_{XYI} User defined nominal flow in l/s

X_F Fractional displacement of valve

P_L Limit of linear pressure/flow characteristic in bar

P_{XY} Differential pressure across XY in bar

P_{XYI} Differential pressure corresponding to the user defined nominal flow in bar

Model Equations

206. If the first mode of input is chosen, the restriction constants of the relevant flow paths must be calculated.

$$C_{XY} = \frac{Q_{XYI}}{(\Delta P_{XYI})^{1/2}}$$

where XY denotes a particular flow path.

207. The pressure/flow characteristic for a particular flowpath through the valve is made up of the usual square law plus a linear region at low differential pressures.

208. If a flow was calculated by simply inputting a differential pressure into this characteristic, then this flow would only be correct when that path is fully open. Therefore, the flow must be multiplied by a flow factor to account for the gradual change in flow area as X_F varies as shown in figure B2.3.

If $X_F < -1$

$$k_1 = 0, \qquad k_2 = 0, \qquad k_3 = 1$$

If $-1 \leq X_F < 0$

$$k_1 = 0, \qquad k_2 = 1 + X_F, \qquad k_3 = -X_F$$

If $0 \leq X_F < 1$

$$k_1 = X_F \qquad k_2 = 1 - X_F, \qquad k_3 = 0$$

If $X_F \geqslant 1$

$$k_1 = 1, \qquad k_2 = 0, \qquad k_3 = 0$$

209. The discontinuities in flow factors at the three valve positions cause the flow to be discontinuous. Therefore, the time deerivative of pressure in an adjacent pipe model will also be discontinuous requiring a special modelling technique incorporating the use of a cubic polynomial. The method is shown below.

210. With reference to figure 2.4,

$$h = X_{F2} - X_{F1}$$

$$z = (X_F - X_{F1})/h$$

$$k_1 = -0.01z^3 + 0.02z^2$$

211. The linearised restriction constants are calculated as shown below

$$C_{LXY} = \frac{C_{XY}(\Delta P_L)^{1/2}}{\Delta P_L}$$

212. The actual flows may now be calculated using the information supplied by adjacent models.

If $|\Delta P_{XY}| < \Delta P_2$

$$Q_{XY1} = k_1 C_{LXY} \Delta P_{XY}$$

$$Q_{XY2} = k_2 C_{LXY} \Delta P_{XY}$$

$$Q_{XY3} = k_3 C_{LXY} \Delta P_{XY}$$

If $|\Delta P_{xy}| \geq \Delta P_L$

$$Q_{xy1} = k_1 C_{xy} (|\Delta P_{xy}|)^{1/2} SIGN(\Delta P_{xy})$$

$$Q_{xy2} = k_2 C_{xy} (|\Delta P_{xy}|)^{1/2} SIGN(\Delta P_{xy})$$

$$Q_{xy3} = k_3 C_{xy} (|\Delta P_{xy}|)^{1/2} SIGN(\Delta P_{xy})$$

213. If a particular flow path does not exist for the configuration being considered, then that flow is set to zero.

214. These flows may now be summed to give the nett flows at the four ports as shown below.

$$Q_S = -Q_{SA1} - Q_{SA2} - Q_{SB2} - Q_{SR2} - Q_{SB3}$$

$$Q_A = Q_{SA1} + Q_{SA2} + Q_{AB2} - Q_{AR2} - Q_{AR3}$$

$$Q_B = -Q_{BR1} + Q_{SB2} + Q_{AB2} - Q_{BR2} + Q_{SB3}$$

$$Q_R = Q_{BR1} + Q_{SR2} + Q_{AR2} + Q_{BR2} + Q_{AR3}$$

B.3 FC8Z PRESSURE COMPENSATED FLOW CONTROL VALVE WITH A REVERSE FLOW CHECK VALVE

Introduction

300. PC8Z models the dynamic and steady state behaviour of a pressure compensated flow control valve with a fixed orifice bypass. The instantaneous behaviour of a reverse free flow check valve operating in parallel with the flow control valve is also included. The ISO symbol for such a valve is shown in figure B3.1.

Model Assumptions

301. Dynamic behaviour of a pressure compensated flow control valve is relevant only when the compensating orifice is open. The differential equation defining the transient flow is therefore employed only when the differential pressure is above the minimum controlling pressure and is below the maximum working pressure of the valve. The dynamic behaviour of the valve is considered to be adequately represented by a first order differential equation. The dynamic behaviour of the reverse free flow check valve is not accounted for.

Model linking

302. FC8Z both have 2 external links and one internal link and their linking diagram is shown in figure B3.2.

SUBROUTINE INPUTS:

P_{IN} Inlet pressure in bar

P_{OUT} Outlet pressure in bar

SUBROUTINE OUTPUTS:

Q_{IN} Inlet flow in l/s

Q_{OUT} Outlet flow in l/s

Q_T Transient flow in l/s

303. User defined parameters

Minimum controlling pressure in bar

Maximum controlling pressure in bar

Nominal flow setting in l/s

Bypass restriction constant in (l/s)/bar$^{1/2}$

Main valve restriction constant for reverse flow in (l/s)/bar$^{1/2}$

Pressure at which compensating orifice closes in bar

Check valve cracking pressure in bar

Differential pressure at which saturation flow exists in the check valve in bar

Corresponding flow through check valve in l/s

Coefficient A of differential equation in (l/s)/bar

Coefficient B of differential equation in s^{-1}

Initial transient flow in l/s

Initial differential pressure in bar.

304. Nomenclature

A Coefficient in differential equation in (l/s)/bar

B Coefficient in differential equation in s

K Transient flow factor

P_{IN} Inlet pressure in bar

P_{OUT} Outlet pressure in bar

Q_{CV} Flow through the check valve in l/s

Q_{FV} Flow through the flow control valve in l/s

Q_{IN} Net inlet flow in l/s

Q_{OR} Flow through the bypass orifice in l/s

Q_{OUT} Net outlet flow in l/s

Q_{SET} Set flow in l/s

Q_T Transient flow in l/s

t Time in s

t_p Response time to reach peak transient flow in s

$t_{0.05}$ Response time to reach set flow ± 5% measured from peak transient flow in s

ΔP Differential pressure across the valve in bar

ΔP_{CL} Differential pressure at which compensating orifice closes in bar

ΔP_{MAX} Maximum controlling pressure in bar

ΔP_{MIN} Minimum controlling pressure in bar

n Subscript to signify current calculation step

n-1 Subscript to signify previous completed calculation step

Model equations

The differential equation

305. The model initially calculates the instantaneous outlet flow, Q_{OUT}, based upon the differential pressure across the valve. The

model actually calculates the algebraic sum of three flows i.e. the flow through the main valve and the flow through the bypass orifice (figure B3.3) and the flow through the check valve (figure B3.4). Due to the large number of cubic smoothing regions, the program automatically derives the cubic function required.

Region 1: Reverse flow $(\Delta P < -0.01 \text{ bar})$

$$Q_{CR} = 0$$

$$Q_{FV} = -C_1|\Delta P|^{1/2} \qquad (B3.1)$$

Region 2: Cubic smoothing region $(-0.01 \leq \Delta P < 0)$

$$Q_{CR} = 0$$

306. The values of the function (F) at the boundaries of Region 2 must be calculated.

$$F_1 = -C_1|0.01|^{1/2}$$

$$F_2 = 0$$

307. Also the gradients (G) of the function at the boundaries must be calculated.

Differentiating equation B3.1,

$$\frac{dQ_{FV}}{d\Delta P} = \frac{-C_1}{2|\Delta P|^{1/2}}$$

Therefore,

$$G_1 = \frac{-C_1}{2|0.01|^{1/2}}$$

$$G_2 = 0$$

Region 3: Cubic smoothing region $(0 \leq \Delta P < 0.01)$

$$Q_{OR} = 0$$

Again, the function and its derivatives at the boundaries must be calculated.

$$F_1 = 0$$

$$F_2 = 0.01\, C_4$$

where

$$C_4 = \frac{Q_{NOM}}{\Delta P_{MIN}}$$

$$G_1 = 0$$

$$G_2 = C_4$$

Region 4: Coming onto control $(0.01 \leq \Delta P < \Delta P_{MIN} - 0.01)$

$$Q_{OR} = 0$$

$$Q_{FV} = C_4 \Delta P$$

Region 5: Cubic smoothing region $(\Delta P_{MIN} - 0.01 \leq \Delta P < \Delta P_{MIN})$

$$Q_{OR} = 0$$

$$F_1 = C_4(\Delta P_{MIN} - 0.01)$$

$$F_2 = Q_{NOM}$$

$$G_1 = C_4$$

$$G_2 = C_6$$

where

$$C_6 = \frac{0.02\, Q_{NOM}}{\Delta P_{MAX} - \Delta P_{MIN}}$$

Region 6: Controlling $(\Delta P_{MIN} \leq \Delta P < \Delta P_{MAX})$

$$Q_{OR} = 0$$

$$Q_{FV} = C_6 (\Delta P - \Delta P_{MIN}) + Q_{NOM}$$

Region 7: High ΔP droop $(\Delta P_{MAX} \leq \Delta P < \Delta P_{CL})$

308. Due to the absence of any better information, it is assumed that the flow through the main section of the valve reduces as a cubic law at differential pressures higher than the maximum controlling pressure. Similarly, the flow through the bypass orifice increases as a cubic law.

(a) Flow through main valve:- The calculation may be treated as a cubic smoothing region.

$$F_1 = 1.02\, Q_{NOM}$$

$$F_2 = 0$$

$$G_1 = C_6$$

$$G_2 = 0$$

(b) Flow through bypass orifice:-

$$F_1 = 0$$

$$F_2 = C_8 \Delta P_{CL}^{1/2}$$

$$G_1 = 0$$

Differentiating equation B3.2 with respect to ΔP

$$\frac{dQ_{OR}}{d\Delta P} = \frac{C_8}{2\Delta P^{1/2}}$$

i.e.

$$G_2 = \frac{C_8}{2\Delta P^{1/2}}$$

Region 8: Compensating orifice closed $(\Delta P \geq \Delta P_{CL})$

$$Q_{OR} = C_8 \Delta P^{1/2} \qquad (B3.2)$$

$$Q_{FV} = 0$$

Region 9: Saturation flow $(\Delta P < \Delta P_{CS} - 0.01)$

$$Q_{FV} = -C_9 |\Delta P|^{1/2}$$

Region 10: Cubic smoothing region $(\Delta P_{CS} - 0.01 \leq \Delta P < \Delta P_{CS})$

$$F_1 = -C_9 |\Delta P_{CS} - 0.01|^{1/2}$$

$$F_2 = -C_9 |\Delta P_{CS}|^{1/2}$$

$$G_1 = \frac{-C_9}{2|\Delta P|^{1/2}}$$

$$G_2 = C_{11}$$

Region 11: Check valve linear characteristic $(\Delta P_{cs} \leq \Delta P < \Delta P_{cc} - 0.01)$

$$Q_{cv} = C_{11} (\Delta P - \Delta P_{cc})$$

Region 12: Cubic smoothing region $(\Delta P_{cc} - 0.01 \leq \Delta P < \Delta P_{cc})$

$$F_1 = 0.01\, C_{11}$$

$$F_2 = 0$$

$$G_1 = C_4$$

$$G_2 = 0$$

Region 13: Check valve inoperative $(\Delta P \geq \Delta P_{cc})$

$$Q_{cv} = 0$$

309. Having calculated the instantaneous value of the constituent parts of the outlet flow, it is corrected by adding a transient flow Q_T. However, the sign of Q_T is dependent upon whether the differential pressure is increasing or decreasing. Consider a pressure compensated flow control valve controlling the outlet flow in steady state. The valve is then subjected to a step increase in differential pressure. Initially, the flow through the valve will be significantly greater than the set value when the compensating spool takes up its new position.

310. Conversely, if the valve is subjected to a step decrease in

differential pressure, the flow through the valve will initially be significantly lower than the set flow.

311. The differential equation is given by

$$\frac{dQ_T}{dt} = -A\,\frac{(\Delta P_n - \Delta P_{n-1})}{(t_n - t_{n-1})} - B\,Q_T$$

The term $\Delta P - \Delta P_{n-1}$ defines the change in differential pressure across the valve during the time interval $t_n - t_{n-1}$. The initial condition of the state variable Q_T must be defined by the user. However, since the previous value of differential pressure, P_{n-1} , is also required in order to calculate the derivative Q_T , an initial differential pressure must also be defined by the user.

312. Care must be taken when defining this initial differential pressure to ensure that the value typed in is in fact the true differential pressure defined by the surrounding models. The program cannot automatically check the user's input.

313. As stated in paragraph 301, the transient flow need only be accounted for in regions 6 or 7. The transient flow is calculated for all regions but is, multiplied by a factor K. this factor is unity in region 6 and is cubically smoothed to zero in regions 5 and 7. Figure B3.5 shows a plot of K against differential pressure.

Regions 1 to 4

$$k = 0$$

Region 5

$$z = \frac{(\Delta P - \Delta P_{MIN} - \Delta P_5)}{\Delta P_5}$$

$$k = -2z^3 + 3z^2$$

Region 6

$$k = 1$$

Region 7

$$z = \frac{(\Delta P - \Delta P_{MAX})}{\Delta P_7}$$

$$k = 2z^3 - 3z^2 + 1$$

Region 8

$$k = 0$$

314. The net outlet flow is given by,

$$Q_{OUT} = Q_{FV} + Q_{OR} + Q_{CV} - kQ_T$$

By convention, the inlet flow is given by,

$$Q_{IN} = -Q_{OUT}$$

The Coefficients A and B

315. Figure B3.6 is a carpet plot from which the coefficients A and B may be found knowing the dynamic response of the valve. In order to derive this plot, the flow control valve was subjected to a ramped

increase in differential pressure from 0 to 120 bar in 10 ms. The user must know the peak flow when the valve is subjected to a differential pressure ramp of this kind. He must also know the time taken before this transient flow decreases to within 5% of the set flow (see figure B3.7). Setting the ordinate of the graph to the transient flow and the abscissa of the graph to the response time, the user may simply interpolate between the contours of the coefficients A and B. These values of A and B are required during the input stage of the simulation.

Note

316. The response time $t_{0.05}$ is measured from the peak of Q_T and not from the initiation of the ramp. Table B.1 shows the variation in time taken to reach the peak transient flow measured from the initiation of the ramp with respect to the coefficient B. It should be noted that these response times are accurate to approximately ± 0.5 ms. The time interval t_p is independent of variation in coefficient A within the limits of accuracy stated above.

B (s^{-1})	t (ms)
1000	7
800	7
600	8
400	8
200	9
100	10
50	10

TABLE B.1

B.4 GE1Z SYNCHRONOUS ELECTRIC MOTOR

Introduction

400. Subroutine GE1Z models the dyanamic behaviour of an electrical synchronous motor/generator.

401. Mathematically, it would not be a difficult task to account for all the abnormal modes of behaviour a synchronous generator may exhibit. However, the parametric data defining a complete mathematical model would be difficult to obtain. Therefore, certain simplifying assumptions have been made. Parameters relevant to these assumptions are displayed to the user on two internal links. Should the values of these parameters exceed specified limits, then the results obtained from a simulation using this model should be viewed with some scepticism. This point is discussed more fully in paragraphs 409 to 421.

Model assumptions

402. It is assumed that the generator is of a non-salient design, i.e. the armature is cylindrical (see paragraphs 411 to 416).

403. It is also assumed that the magnitude opf the torque created by the damper windings increases linearly with respect to rotor slip which is an adequate assumption over the normal working range [26] - see paragraph 417 for the definition of rotor slip.

404. Model linking

SUBROUTINE INPUTS:

T_m - Driving torque in Nm

SUBROUTINE OUTPUTS:

w_G - Generator speed in rev/min.

θ - Rotor lead/lag angle in radians.

T_E - Electrical torque neglecting damping in Nm.

S_r - Rotor slip

T_D - Damping torque in Nm.

405. The model has one external link and two internal links and its linking diagram is shown in figure B4.1.

406. User defined parameters

Synchronous speed of generator in rev/min.

Initial speed of generator in rev/min.

Electrical torque constant in Nm

Frictional torque constant in Nm (min/rev).

Moment of inertia of generator rotor in kg m^2.

Damping torque constant in Nm.

Initial rotor lead angle in radians.

407. Nomenclature

J Moment of inertia of generator rotor in kg m^2.

K_1 Electrical torque constant in Nm.

K_2 Damping torque constant in Nm (min/rev).

K_3 Frictional torque constant in Nm (min/rev).

S_r Rotor slip (see paragraph 415 for definition).

T_D Damping torque in Nm.

T_E Electrical torque neglecting damping in Nm.

T_{EL} Net electrical torque in Nm.

T_f Frictional torque in Nm.

T_m Driving torque in Nm.

ε_T Accelerating torque in Nm.

ε_ω Speed error in rev/min.

θ Rotor lead/lag angle (leading - positive, generator mode) in radians.

w_a Rotor speed in rev/min.

w_s Synchronous speed in rev/min.

Model equations

408. Figure 4.7 in the main text shows a block diagram of the method used to calculate w given the driving torque and other constants. The calculation is considered in three sections.

409. Calculation of the electrical torque generated neglecting the effect of damper windings

Figure B4.2 shows the variation of electrical torque, T_E, with respect to rotor lead/lag angle, θ, for a non-salient and a salient machine.

410. If the machine were of a salient design, i.e. one with an armature of a dumbell type cross section, the non-salient function shown in figure B4.2 would no longer be a sine curve and would peak at approximately $\pi/6$ radians. In this case, the model may still be

used provided θ is always small and K_1 is factored by 1.5 to account for the different function. To summarise, this model should only be used to simulate a salient generator when the inaccuracies mentioned above are deemed acceptable. For the remainder of this section, it will be assumed that the generator is non-salient.

411. If θ becomes greater than $\pi/4$ radians, then loss of synchronisation will occur. When a synchronous machine pulls out of step with its synchronous system, damage to both the machine and to service units may occur. Damage to the generator may be in the form of overheating of the rotor core or damper windings, or of mechanical damage. Loss of synchronism may occur due to:

(1) An applied mechanical torque in excess of K_1 ,

(2) An abnormally low field current, or

(3) A decrease in applied voltage.

412. For this reason, relays are often incorporated in generators to signal an alarm some time before this undesirable condition is reached [27].

413. Therefore, attention should be given to the absolute value of θ (displayed on an internal link) to ensure that it does not approach 0.75 radians (0.5 radians for salient machines).

414. The speed error, ε_ω, is given by,

$$\varepsilon_\omega = \omega_s - \omega_G$$

The rotor lead angle, θ , is calculated using the differential

equation,

$$\frac{d\theta}{dt} = \frac{2\pi \varepsilon_\omega}{60}$$

The electrical torque is given by,

$$T_E = k_1 \sin(2\theta)$$

415. Calculation of the damping torque created by the damper windings

Figure B4.3 shows the variation of damping torque with respect to rotor slip, S_r , where S_r is defined as

$$S_r = 1 - \frac{\omega_G}{\omega_s}$$

416. The function shown in figure B4.3 is approximately linear for values of S_r between approximately -0.06 and 0.06 [27]. These limits of linearity are in fact greater if the electrical resistance of the damper winding is high. Rotor slip is also displayed on an internal link and should be checked following every simulation in order to ensure the assumption of linearity holds good. If the assumption does not hold, then the results of the simulation will be erroneously overdamped.

417. The damping torque, T_D , is given by

$$T_D = k_2 S_r$$

418. Calculation of the generator speed, ω_G

The net error in torque (i.e. the torque tending to accelerate or retard the generator) is given by,

$$\varepsilon_T = (T_E + T_D) - T_m - T_F$$

419. Therefore, the generator speed, w_G, may be calculated using the differential equation

$$\frac{dw_G}{dt} = \frac{60\,\varepsilon_T}{2\pi J}$$

B.5 PCDZ METER-IN PRESSURE COMPENSATOR

Introduction

500. PCDZ models the instantaneous behaviour of a meter-in pressure compensator such as the Rexroth ZDC compensator. Such compensators are intended for use with proportional valves in order to maintain a constant pressure drop across the valve, thus ensuring constant flow for a given proportional valve spool displacement. Figure B5.1 shows the standard use of the compensator.

Model assumptions

501. It is assumed that the frictional forces on the valve can be neglected. Furthermore, both cross-port and external leakages are neglected.

Model linking

502. The inputs and outputs of PCDZ are listed below.

SUBROUTINE INPUTS

- P_p Supply pressure in bar
- P_1 Service pressure in bar
- P_T Return pressure in bar
- P_x External pilot pressure in bar

SUBROUTINE OUTPUTS

- Q_p Supply flow in l/s
- Q_1 Service flow in l/s
- Q_T Return flow in l/s

Q_x External pilot flow in l/s

503. The model has four external links and its linking diagram is shown in figure B5.2. Figure B5.3 shows the complete linking diagram corresponding to figure B5.1.

504. User defined parameters

Spool diameter in mm

Spool displacement when supply port closes in mm

Spool displacement when service port and return line port become connected in mm

Discharge coefficient of orifices

Spring rate in N/m

Spring preload force in N

505. Nomenclature

A Flow area in m

C_d Discharge coefficient

d Spool diameter in m

F_{mp} Momentum force on spool due to flow from supply to service in N

F_{mT} Momentum force on spool due to flow from service to return in N

F_p Force on spool due to differential pressure in N

F_{PRE} Preload force in spring in N

F_S Force on spool due to spring in N

k A combination of terms applicable to rectangular ports in m ($2C_d\pi d\cos\theta$)

k_S Spring rate in N/m

l_1 Spool displacement when supply port closes in m

l_T Spool displacement when service port and return line port become connected in m

P_1 Service pressure in bar

P_P Supply pressure in bar

P_T Return pressure in bar

P_X External pilot pressure in bar

Q Flow through orifice in m^3/s

Q_1 Servive flow in l/s

Q_P Supply flow in l/s

Q_T Return flow in l/s

Q_X External pilot flow in l/s

x Spool displacement in m

ΔP Differential pressure across orifice in N/m^2

θ Flow angle of orifice in degrees

ρ Fluid density in kg/m^3

Model equations

506. Figure B5.4 is a schematic of the compensator. It is shown in a condition when there is no flow through the valve and no pressure differential across the spool. The spool displacement x is defined as being zero in this condition. Therefore, at some displacement of the spool, the area of the metering orifice on the pressure port is given by

$$A = \pi d(l_1 - x) \quad \text{if } l_1 > x$$

Similarly, the area of the metering orifice on the return line port is given by

$$A = \pi d(x - l_T) \quad \text{if} \quad x > l_T$$

507. The spool displacement is calculated assuming that the spool is in static equilibrium. However, the forces acting on the spool are dependent upon displacement not only in magnitude, but also in the form of the expression.

508. Three possible flow configurations exist:

- Supply port connected to service port
- Service port and return line port closed
- Service port connected to return line port

509. Initially, x is calculated assuming that no momentum forces exist, i.e. the second case above. If the value of x thus obtained defines the spool to be in a different flow configuration, then the displacement is recalculated accounting for the momentum force(s) relevant to that region. This process is repeated until the spool displacement obtained corresponds to the configuration defining the expression.

510. The derivation of the forces acting on the spool are shown below. Forces tending to compress the spring are defined as being positive.

(a) Pressure force

$$F_p = (P_i - P_x)\frac{\pi d^2}{4}$$

(b) Spring force

$$F_s = -F_{PRE} - k_s x$$

(c) Momentum force

In general, the momentum force is given by

$$F = \rho Q V \cos\theta$$

where

$$Q = C_d A \left(\frac{2|\Delta P|}{\rho}\right)^{1/2}$$

and

$$V = \left(\frac{2|\Delta P|}{\rho}\right)^{1/2}$$

Therefore

$$F = 2C_d A |\Delta P| \cos\theta$$

Let

$$k = 2C_d \pi d \cos\theta \times 10^{5}$$

Therefore, the momentum force caused by flow through the supply port is given by

$$F_{mp} = k(l_1 - x).|P_p - P_1|$$

Similarly, the momentum force caused by flow through the return line port is given by

$$F_{MT} = -k(x-l_T)\cdot|P_1-P_T|$$

Each configuration is dealt with separately.

Case 1 $x < l_1$, $x < l_T$

For static equilibrium,

$$F_P + F_S + F_{MP} = 0$$

i.e.

$$F_P - F_{PRE} - k_S x + k(l_1-x)|P_P-P_1| = 0$$

Rearranging gives

$$x = \frac{F_P - F_{PRE} + kl_1|P_P-P_1|}{k_S + k|P_P-P_1|}$$

Case 2 $l_1 < x$, $x < l_T$

$$F_P + F_S = 0$$

i.e.

$$F_P - F_{PRE} - k_S x = 0$$

Rearranging gives

$$x = \frac{F_P - F_{PRE}}{k_s}$$

Case 3 $x > l_1$, $x > l_T$

$$F_P + F_s + F_{MT} = 0$$

i.e.

$$F_P - F_{PRE} - k_s x - k(x - l_T)|P_1 - P_T|$$

Rearranging gives

$$x = \frac{F_P - F_{PRE} + k l_T |P_1 - P_T|}{k_s + k|P_1 - P_T|}$$

511. The derivation of the flows is given below.

The flow through the metering orifice on the pressure port is given by

$$Q = -C_d \pi d (l_1 - x) \left(\frac{2|P_P - P_1| \times 10^5}{\rho} \right)^{1/2} SIGN(P_P - P_1) \times 10^3$$

provided $l_1 > x$

Similarly, the flow through the metering orifice on the return line

port is given by

$$Q_T = C_d \pi d (x - l_T) \left(\frac{2|P_1 - P_T| \times 10^5}{\rho} \right)^{1/2} SIGN(P_1 - P_T) \times 10^3$$

The flow through the service port is given by

$$Q_1 = Q_P - Q_T$$

The external pilot is used only to sense pressure. Therefore,

$$Q_x = 0$$

B.6 PM3Z DIESEL ENGINE

Introduction

600. Subroutine PM3Z models the dynamic behaviour of a diesel engine.

601. The model accounts for the speed droop characteristic and linearised maximum torque and motoring characteristcs. The governor speed is set by an adjacent model and PM3Z supplies speed, torque and acceleration as information to be used by an electronic control circuit. This information is supplied on signals. Therefore, any combination of of these three feedback variables may be used. However, since the information on signals is not stored as results data, the same information is available on an external link.

Model Assumptions

602. The characteristics representing maximum torque, speed droop and motoring are assumed to be linear.

Model linking

603. PM3Z has two external links, one internal link and three signals and its linking diagram is shown in figure B6.1.

SUBROUTINE INPUTS:

T_H hydraulic torque in Nm

ω_G governor setting in rev/min

SUBROUTINE OUTPUTS:

ω_E angular velocity in rev/min

T_E engine torque in Nm (signal 1)

ω_E angular velocity in rad/sec (signal 2)

$\dot{\omega}_E$ angular acceleration in rad/sec^2 (signal 3)

User defined parameters

604. Reference speed at zero torque for which the engine torque/speed characacteristic is known in rev/min

Torque at which the maximum torque characteristic intersects the speed droop characteristic in Nm

Torque at which the motoring characteristic intersects the speed droop characteristic in Nm

Gradient of the maximum torque characteristic in (Nm min)/rev

Gradient of the speed droop characteristic in (Nm min)/rev

Gradient of the motoring characteristic in (Nm min)/rev

Initial speed in rev/min

Moment of inertia of engine in kg m^2

Model equations

605. Figure B6.2 shows a block diagram representation of the method used to calculate the engine speed .

606. Since the model is dynamic, there must be an initial condition for ω_E. Knowing ω_E, the engine torque T_E may be found using the user defined speed/torque characteristic as shown in figure B6.3. This characteristic consists of three main operating regions. There are also two transition regions. The equations defining T_E are given below.

Region 1 Maximum torque characteristic $(\omega_E < \omega_T)$

The maximum torque characteristic is assumed to be linear. Thus, the torque is defined as

$$T_E = T_o + k_2 \omega_E$$

Region 2 Upper smoothing region $(\omega_T \leq \omega_E < \omega_T + \omega_{cs})$

The maximum torque and the speed droop characteristics are connected by a cubic polynomial over a small range of speed. This cubic polynomial is set up such that the resulting characteristic is continuous in both function and derivative.

$$\omega_T = \omega_o - \frac{(k_1 \omega_{DR} - T_{DR})}{k_1 - k_2}$$

$$z = (\omega_E - \omega_T) / \omega_{cs}$$

$$T_E = \omega_{cs}\left((3k_1+k_2)z^3 - (4k_1+2k_2)z^2 + k_2 z\right) + k_1(\omega_o - \omega_T)$$

Region 3 Speed droop characteristic $(\omega_T + \omega_{cs} \leq \omega_E < \omega_M)$

The speed droop characteristic of the diesel engine is defined as

$$T_E = k_1(\omega_o - \omega_E)$$

Region 4 Lower smoothing region $(\omega_M \leq \omega_E < \omega_M + \omega_{cs})$

The speed droop and the motoring characteristics are smoothed in a similar manner to that described for region 2.

Region 5 Motoring $(\omega_M + \omega_{cs} \leq \omega_E)$

The motoring characteristic is assumed to be linear and is defined as

$$T_E = k_3 \omega_E + T_1$$

607. ω_E is now calculated using the following differential equation which accounts for the inertial effects of the engine.

$$\frac{d\omega_E}{dt} = \frac{60\,\varepsilon_T}{2\pi J}$$

where

$$\varepsilon_T = T_E - T_P$$

B.7 PU0Z DIESEL ENGINE/HYDRAULIC PUMP

Introduction

700. Subroutine PU0Z models the characteristics of a fixed displacement hydraulic pump and a diesel engine exhibiting a linear speed droop with increasing torque and a linear torque droop with increasing speed. The model is instantaneous.

Model assumptions

701. The coefficients for slip, compressibility, and frictional lossed are assumed to be constant throughout the pump operating range.

702. Both the torque droop and the speed droop are assumed to be linear.

Model linking

703. The inputs and outputs of subroutine PU0Z are listed below.

SUBROUTINE INPUTS:

P_{IN} - Inlet pressure in bar

P_{OUT} - Outlet pressure in bar

SUBROUTINE OUTPUTS:

Q_{IN} - Inlet flow in l/s

Q_{OUT} - Outlet flow in l/s

T_P - Pump torque in Nm

w_P - Pump speed in rev/min

η_m - Mechanical efficiency

η_v - Volumetric efficiency

704. The model has two external links and two internal links and its linking diagram is shown in figure B7.1.

705. User defined parameters

Type of pump - gear or piston unit

Displacement of pump in l/rev

Swash fraction

Clearance volume as a fraction of full displacement (advice given)

Overall slip loss coefficient (advice given)

Speed dependent viscous torque loss coefficient (advice given)

Overall pressure dependent torque loss coefficient (advice given)

Maximum engine torque in Nm

Torque fall off with naximum speed in Nm

Maximum engine speed in rev/min

Governor speed droop at maximum torque in %

Moment of inertia in kg m^2

706. Nomenclature

C_F Speed dependent viscous torque loss coefficient

C_P Pressure dependent torque los coefficient

C_S Slip loss coefficient

D_P Pump displacement in l/rev

K_1 Speed droop constant in rad/Nms

K_2 Torque droop constant in rad/Nms

T_o Net torque at zero speed in Nm

T_{12}, T_{23}, T_{34}, T_{45} Torques at region boundaries in Nm

T_{CS} Cubic smoothing region in Nm

T_{DR} Torque droop at maximum speed in Nm

T_P Net pump torque in Nm

T_{PT} Pressure dependent torque loss in Nm

T_{TH} Theoretical pump torque in Nm

T_{VT} Viscous torque loss in Nm

V_c Clearance volume in l

X_{SW} Swash fraction (fraction of full displacement)

Z A variable in the cubic smoothing analysis

μ Fluid viscosity in Ns/m

η_m Mechanical efficiency

η_v Volumetric efficiency

w_o Pump speed at zero torque in rad/sec

w_{DR} Speed droop at maximum torque in rad/sec

w_p Pump speed in rad/sec

w_{p_n}, $w_{p_{n+1}}$ Succesive values of w_p in Newton-Raphson calculation

Model equations

707. Figure B6.3 shows the speed/torque characteristic for a typical engine.

708. However, in order to find w_o , it is not possible merely to calculate T_p then read w_p from the characteristic. The reason for this is that T_p is dependent upon the viscous torque loss which is in turn dependent upon w_p , i.e. the relationship between T_p and w_p is implicit. Figure B6.3 also shows that the characteristic is split into five regions. A cubic polynomial is fitted to regions 2 and 4 in order that the characteristic may be made continuous.

709. Figure B7.2 is a flow diagram which shows the method adopted for determing T_p and w_p .

710. Region 1

From figure B6.3,

$$\omega_p = \omega_o - k_1 T_p \qquad (B7.1)$$

The net torque required to drive the pump is given by

$$T_p = T_{TH} + T_{VT} + T_{PT} \qquad (B7.2)$$

The viscous torque loss is given by

$$T_{VT} = C_F \mu D_p \omega_p \qquad (B7.3)$$

Equations B7.1 and B7.3 combine to give

$$\omega_p = \frac{\omega_o - k_1 (T_{TH} + T_{PT})}{1 + k_1 C_F \mu D_p}$$

T_p is then calculated using equation B7.2 and B7.3 above.

711. Region 2

In this region, the normal method for smoothing discontinuous functions is employed. However, the implicit cubic polynomial obtained must be solved for w_p using an iterative method. The Newton- Raphson method for solving differentiable polynomials is used.

712. The lower boundary of the cubically smoothed region is given by,

$$T_{12} = T_o - \left(\frac{k_2 T_{DR} - \omega_{DR}}{k_2 - k_1}\right)$$

713. The smoothing polynomial is derived in the standard manner (Appendix D).

$$F_1 = k_2 (T_o - T_{12})$$
$$z = (T_p - T_{12})/T_{cs}$$
$$Q_1 = -k_1 T_{cs}$$
$$A = T_{cs}(k_2 - k_1)$$
$$B = 2T_{cs}(k_1 - k_2)$$

w_p may be expressed as,

$$\omega_p = Az^3 + Bz^2 + Q_1 z + F_1 \qquad (B7.4)$$

However, Z is a function of W_p,

$$z = \frac{T_p - T_{12}}{T_{cs}} = \frac{T_{TH} + T_{pT} + C_F \mu D_p \omega_p - T_{12}}{T_{cs}}$$

Therefore, rearranging equation B7.4,

$$f(\omega_p) = Az^3 + Bz^2 + Q_1 z + F_1 - \omega_p = 0 \quad (B7.5)$$

Differentiating Z with respect to w_p,

$$\frac{dz}{d\omega_p} = \frac{C_F \mu D_p}{T_{cs}}$$

Differentiating $f(w_p)$ (equation B7.5) with respect to w_p,

$$f'(\omega_p) = \frac{C_F \mu D_p}{T_{cs}} (3Az^3 + 2Bz + Q_1) - 1$$

714. Using the Newton-Raphson iterative formula, w_p is found to an acceptable degree of accuracy (1×10^{-5} rev/min).

$$\omega_{p_{n+1}} = \omega_{p_n} - \frac{f(\omega_{p_n})}{f'(\omega_{p_n})}$$

715. T_p is again calculated using equations B7.2 and B7.3. It may be found that T_p actually lies outside region 2 and reassessment of w_p may be necessary (see figure B7.2).

716. Region 3

The calculations in region 3 are simialr to those in region 1. From figure B6.3,

$$\omega_p = K_2 T_o - K_2 T_p \quad (B7.6)$$

Combining equations B7.2, B7.3 and B7.6,

$$\omega_p = \frac{k_2 T_o - k_2 (T_{TH} + T_{PT})}{(1 + k_2 C_F \mu D_p)}$$

717. Region 4

The calculations in region 4 are similar to those in region 2.

The lower boundary of the cubically smoothed region is given by,

$$T_{34} = T_o - T_{cs}$$

$$F_1 = k_2 T_{cs}$$

$$z = (T_{TH} + T_{PT} + C_F \mu D_p \omega_p) / T_{cs}$$

$$Q_1 = -k_2 T_{cs}$$

$$A = k_2 T_{cs}$$

$$B = -k_2 T_{cs}$$

giving

$$\omega_p = Az^3 + Bz^2 + Q_1 z + F_1 \qquad (B7.7)$$

Since $A = -B = -Q_1 = F_1$, rearranging B7.7 gives,

$$f(\omega_p) = A(z^3 - z^2 - z + 1) - \omega_p$$

Differentiating,

$$f'(\omega_p) = \frac{C_F \mu D_p A}{T_{cs}} (3z^2 - 2z - 1) - 1$$

The Newton-Raphson iterative method is again used to calculate w .

718. Region 5

In this region, $w_p = 0$

Therefore,

$$T_p = T_{TH} + T_{PT}$$

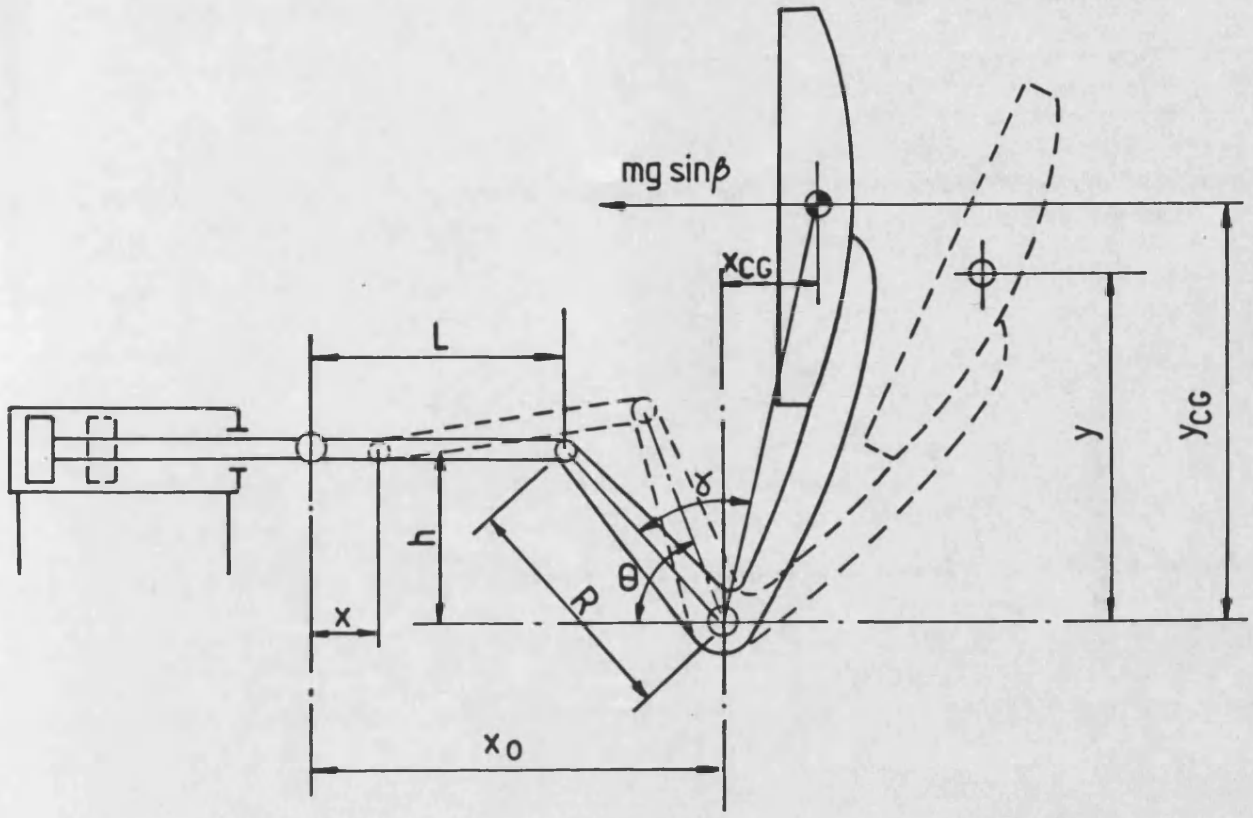
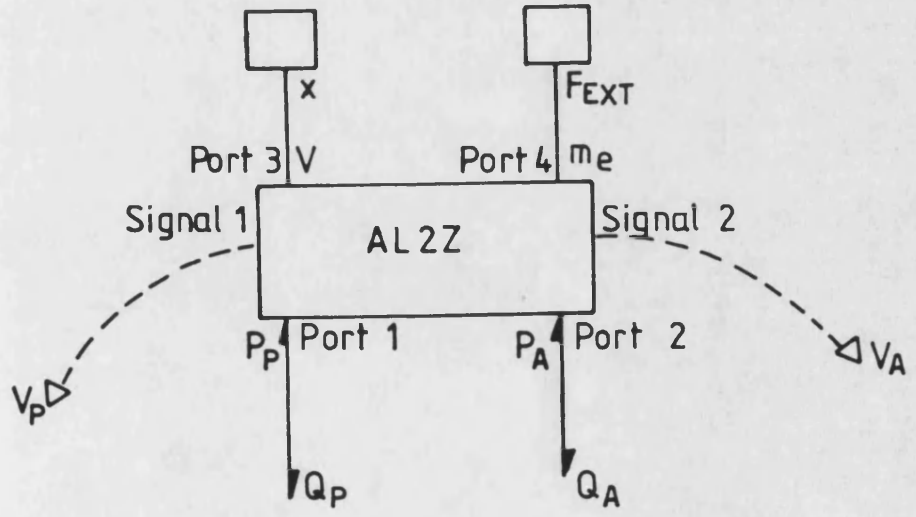


FIG. B1.1 The cap actuator and associated mechanism



EFFORTS : P_P P_A x F_{EXT}
FLOWS : Q_P Q_A v m_E
STATE VARIABLES : x v

FIG. B1.2 Linking diagram of AL2Z

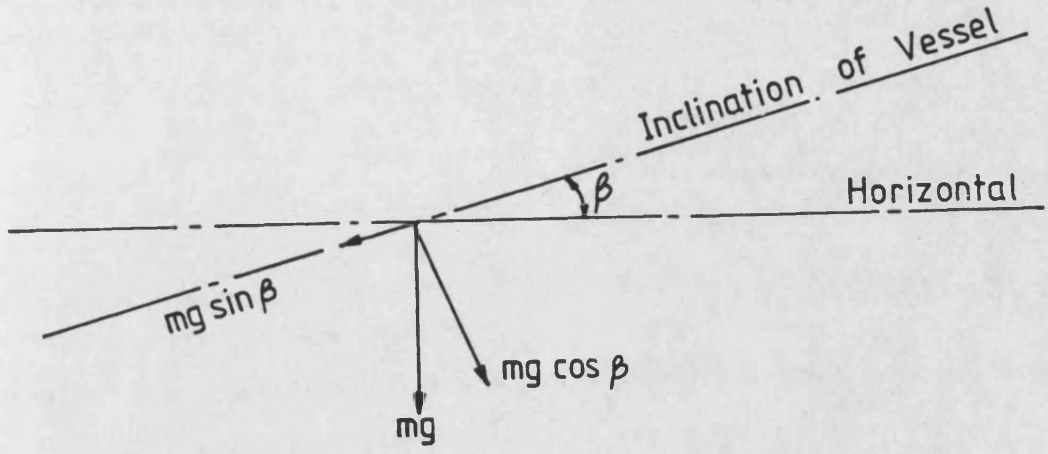


FIG. B1.3 Resolution of the weight of the cap

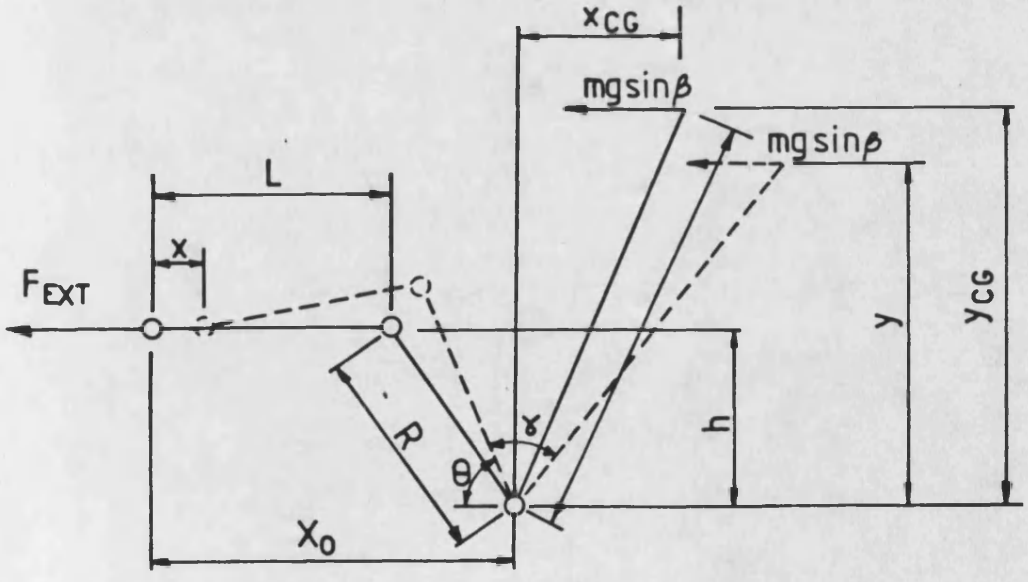


FIG. B1.4 Geometry of the cap mechanism

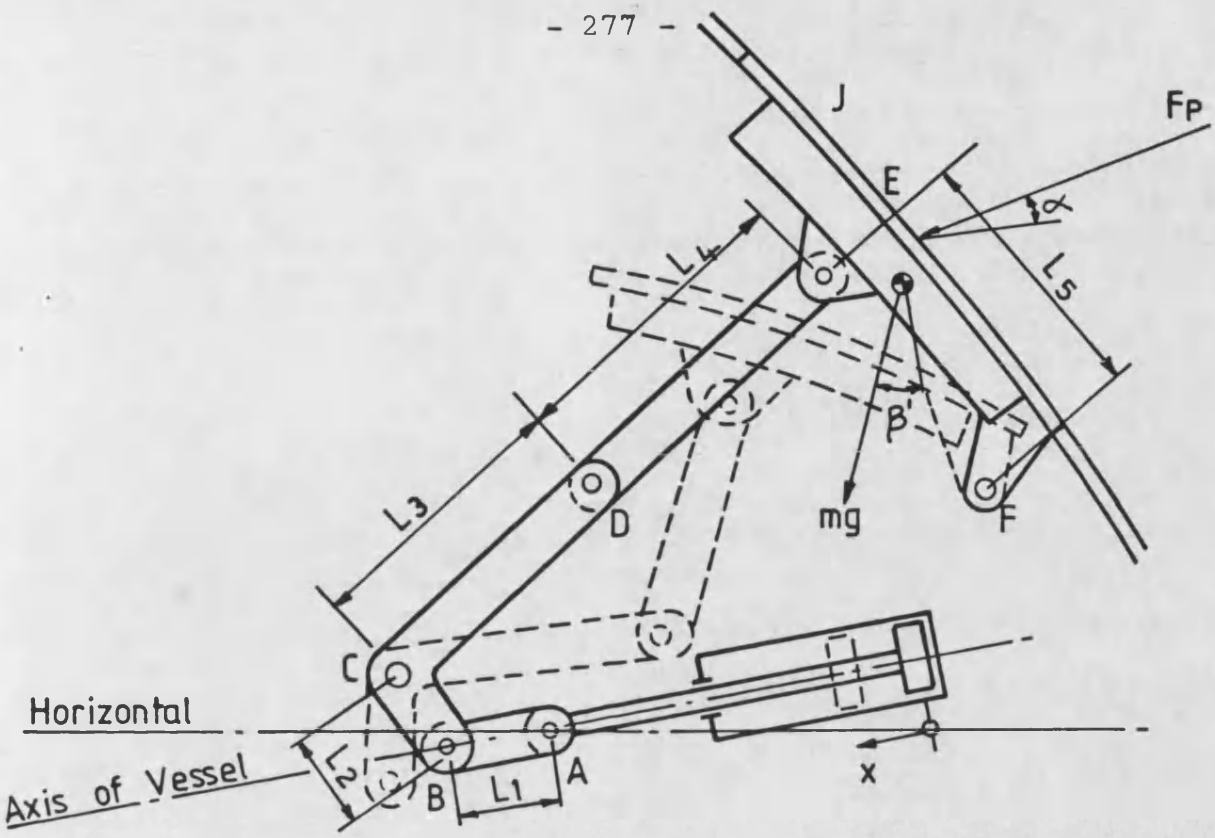
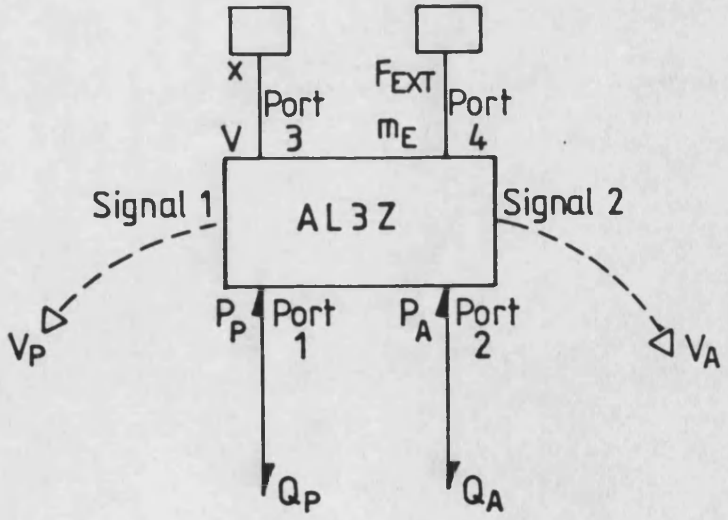


FIG. B1.5 The shutter actuator and associated mechanism



EFFORTS : $P_P P_A \times F_{EXT}$
FLOWS : $Q_P Q_A V m_E$
STATE VARIABLES : x V

FIG. B1.6 Linking diagram of AL3Z

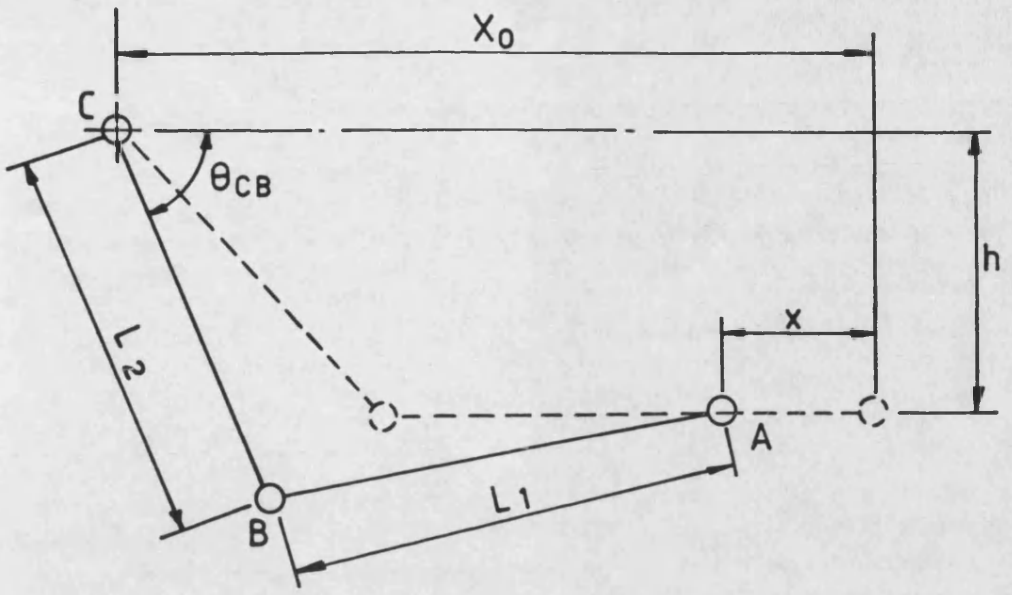


FIG. B1.7 Shutter mechanism geometry - Actuator rod to bellcrank centre

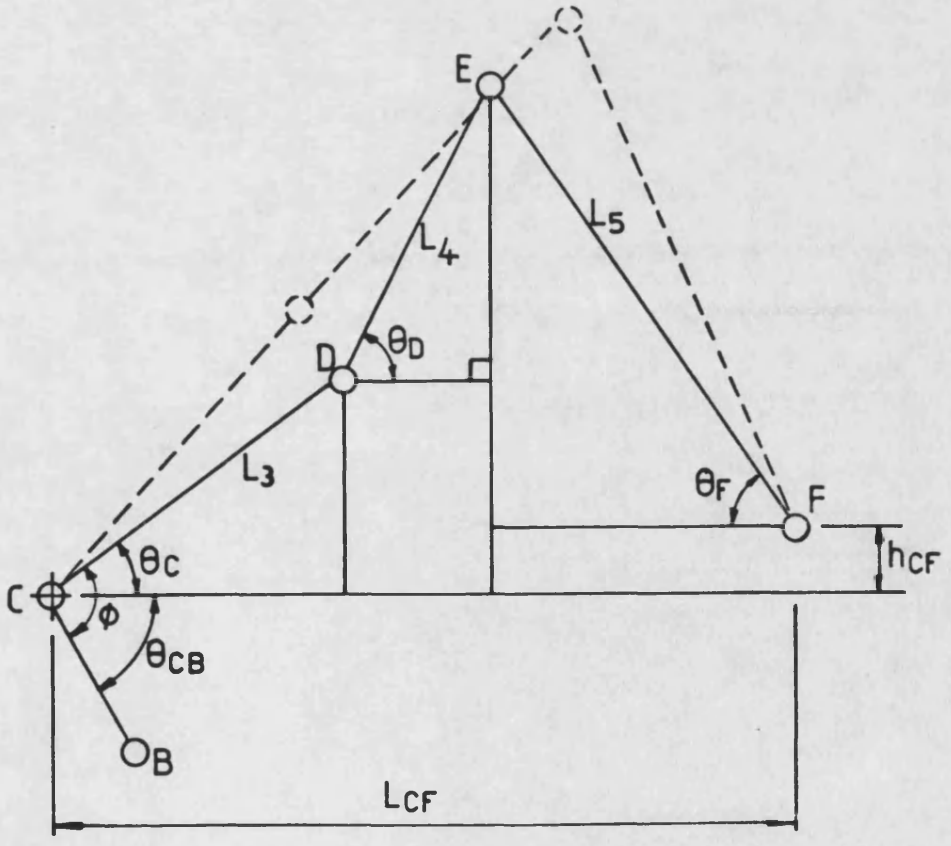


FIG. B1.8 Shutter mechanism geometry - Bellcrank centre to shutter hinge

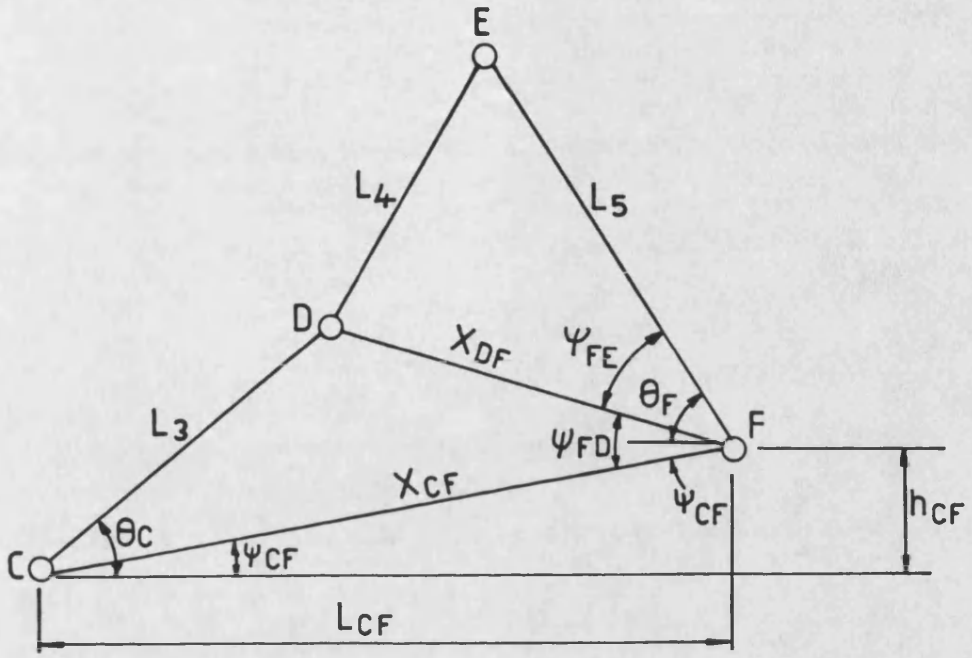


FIG. B1.9 Shutter mechanism geometry - Angle definition, bellcrank to hinge

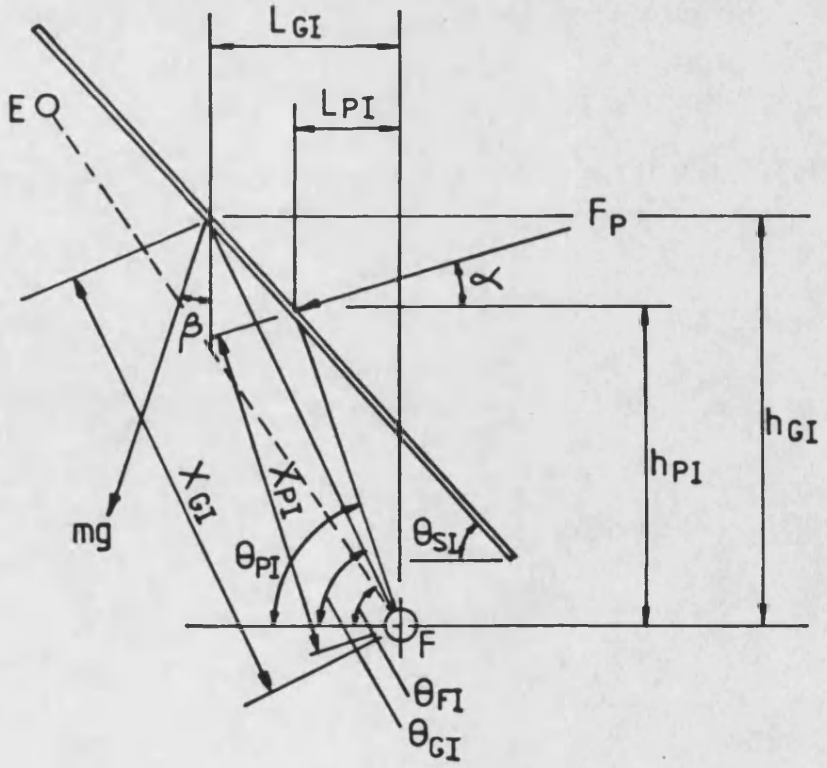


FIG. B1.10 Shutter mechanism geometry - Definition of load on shutter

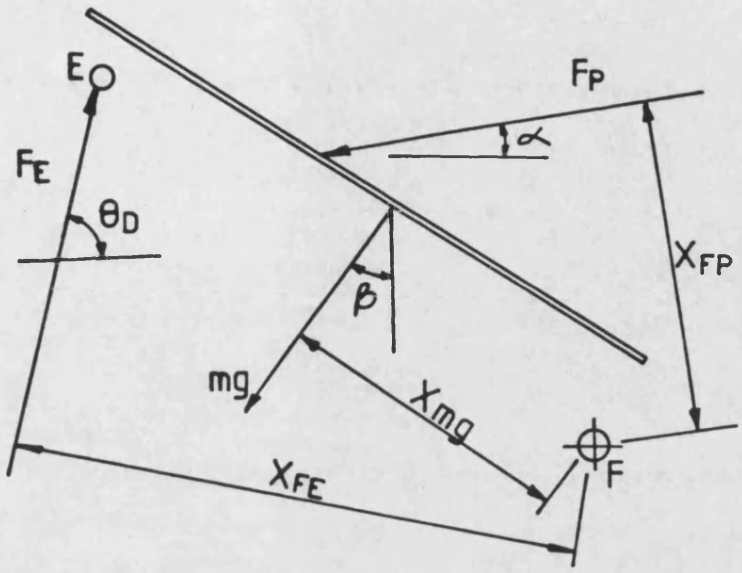


FIG. B1.11 Shutter mechanism geometry - Moment balance on shutter hinge

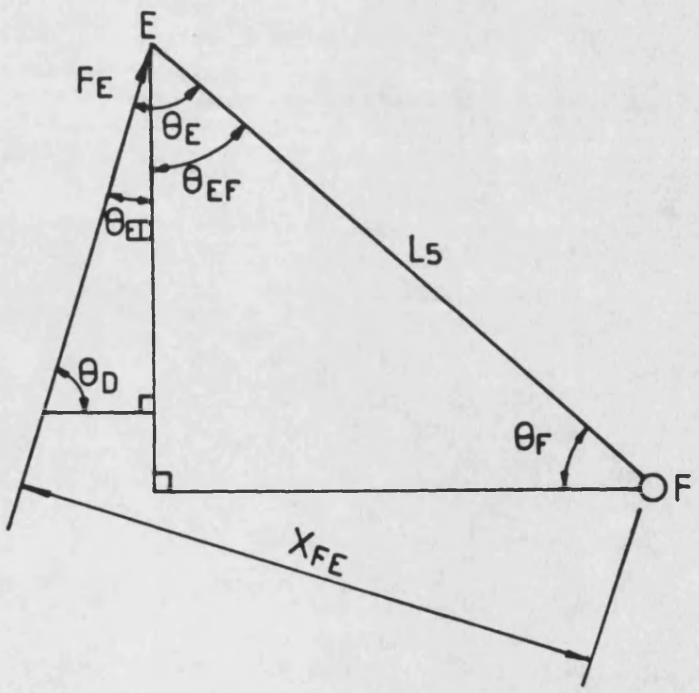


FIG. B1.12 Shutter mechanism geometry - Force of mechanism on shutter

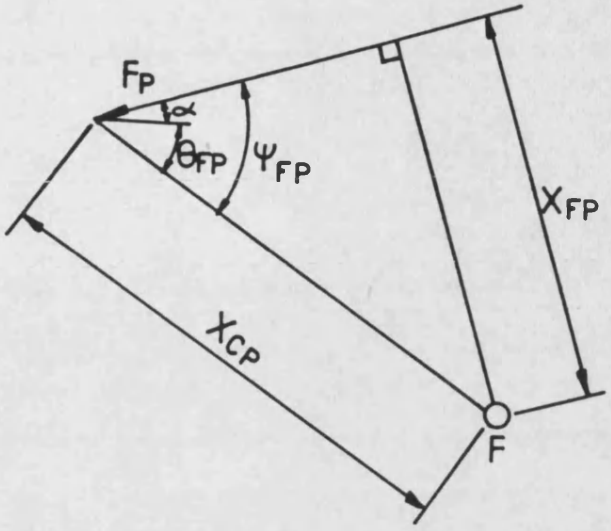


FIG. B1.13 Shutter mechanism geometry - External force on shutter

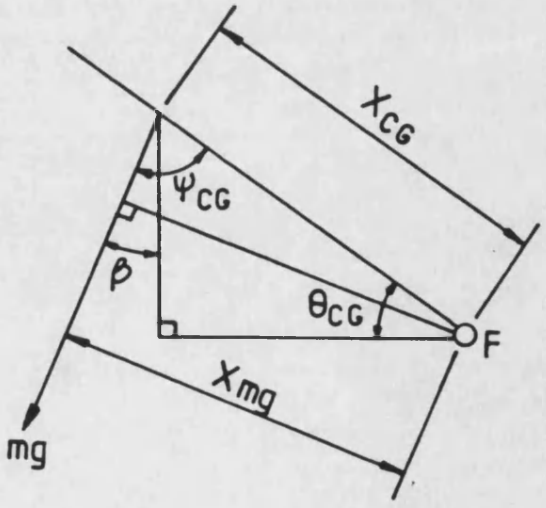


FIG. B1.14 Shutter mechanism geometry - Weight of shutter

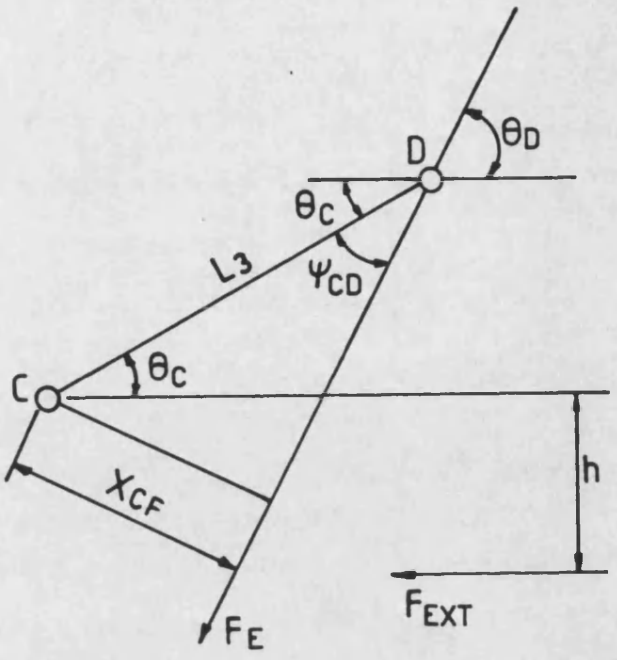


FIG. B1.15 Shutter mechanism geometry - Moment balance on bellcrank pivot

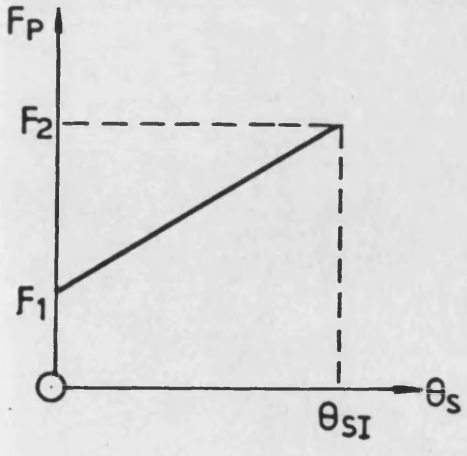


FIG. B1.16 Linear load

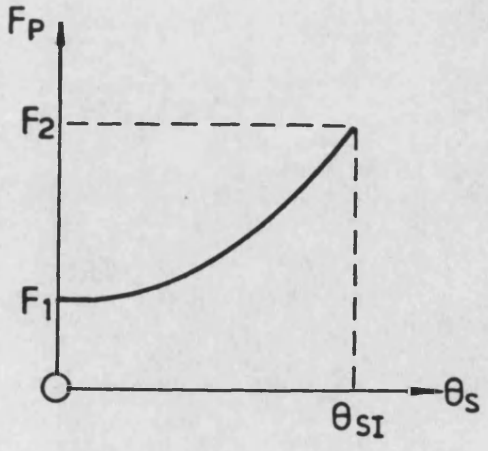


FIG. B1.17 Square load

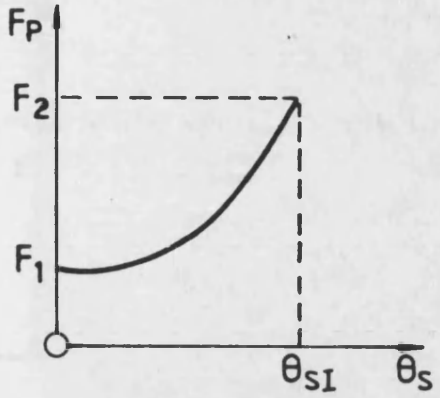


FIG. B1.18 Cubic load

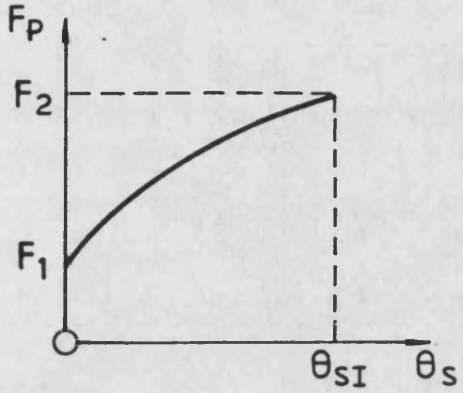


FIG. B1.19 Sinusoidal load

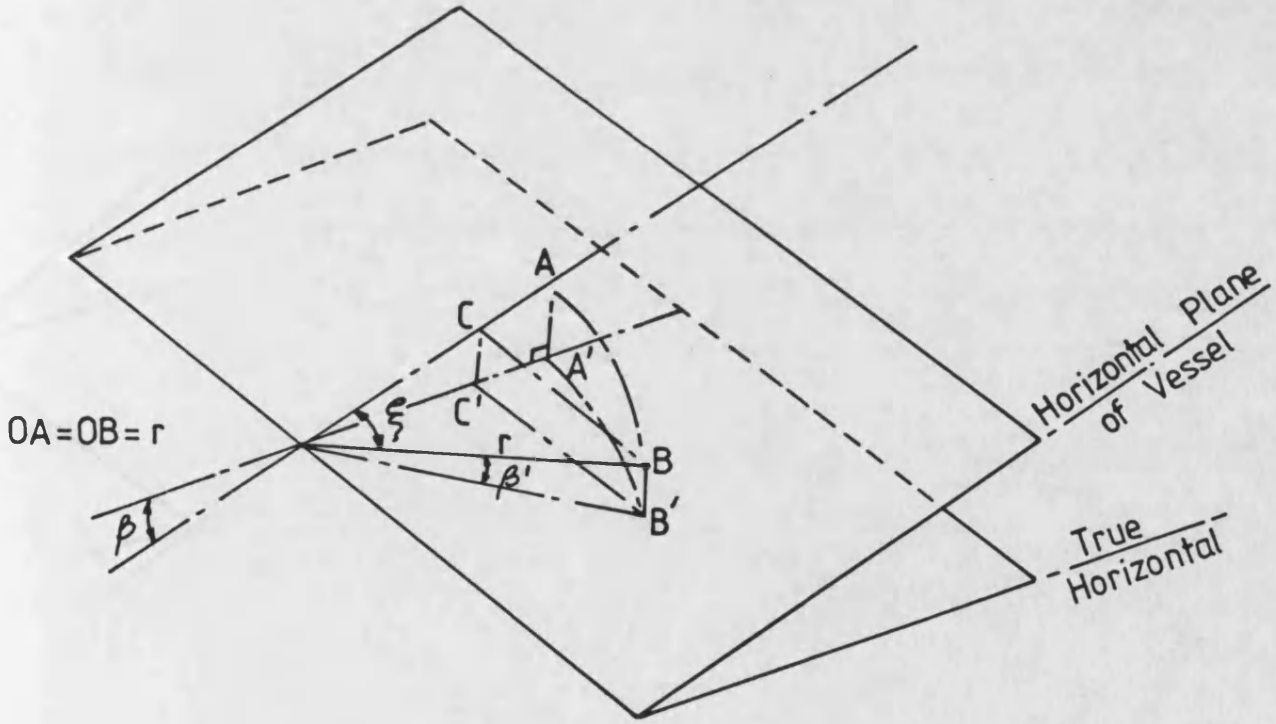


FIG. B1.20 Shutter mechanism geometry - Definition of angles of inclination

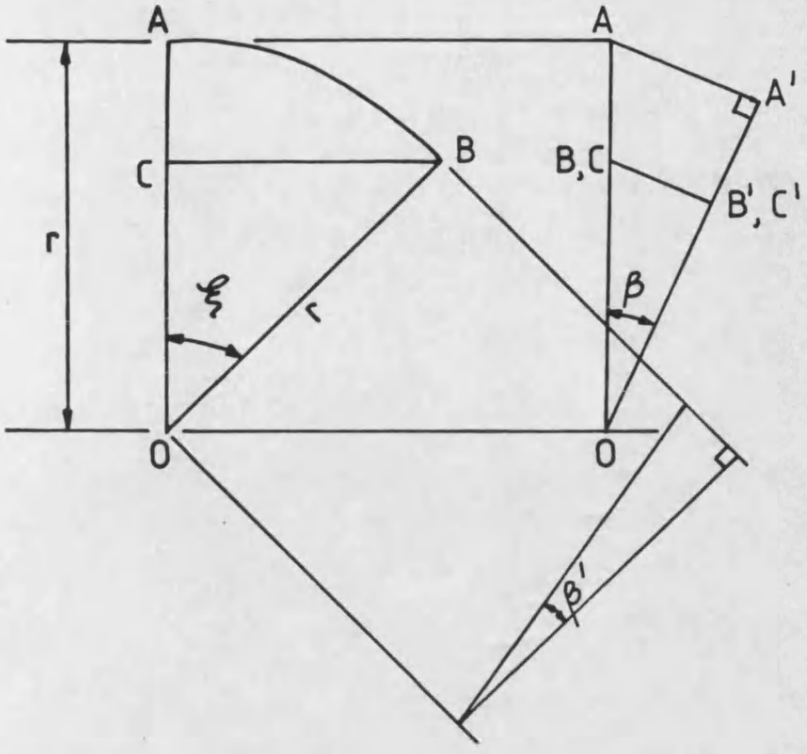
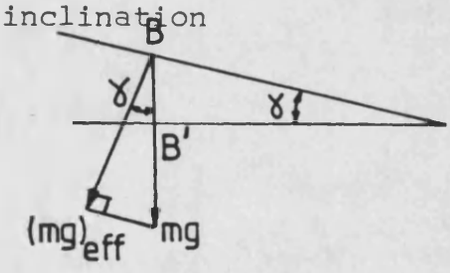


FIG. B1.21 Shutter mechanism geometry -
Resolution of angles of inclination



Resolution of weight

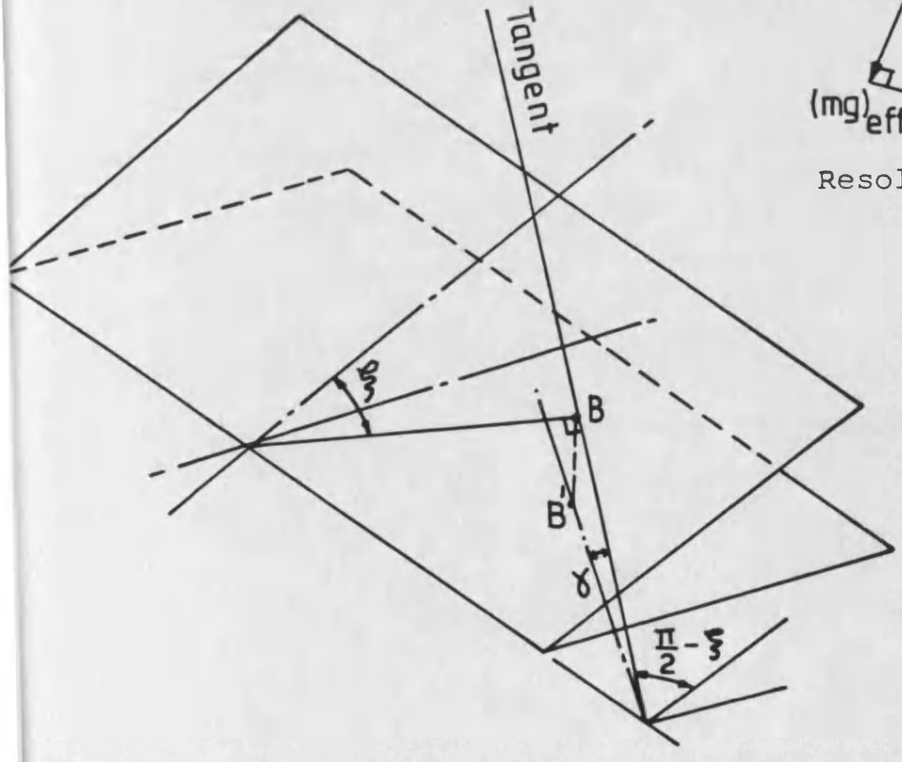


FIG. B1.22 Shutter mechanism geometry
Definition of angles of inclination

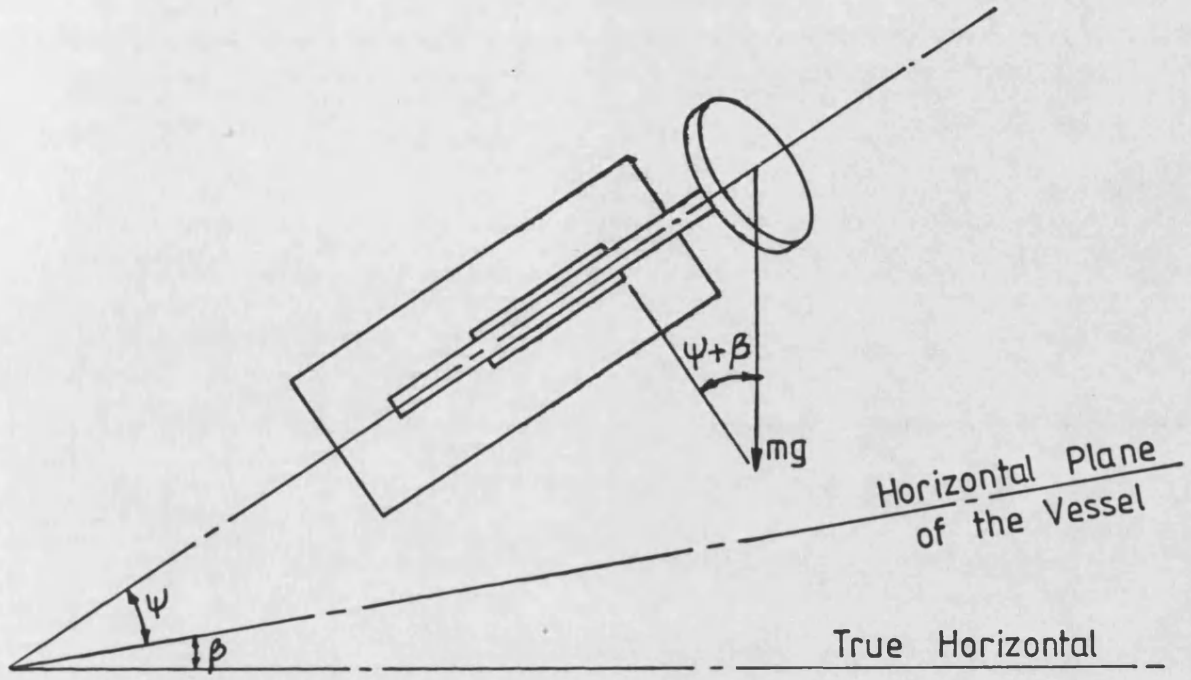


FIG. B1.23 Shutter mechanism geometry - Resolution of weight

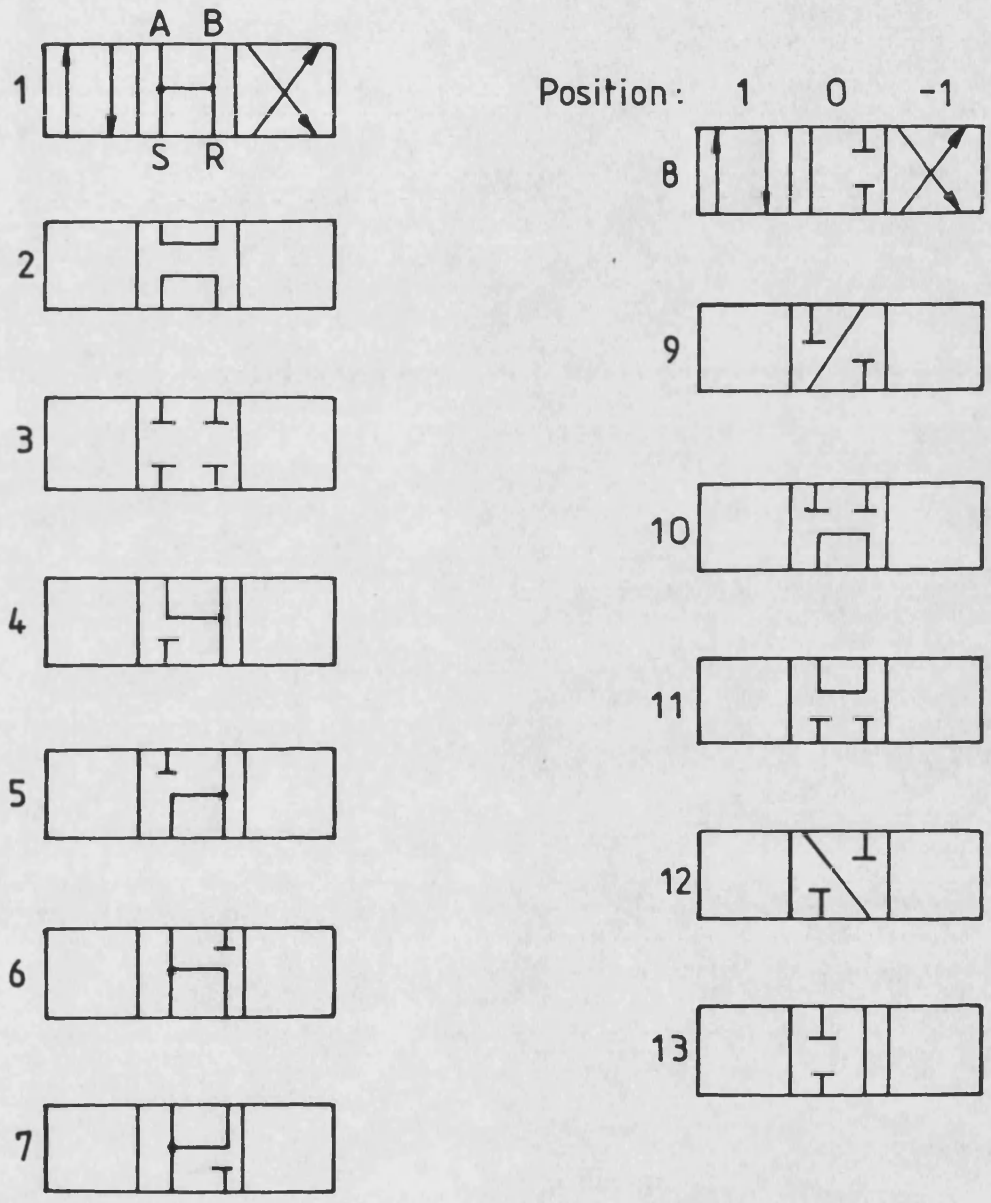
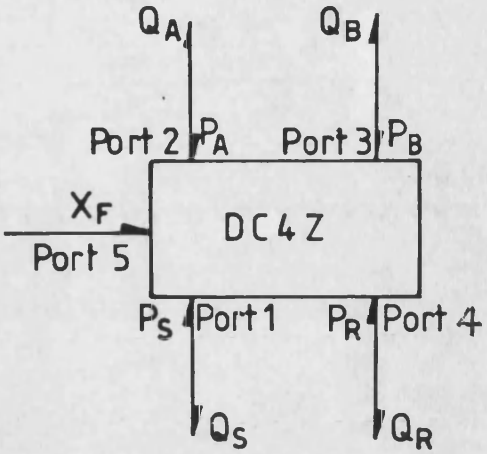


FIG. B2.1 Possible configurations for DC4Z



EFFORTS : $P_S P_A P_B P_R X_F$
FLOWS : $Q_S Q_A Q_B Q_R$
STATE VARIABLES : —

FIG. B2.2 Linking diagram of DC4Z

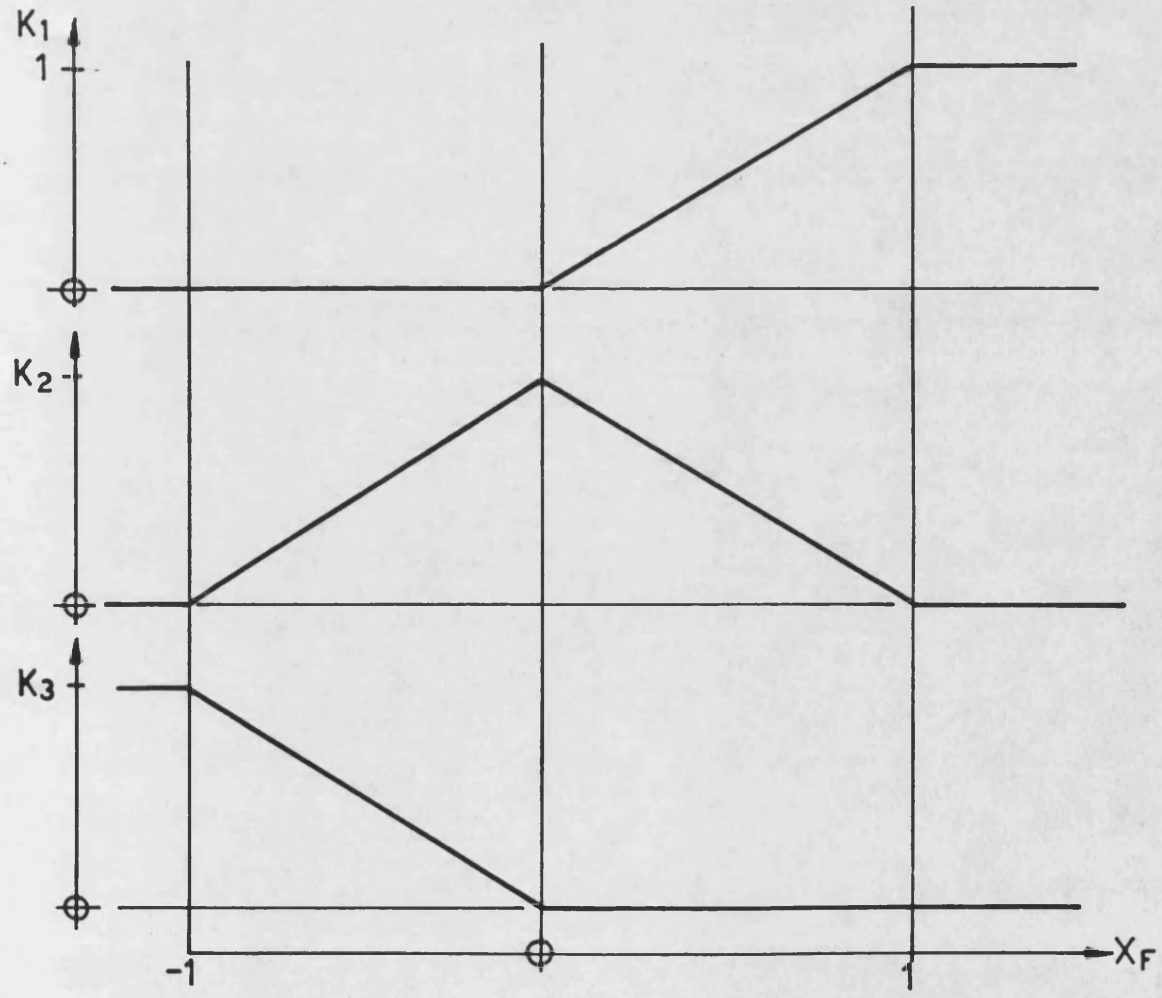


FIG. B2.3 Vartiation of flow factor with position

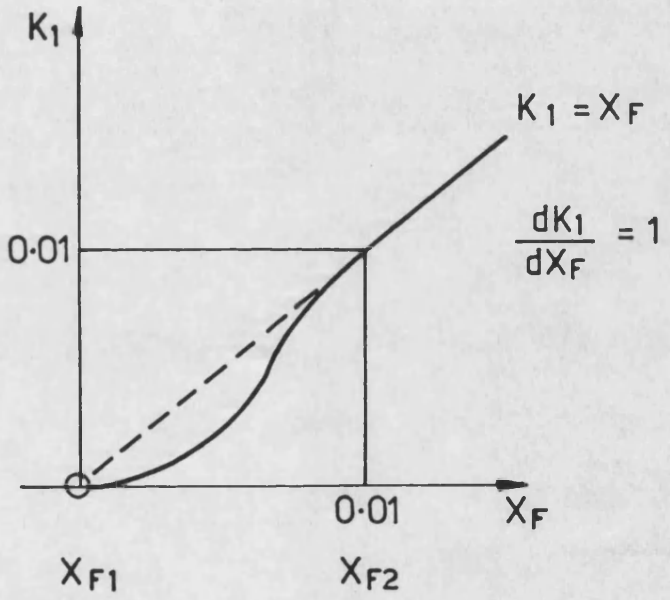


FIG. B2.4 Smoothing of flow factors

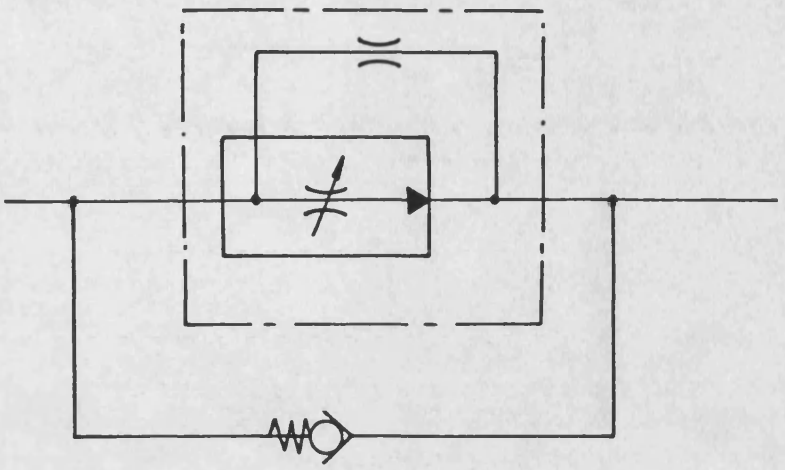
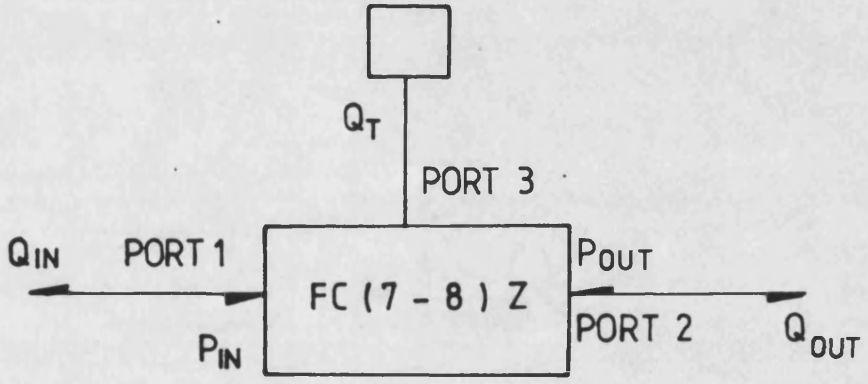


FIG. B3.1 ISO symbol for model FC8Z



EFFORTS : P_{IN} P_{OUT}
FLOWS : Q_{IN} Q_{OUT} Q_T
STATE VARIABLES : Q_T

FIG. B3.2 Linking diagram of FC8Z

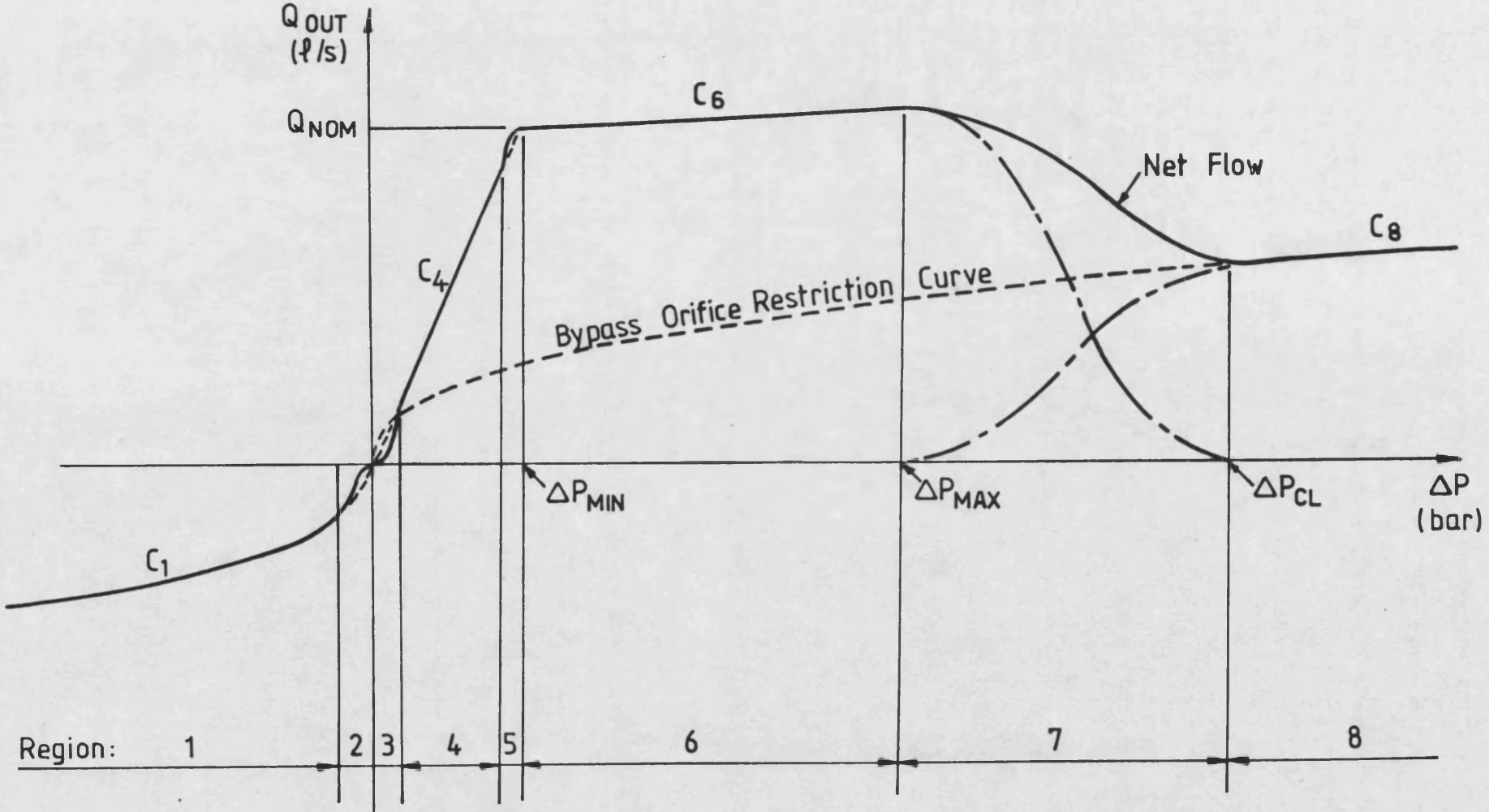


FIG. B3.3 Pressure/flow characteristic of main valve

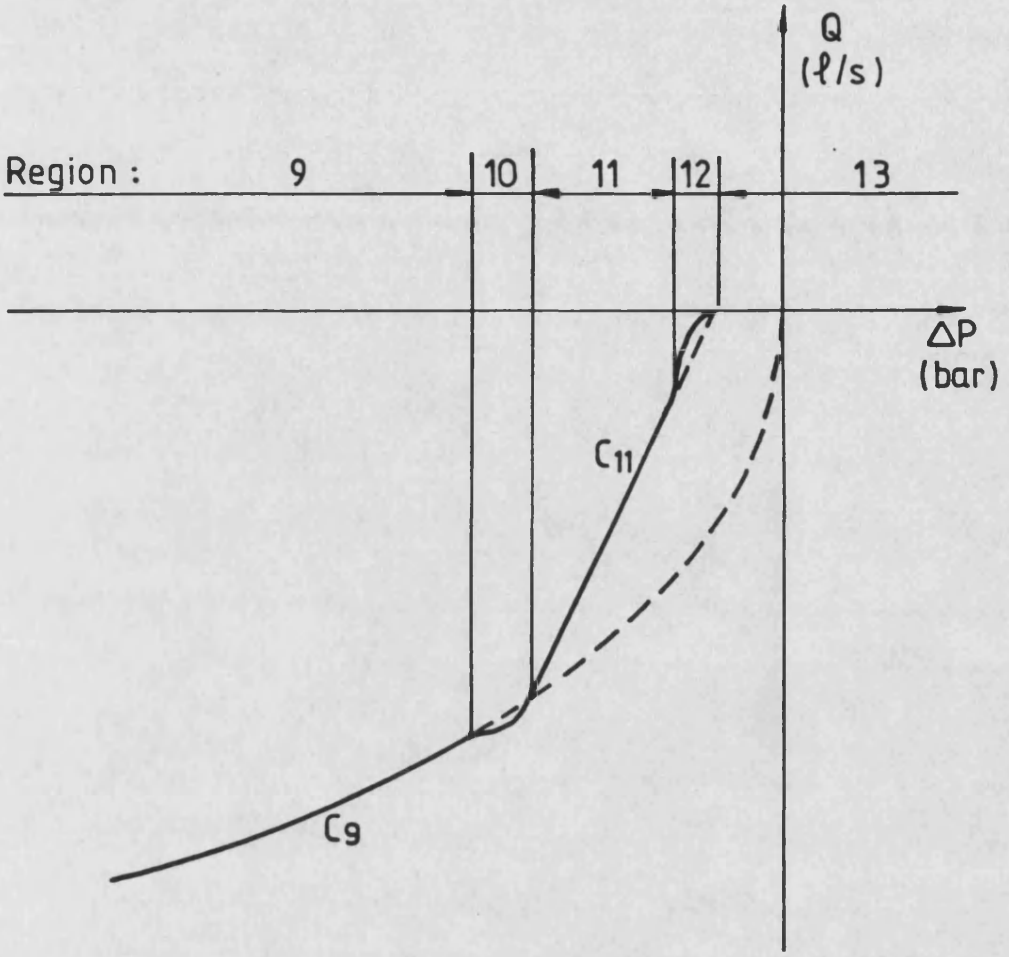


FIG. B3.4 Pressure/flow charactersistic of the reverse fre-flow check valve

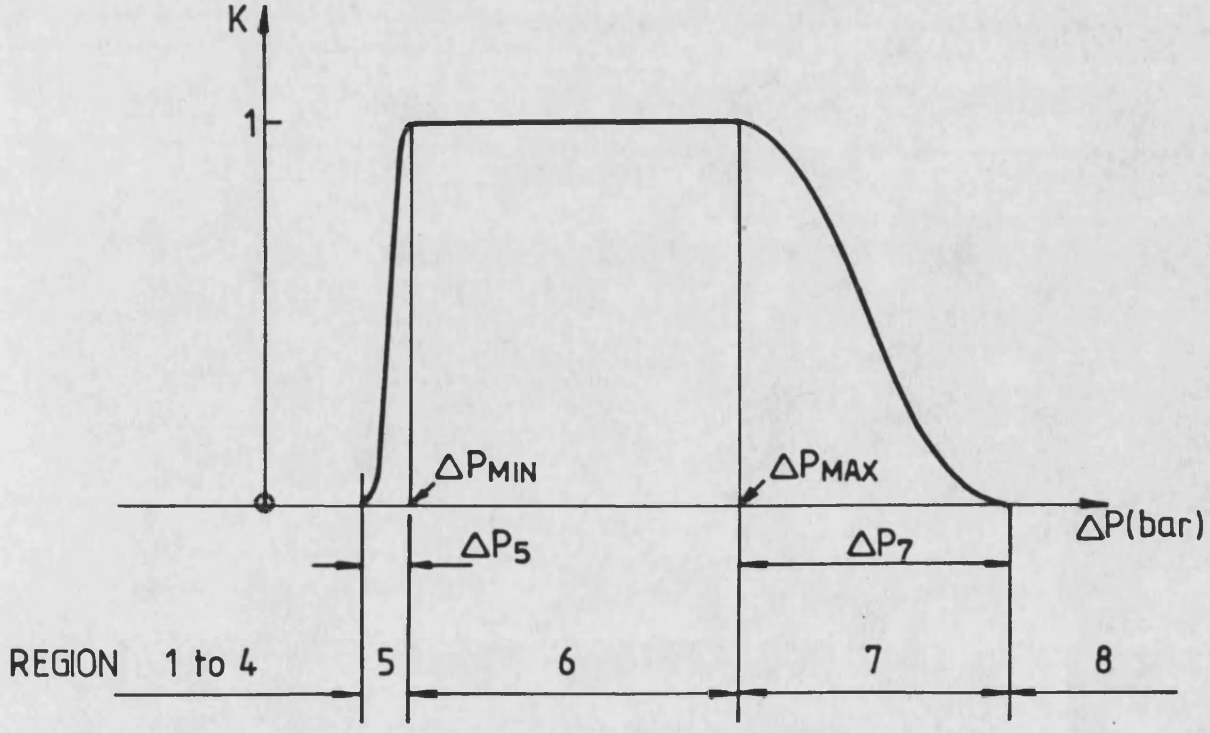


FIG. B3.5 Variation of transient flow factor with pressure

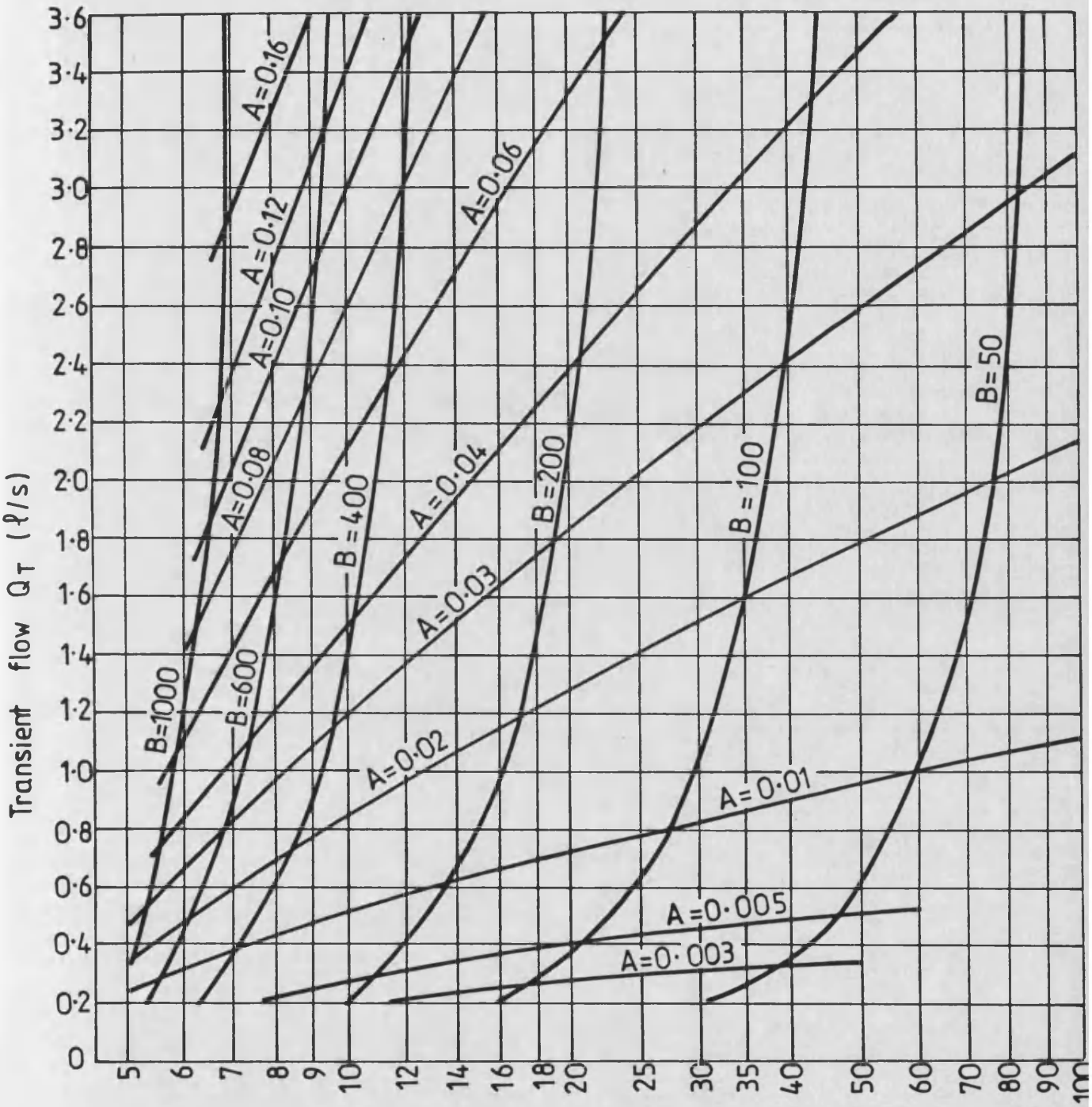


FIG. 3.6 The coefficients A and B

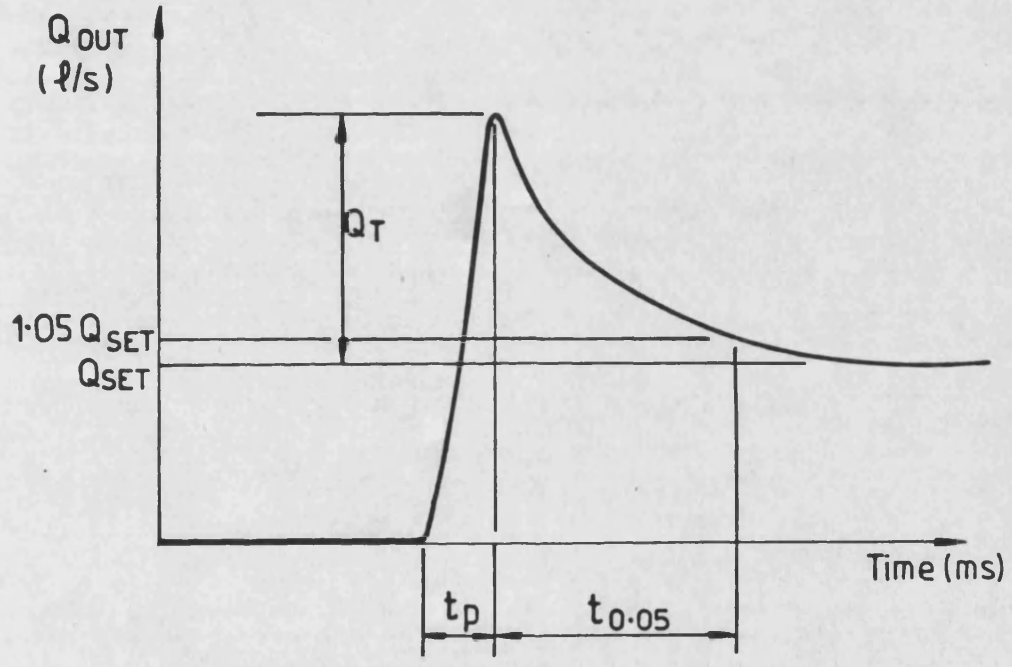


FIG. B3.7 Definition of response times

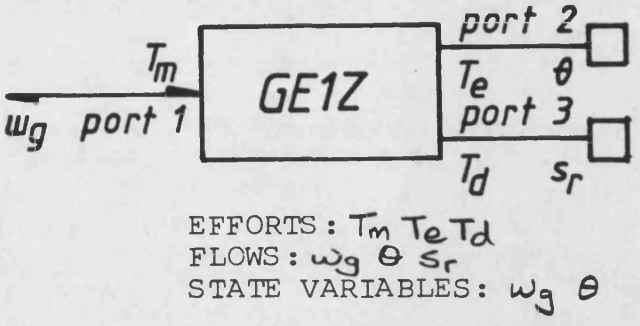


FIG. B4.1 Linking diagram of GE1Z

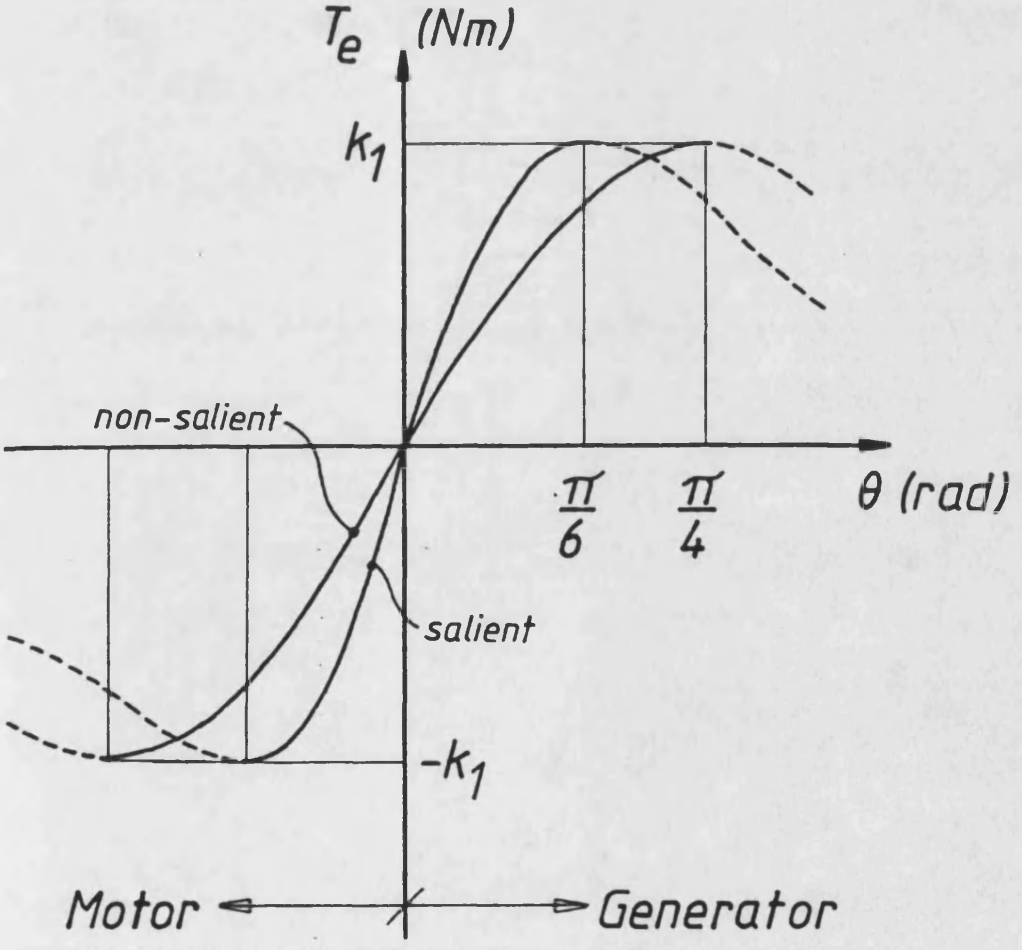


FIG. B4.2 Variation of electrical torque with angle

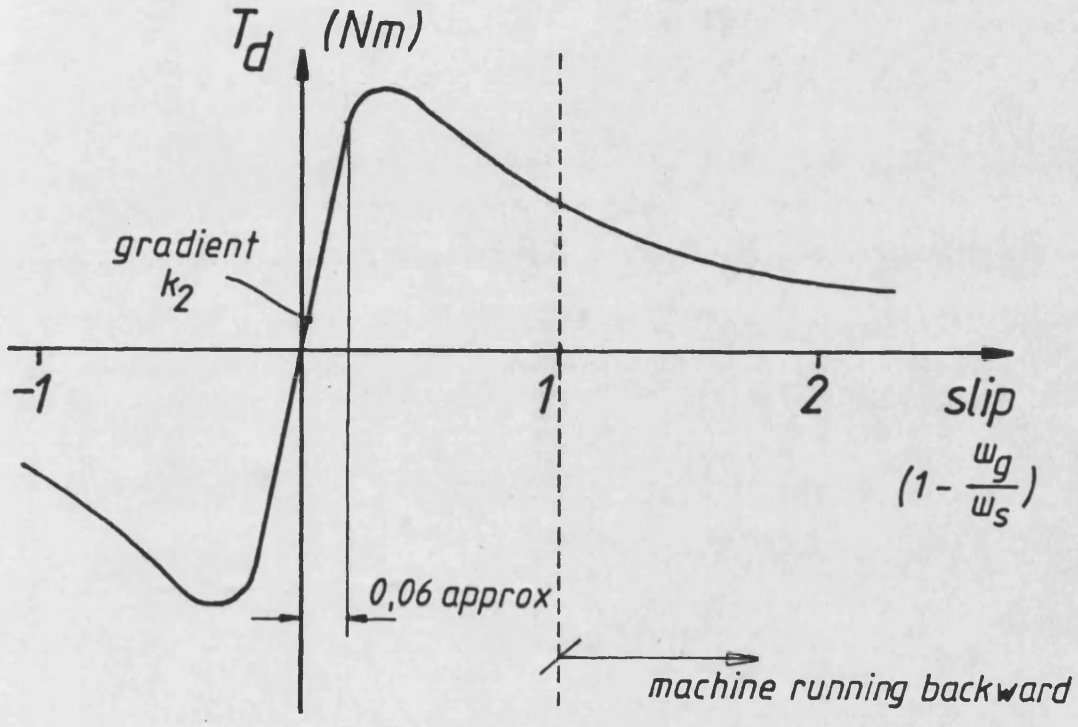


FIG. B4.3 Variation of damping torque with rotor slip

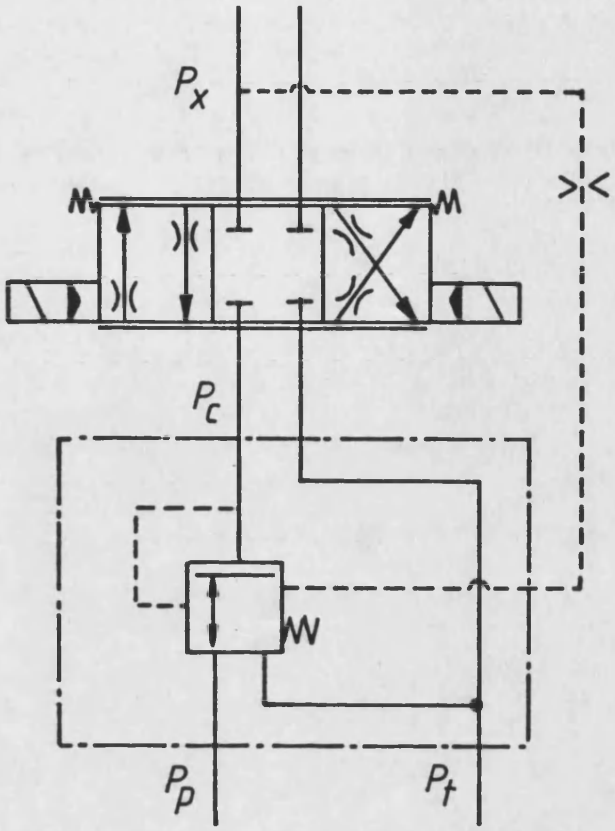
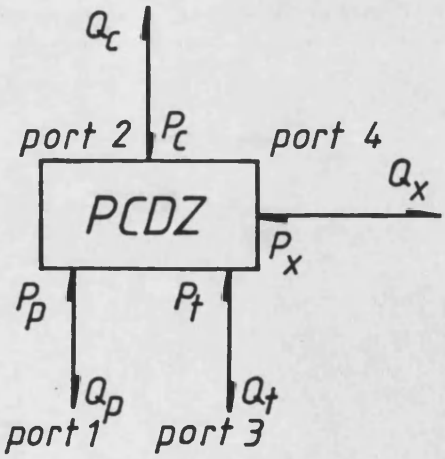


FIG. B5.1 An example of the use of the pressure compensator



EFFORTS: $P_p\ P_c\ P_t\ P_x$
FLOWS: $Q_p\ Q_c\ Q_t\ Q_x$
STATE VARIABLES: —

FIG. B5.2 Linking diagram for PCDZ

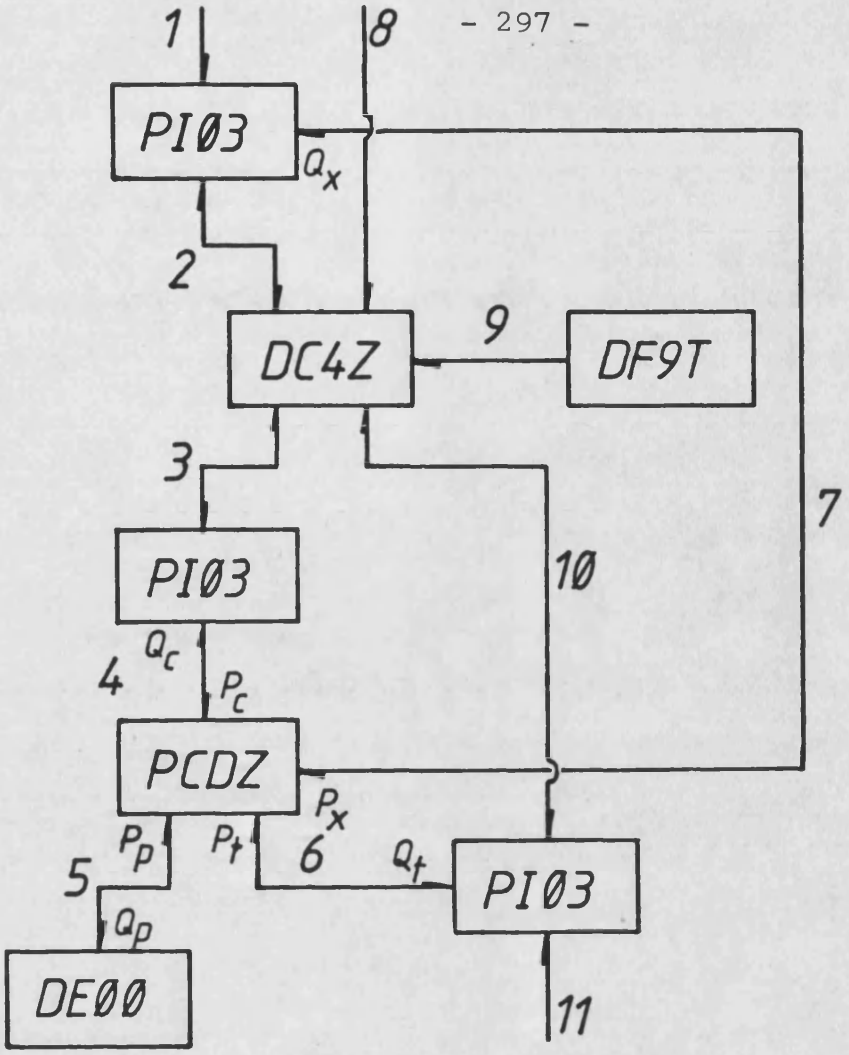


FIG. B5.3 The linking of models to form the example circuit of figure B5.1

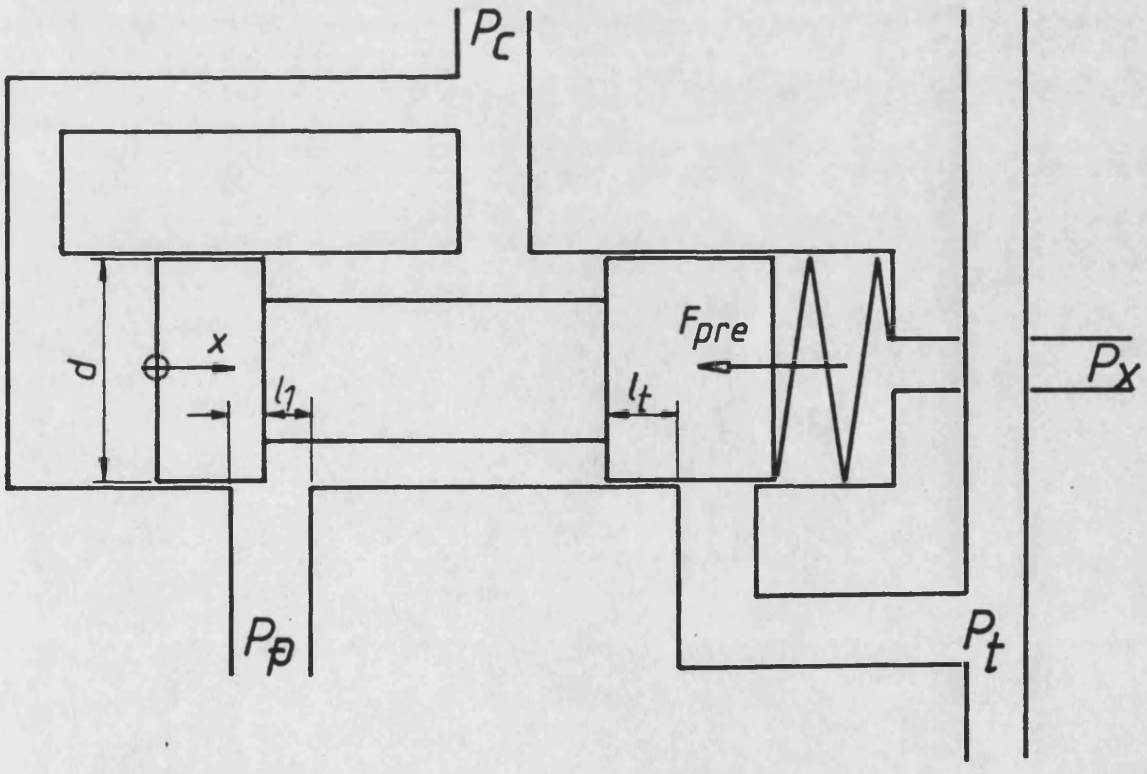
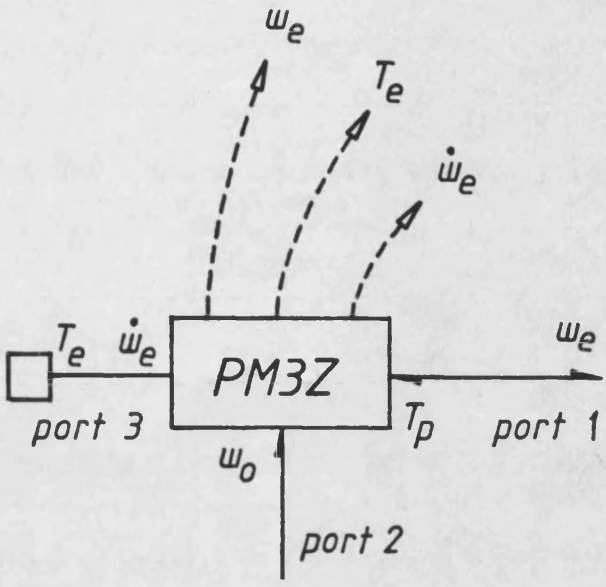


FIG. B5.4 A schematic of the pressure compensator



EFFORTS: T_p T_e $\dot{\omega}_e$
FLOWS: ω_e ω_o $\dot{\omega}_e$
STATE VARIABLES: ω_e

FIG. B6.1 Linking diagram of PM3Z

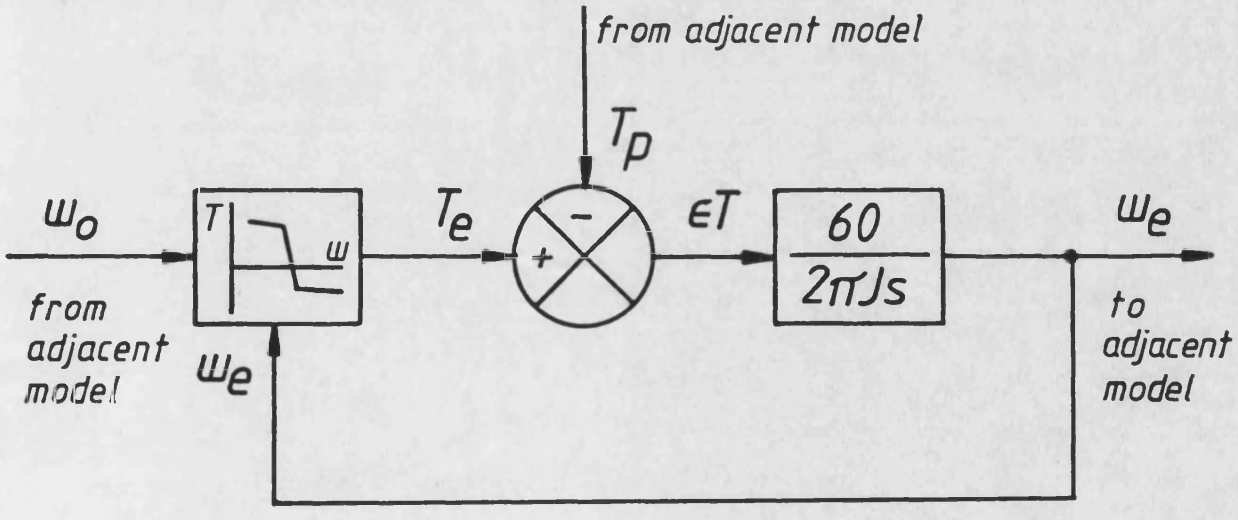


FIG. B6.2 Block diagram of the algorithm of PM3Z

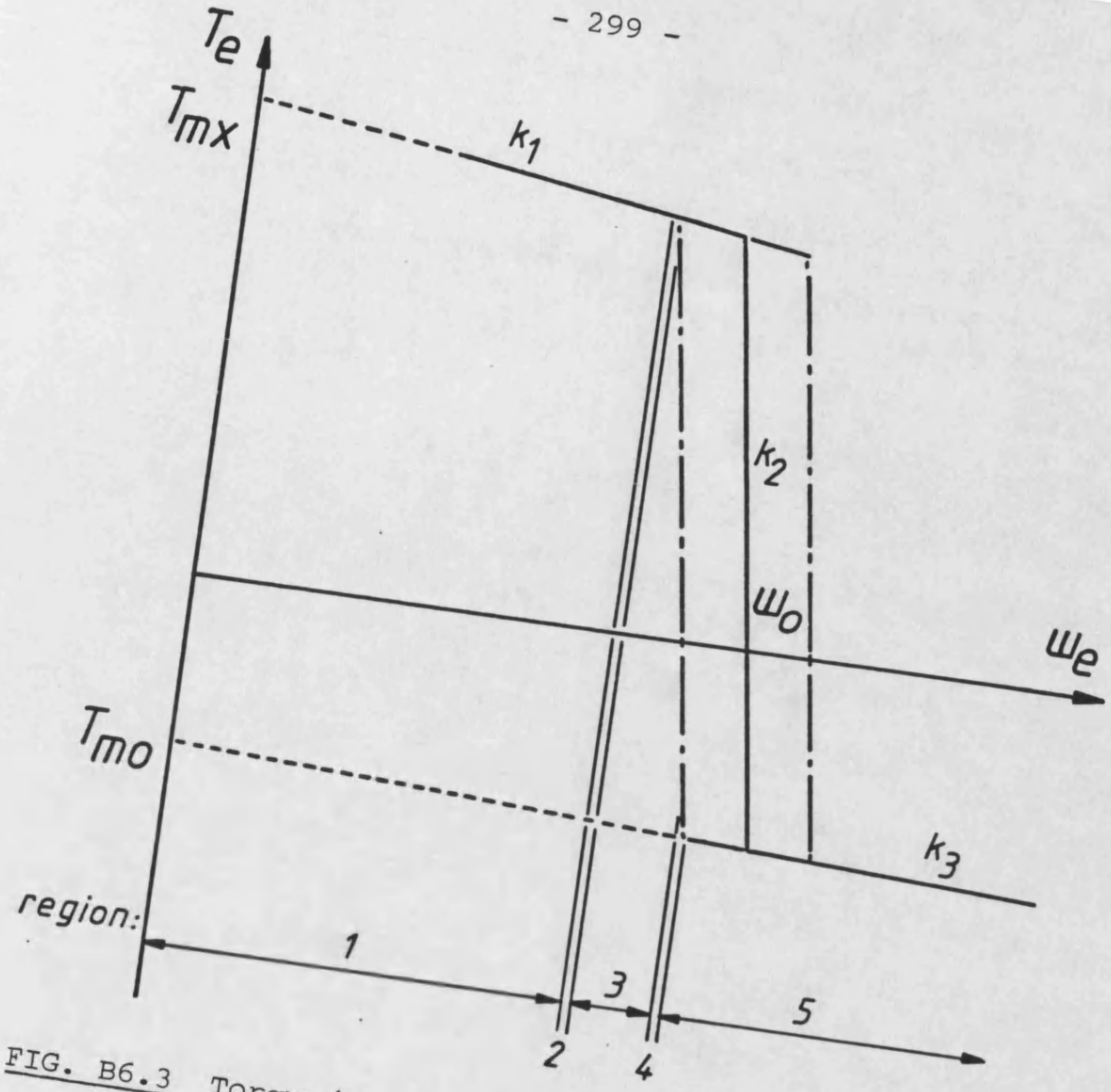


FIG. B6.3 Torque/speed characteristic

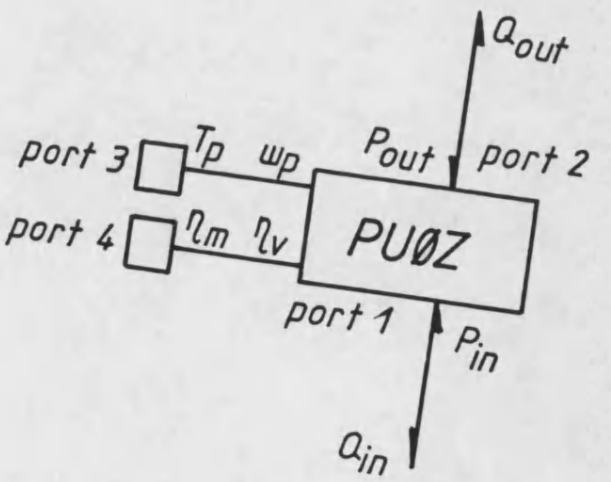


FIG. B7.1 Linking diagram of PUOZ

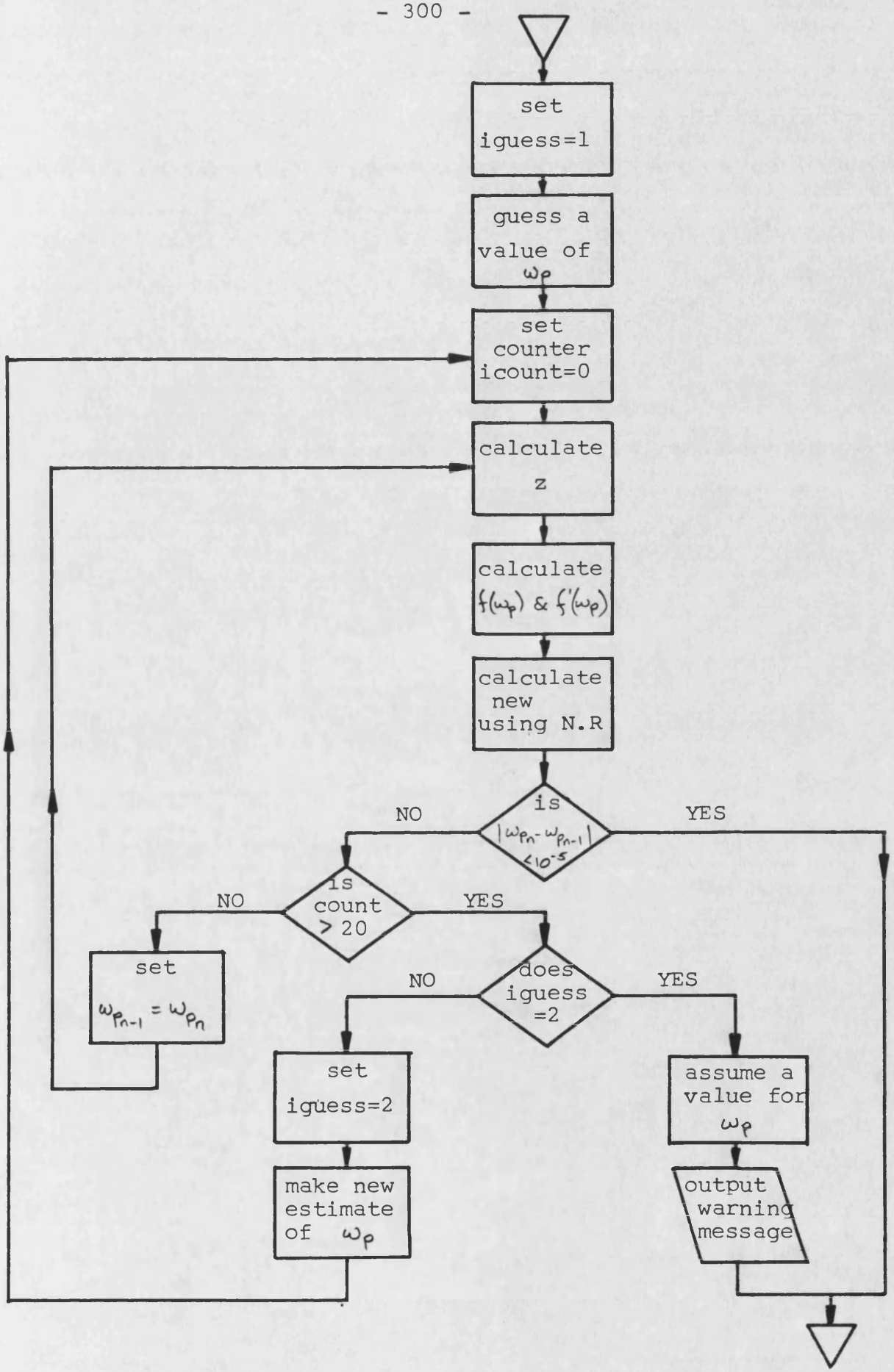


FIG. B7.2 Flow chart of the algorithm of PUOZ

APPENDIX C - GENERAL UTILITY ROUTINES

C.1 MESAGE - ERROR WARNING SUBROUTINE

Introduction

100. If a user inputs a value which is outside normal working limits for the parameter concerned, a message to this effect should be displayed and the user allowed to reconsider. However, if he again inputs a value outside the normal limits then the calculation should proceed unless acceptance of this unusual value would cause a fatal error in either of the model subroutines.

Subroutine MESAGE

101. Subroutine MESAGE is not intended as a complete solution to the problem outlined above, but is as comprehensive as a general subroutine can be. It will inform the user that he has typed an unusual value and will allow him to reconsider. If he then types in the same unusual value the program will continue.

Calling subroutine MESAGE

102. Subroutine MESAGE is listed in Table C.1. MESAGE has 5 arguments which are listed below.

N	A multiple question identifier. In general, N will correspond to the argument of CON( ).
CON	The value of the parameter typed by the user.

VLOW The lower acceptance limit.

VUP The upper acceptance limit.

NPRE An integer variable required to inform the input subroutine whether or not the question should be repeated.

103. Firstly, an integer variable, say IERROR, must be initialized to zero at the beginning of the model input subroutine.

```
e.g.      MR = NR
          IERROR = 0
```

104. The example below is of a question and call statement.

```
1   WRITE(5,2)
2   FORMAT(' TYPE IN CRACKING PRESSURE IN BAR')
    READ(5,3,ERR=1)CON(5)
3   FORMAT(F25.0)
    CALL MESAGE(5,CON(5),1.D0,500.D0,IERROR)
    IF(IERROR.EQ.5)GOTO 1
    IF(MR.EQ.7)GOTO 500
```

105. Occasionally, one of the acceptance limits may depend upon the previous values inputed. In this case, the limit should be calculated before the call statement.

```
e.g.    VLOW=0.1D0*CON(2)
        VUP =1.5D0*CON(2)
        CALL MESAGE(5,CON(5),VLOW,VUP,IERROR)
```

106. Table C.2 is an example of how MESAGE would look to the user.

Changes to the program generator

107. The only change required in the program generator is to PGODL/PGSEL, the subroutines which write the component selector files, where MESAGE is given the same status as FPROP.

C.2 PTI/PTC - PORT AREA SUBROUTINES (Code No.1)

Introduction

200. Subroutine PTC is a general calculation segment which may be called from any model calculation subroutine where the area of flow through a valve port is required. PTC is valid for annular, circular or poppet type ports. PTI is the corresponding input subroutine and is called from the parameter definition subroutine of the model. Two additional features of PTI/PTC are (i) the valve may be normally open or normally closed, though PTI will not allow a normally open poppet type port, and (ii) in the case of annular and circular ports, underlap or overlap may exist.

201. Nomenclature

(see figure C.2)

A_{APC} Area of triangle APC in m^2

A_F Area of flow in m^2

A_{MAX} Maximum possible area of flow in m^2

A_{SECT} Area of sector APCF in m^2

A_{SEG} Area of segment ACE in m^2

D Diameter of port in cm

d_1 Length of port in cm

d_{SP} Diameter of spool in cm

X_l Poppet travel when saturation flow is reached in m

X_o Amount of overlap or underlap in cm

X_v Spool displacement in m

Z Constant in cubic smoothing analysis

Z_1 Line AB in cm

Z_2 Line BP in cm

θ Angle in degrees

Circular port

202. Figure C.1 shows the elevation of the outlet port of a valve.

203. From figure C.2, OE = X_o , OB = X_v , and EF = D. Therefore,

$$EB = X_v - X_o \quad \text{and} \quad BP = D/2 - X_v + X_o$$

204. Since Fortran does not contain the function $\cos^{-1}$, θ must be calculated in terms of $\tan^{-1}$ which does exist as a Fortran function. Therefore, AB must be calculated.

205. By Pythagoras: $AB = (AP^2 - BP^2)^{1/2}$

$$AB = ((D/2)^2 - (D/2 - x_v + x_o)^2)^{1/2}$$

i.e. $$AB = (Dx_v - Dx_o - x_v^2 + 2x_v x_o - x_o^2)^{1/2}$$

Let $Z_1 = AB$

and $Z_2 = BP$

$$\frac{\theta}{2} = \tan^{-1}\left(\frac{Z_1}{Z_2}\right)$$

i.e. $$\theta = 2\tan^{-1}\left(\frac{Z_1}{Z_2}\right)$$

206. However, if OB > OP, i.e. $\theta > 180^\circ$, then BP becomes negative.

207. Therefore, $|Z_2|$ must be used in the expression for θ and added, if θ should be greater than 180°.

i.e.

$$\theta = 2\tan^{-1}\left(\frac{Z_1}{|Z_2|}\right) \quad \text{if } x_v \leq x_o + \frac{D}{2}$$

and

$$\theta = 2\tan^{-1}\left(\frac{Z_1}{|Z_2|}\right) + \pi \text{ if } x_v > x_o + \frac{D}{2}$$

208. Areas

$$A_{SECT} = \frac{\theta}{2\pi}\frac{\pi D^2}{4} = \frac{D^2\theta}{8}$$

$$A_{APC} = Z_1 Z_2$$

$$A_{SEG} = A_{SECT} - A_{APC} = \frac{D^2\theta}{8} - Z_1 Z_2$$

If $\theta > 180^\circ$, Z_2 is negative. Therefore, the expression for A_{SEG} is true, even if θ is reflex.

The flow area,

$$A_F = \frac{\pi D^2}{4} - A_{SEG}$$

i.e.

$$A_F = \frac{D^2}{4}\left(\pi - \frac{\theta}{2}\right) + Z_1 Z_2$$

when

$$x_o = x_v, \quad \frac{dA_F}{dx_v} = 0$$

Therefore, cubic smoothing is not required.

209. Annular port

(i) Calculation of the flow area

From figure C.3:

$$OE = x_v$$

$$OF = x_o$$

$$EG = d_1$$

Therefore,

$$EF = x_v - x_o$$

$$FG = d_1 - x_v + x_o$$

The flow area,

$$A_F = (d_1 - x_v + x_o\pi)\, d_{SP}$$

(ii) Cubic smoothing of the function

210. From figure C.4 it can be seen that the variation of A_F with X_v is of the form of a ramp function. Therefore cubic smoothing is required in two regions.

211. The lower cubic smoothing region

$$f(x_1) = A_{MAX}$$

$$z = (x_v - x_o)/h$$

$$A_F = 0.01A_{MAX}z^3 - 0.02A_{MAX}z^2 + A_{MAX}$$

212. The upper cubic smoothing region

$$f(x_1) = 0.01 A_{MAX}$$

$$z = (x_v - x_o)/h$$

$$A_F = 0.01 A_{MAX} z^3 - 0.01 A_{MAX} z^2 - 0.01 A_{MAX} z + 0.01 A_{MAX}$$

Poppet type port

213. Figure C.5 shows a schematic section of a poppet type port. Figure C.6 shows a plot of X_v against A_F.

(i) Calculation of flow area

214. The flow area is a conic frustum where

$$A_F = \pi d x_v \sin\theta$$

215. However, X_v is limited to allow for saturation flow. This limit is reached when the flow area of the frustum is equal to the cross-sectional area of the port.

i.e. $$\frac{\pi d^2}{4} = \pi d x_L \sin\theta$$

Therefore,

$$x_L = \frac{d}{4 \sin\theta}$$

216. Therefore, if X_v is calculated to be greater than X_L, the flow area calculation is based upon X_L.

(ii) Cubic smoothing region

217. The graph of X_V against A_F would show a discontinuity at $X_V = X_L$ if a cubic polynomial was not fitted to this region.

$$f(x_1) = 0.99\, A_{MAX}$$

$$z = (x_V - 0.99\, x_L)/0.01\, x_L$$

$$A_F = -0.01\, A_{MAX} z^3 + 0.01\, A_{MAX} z^2 + 0.01\, A_{MAX} z + 0.99 A_{MAX}$$

Calling up subroutine PTI

218. PTI should be called from the model input subroutine during the interactive read section and the display and edit section.

219. PTI is called from the interactive read section as shown below.

```
C ***** INTERACTIVE READ SECTION
        CALL PTI(1,MR,ICON(1),CON(1),CON(2),CON(3))
```

where 1 is a multiple port identifier

MR is the read mode identifier

ICON(1) is the shape of the port

CON(1) is the port lap/
diameter if poppet type

CON(2) is the port size/
poppet angle if poppet type

CON(3) is the spool diameter/
limit in X if poppet type

220. PTI is called from the display and edit section as shown below.

```
C **** DISPLAY AND EDIT SECTION
      MR=7
  499 CALL PTI (1,MR,ICON(1),CON(1),CON(2),CON(3))
  500 WRITE ....
```

221. The only other unusual feature of the model input subroutine is in the "TRANSFER TO APPROPRIATE READ SECTION" where the following statement occurs.

```
    IF(MR.EQ.7)GOTO 499
```

Calling up subroutine PTC

222. PTC should be called up from the model calculation subroutine after the spool displacement, X_V , has been evaluated and before the flow area A_F , is required.

223. Using the variable names defined above, PTC is called as follows.

```
   CALL PTC(0,ICON(1),CON(1),CON(2),CON(3),XV,AF)
```

where the first argument indicates the type of port

i.e. 0 for normally open,

1 for normally closed.

Changes to the program generator

224. The only change required in the program generator is to PGODL/PGSEL, the subroutines which write the component selector

files, where PTI and PTC are given the same status as FPROP and MESAGE.

C.3 REGR/SIMUL - REGRESSION FUNCTIONS (Code No.2)

Introduction

300. In principle, the regression functions may be used for two purposes. Firstly, they may be used to fit polynomials to discrete data defined by the user. For example, the user could define several points of a pump torque loss characteristic during the parameter definition section. The regression routines would then be employed to fit a suitable curve which would subsequently be employed by the model calculation routines.

301. However, the regression routines may also be used to fit curves to functions which are computationally very slow to evaluate. In this case, the function in question is evaluated at a number of points and these points used to derive the polynomial. It is important that the values of the independent variable should encompass the possible range of operation. It is well known that the error between the polynomial and the supplied data points is greatest at the extremes of the independent variable. Therefore, since the regression routines described do not apply extra weighting to the extreme points, points should ideally be defined well outside the normal operating range. It was for this second method of use that the regression routines were originally developed.

302. Nomenclature

a polynomial constant

b_i polynomial coefficient of x_i (i = 1,n)

c_{ij} coefficient of the normal equations (i,j = 1,n)

m number of data points

n order of polynomial

s sum of the squares of the errors

x_i data point

y independent variable

Calling the regression routines

303. The regression routines are called from a model parameter definition subroutine. The function REGR is called by the model routine in order to set up the regression equations, and the function SIMUL is called by function REGR to solve the resulting matrix.

304. The component model must define the dependent and independent variables contained in the arrays x and y. It must also define the number of data points, m, and the order of polynomial required, n. In return, the regression function calculates the coeffcicients of the polynomial. The constant is defined as the variable a and the coefficients of the powers of x is contained in the array b. Additionally, the function returns the standard deviation of the defined points about the polynomial, s.

305. It is possible to write the order of polynomial required into the parameter definition subroutine at the time of its development.

However, the order which gives the best fit is often dictated by the relative magnitude of various user defined parameters. Therefore, it is considered better practise to ensure that the order may be defined during the simulation process. This may be done by calculating a series of polynomials of differring order and using the polynomial which produces the lowest standard deviation.

306. An example of a section of coding from a parameter definition subroutine is given below. In this example, the polynomial of lowest standard deviation is automatically selected.

```
      IMPLICIT  REAL*8 (A-H,O-Z)
      DIMENSION X(50), Y(50), B(10), S(10)
      .
      .
C
C ***** DEFINE THE ARRAYS X AND Y
      .
      .
C
C ***** DEFINE M TO INDICATE 50 DATA POINTS
      M = 50
C
C ***** INITIALISE THE FLAG TO INDICATE LOWEST S
      LOWEST = 1
C
C ***** CARRY OUT THE REGRESSION CALCULATION UP TO 10TH ORDER
      DO 100 N=1,10
C
C ***** CALL THE FUNCTION REGR FOR THE NTH ORDER POLYNOMIAL
      S(N) = REGR (M, X, Y, N, A, B)
C
C ***** UPDATE THE FLAG IF CURRENT S IS LESS THAN 1.05 x LOWEST SO FAR
      IF (N .GT.  1 .AND. S(N) .LT. 1.05D0*S(LOWEST)) LOWEST = N
C
C ***** END OF LOOP
 100  CONTINUE
C
C ***** INITIALISE THE COEFFICIENT ARRAY
      DO 200 N=1,10
 200  B(N) = 0.0D0
C
C ***** CALL FUNCTION REGR AGAIN FOR POLYNOMIAL WITH LOWEST S
```

```
DUMMY = REGR (M, X, Y, LOWEST, A, B)
.
.
```

307. The regression function is called ten times to calculate the coefficients for the corresponding orders. The flag LOWEST is employed to record the order of polynomial required. One would expect the standard deviation to rapidly reduce with increasing order until a good fit were obtained, then to possibly increase again as the order increased further. In the coding above, the optimum order is defined as that which for the next higher order polynomial causes a 5% reduction in the standard deviation. The regression routine is then called again in order to re-calculate the coefficients for the chosen polynomial. The reason for this duplication is one of minimising size at the expense of longer execution time. It would be possible to store the coefficients for every order of polynomial in A and B. However, A would be an array rather than a single variable and B would be a two-dimensional array rather than one-diemnsional. The execution time is subsequently increased but since the time required to carry out the regression analysis is approximately one second, the extra time is unlikely to be noticed by the user.

Invoking the regression routines

308. The functions REGR and SIMUL are incorporated in a simulation program provided the modeller has included an appropriate entry in the component attributes data file COMPON.DAT. This additional entry consists of single integer digits. The first digit states the number of optional utility routines that are required and the subsequent

digits are code numbers which define those utilities. In the case of the regression functions, the code number is 2.

Function REGR

309. The function REGR performs a polynomial regression analysis on a set of data points. The resulting equation is of the form

$$y = a + b_1x + b_2x^2 + \ldots + b_ix^i + \ldots + b_nx^n$$

The requirement is to minimise the sum of the squares of the errors between the polynomial and the data points. The sum of the squares of the errors, s, is given by

$$s = \sum (y - a - b_1x - b_2x^2 - \ldots - b_ix^i - \ldots - b_nx^n)$$

The minimum is found by setting the partial derivatives of the coefficients a and b_i to zero. The set of resulting equations are termed the normal equations and are shown below.

$$\begin{aligned}
c_{11}b_1 + c_{12}b_2 + \ldots + c_{1n}b_n &= c_1y \\
c_{21}b_1 + c_{22}b_2 + \ldots + c_{2n}b_n &= c_2y \\
\vdots \qquad\qquad \vdots \qquad\qquad &\vdots \\
c_{n1}b_1 + c_{n2}b_2 + \ldots + c_{nn}b_n &= c_ny
\end{aligned}$$

and also

$$a = \bar{y} - \sum_i b_i\bar{x}_i$$

where the coeffcients c_{ij} , c_{iy} and c_{yy} are defined as

$$c_{ij} = \sum x_i x_j - \frac{\sum x_i \sum x_j}{m}$$

$$c_{iy} = \sum x_i y - \frac{\sum x_i \sum y}{m}$$

$$c_{yy} = \sum y^2 - \frac{(\sum y^2)^2}{m}$$

The standard deviation is calculated by

$$s = \left(\frac{(c_{yy} - \sum b_i c_{iy})}{m-n-1} \right)^{\frac{1}{2}}$$

310. The function REGR initially computes the sums of the powers and the products in order to define the coeffcients c , c and c . The function SIMUL is then called in order to solve the set of simultaneous equations. If the function SIMUL encounters a near singular matrix, then REGR returns a value of zero. Finally, REGR calculates the constant a and the standard deviation s.

Function SIMUL

311. The function SIMUL solves the set of n simultaneous normal equations by the Gauss-Jordan complete elimination method employing a maximum pivot strategy [15].

C.4 CUBIC - POLYNOMIAL SMOOTHING OF DISCONTINUITIES

Introduction

400. Almost all HASP component models incorporate functions which cause derivatives to become discontinuous with respect to time. As mentioned in Chapter 4, discontinuous derivatives may cause failure of the integration routine. Therefore, almost all HASP component models incorporate cubic polynomials to smooth these discontinuous functions. Appendix D derives this cubic polynomial and expresses it in two different forms. The first is in terms of a non-dimensionalised parameter z, the second is in terms of the independent variable x. Traditionally, all HASP modellers adopted the first approach since the derivation of the coefficients of z is certainly simpler than the derivation of the coefficients of x. However, computationally, it is more efficient to use the second approach since the parameter z would no longer require evaluation.

401. The subroutine CUBIC calculates the coefficients of a cubic polynomial expressed in terms of the independent variable x. The routine is called from a model input routine and since it should be used universally, indication of its use need not be included in the component attributes file COMPON.DAT.

402. Nomenclature

a_n coefficient of x^n

f dependent variable

x independent variable

Calling the subroutine CUBIC

403. CUBIC is called from a parameter definition subroutine once the boundary conditions of the cubic have been evaluated. The parameters required are the values of the independent variable, the function and the derivative of the function at the upper and lower boudaries of the range through which the cubic is to be fitted. The subroutine returns the coefficients of the equation

$$f = a_3x^3 + a_2x^2 + a_1x + a_0$$

404. The subroutine is called as shown below.

```
X1  = ...
X2  = ...
F1  = ...
F2  = ...
FD1 = ...
FD2 = ...
.
.
CALL CUBIC (X1, X2, F1, F2, FD1, FD2, A3, A2, A1, A0)
.
.
```

In practise, the variables containing the coefficients of the cubic, A0 to A3, would be elements of the real array CON.

405. Since the subroutine is called during the parameter definition process of a simulation, the CALL statement has a negligible effect on the overall run-time of the simulation. However, the simulation program no longer calculates z and perhaps more importantly, the modeller no longer has to calculate polynomial coeffcients by hand.

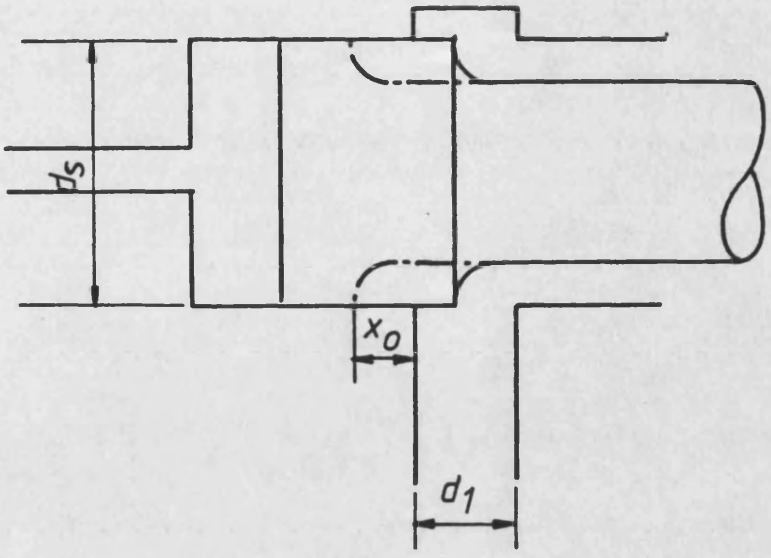


FIG. C.1 A circular outlet port

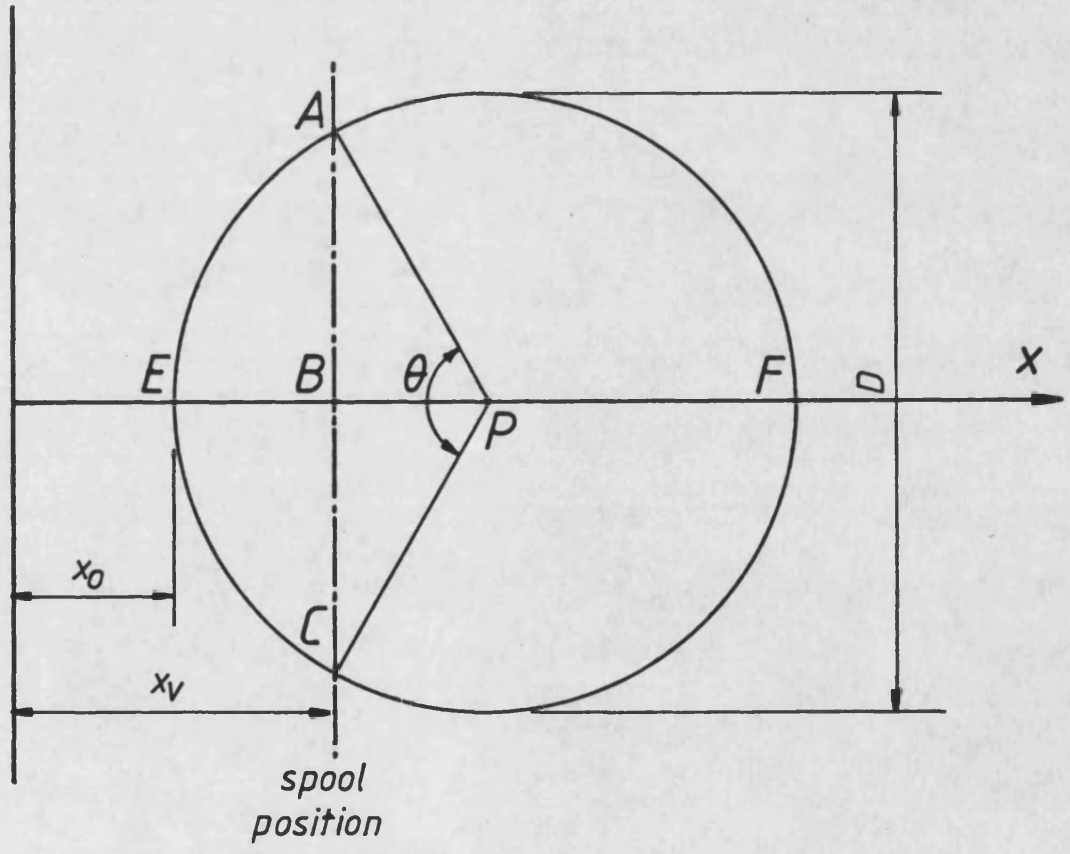


FIG. C.2 The geometry of a partially open port

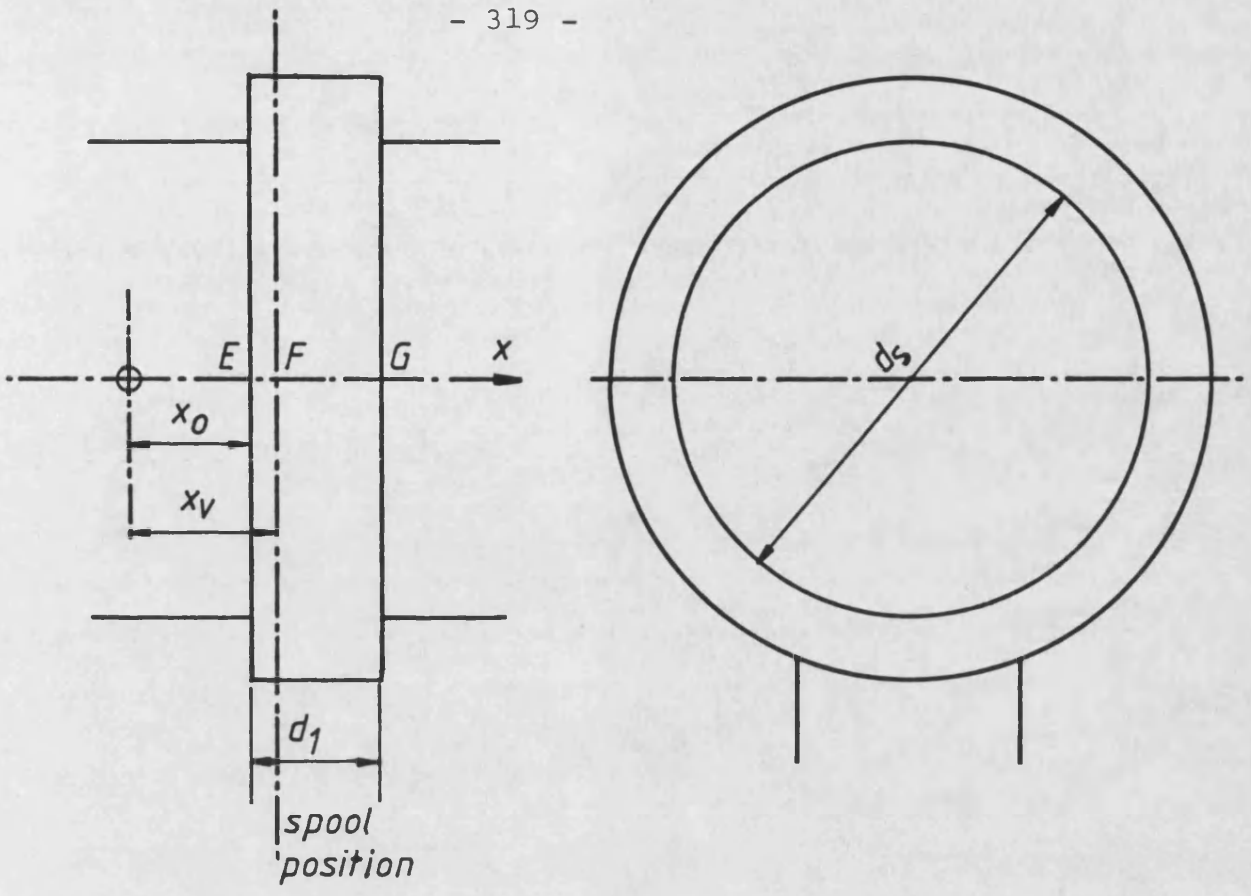


FIG. C.3 An annular outlet port

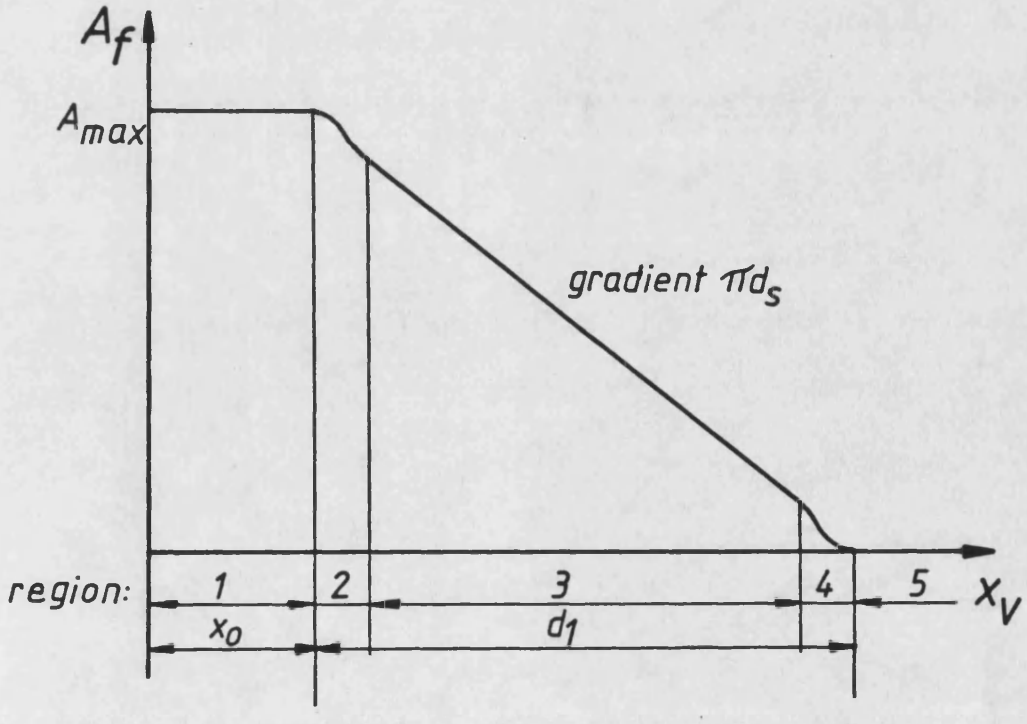


FIG. C.4 The variation of flow area of an annular port

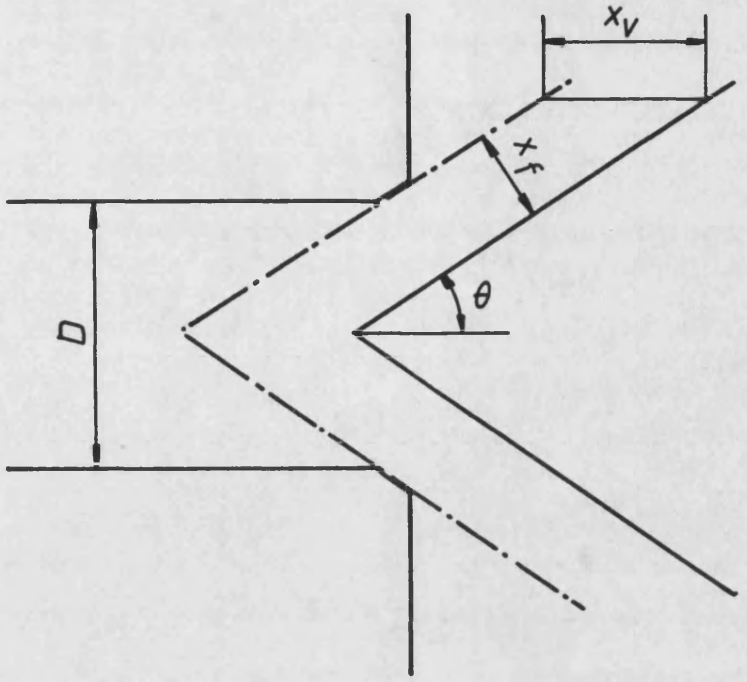


FIG. C.5 A poppet port

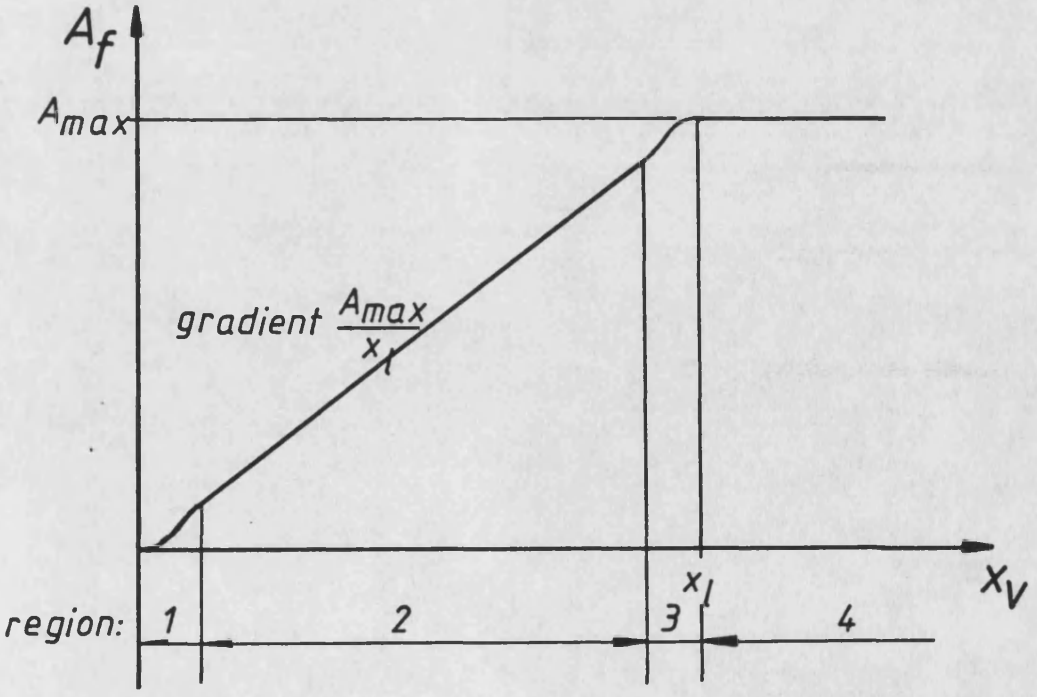


FIG. C.6 The variation of the flow area of a poppet port

```
      SUBROUTINE MESAGE(N,CON,VLOW,VUP,NPRE)
      IMPLICIT REAL*8(A-H,O-Z)
C ***************************************************************************
C ***** GENERAL SUBROUTINE TO OUTPUT POSSIBLE ERROR WARNINGS ***********
C ***** DURING THE PARAMETER DEFINITION SECTION OF THE HASP. ***********
C ***************************************************************************
      IF(CON.LT.VLOW.OR.CON.GT.VUP)GOTO 1
C ***** CON IS WITHIN ACCEPTABLE LIMITS
      NPRE=0
      RETURN
C ***** CON MAY BE UNACCEPTABLE
   1  IF(NPRE.NE.0)GOTO 2
      GOTO 3
C ***** IF QUESTION IS STILL THE SAME AS IN PREVIOUS CALL TO MESAGE
C ***** AND THIS CON IS NUMERICALLY THE SAME AS THE LAST CON
C ***** THEN ACCEPT THIS VALUE
   2  IF(CON.LT.(CONPRE-1.D-5).OR.CON.GT.(CONPRE+1.D-5))GOTO 3
      NPRE=0
      RETURN
C ***** OUTPUT WARNING MESAGE AND STORE CON AND N FOR POSSIBLE USE
C ***** IN NEXT CALL TO MESAGE
   3  WRITE(6,4)
   4  FORMAT('0***** VALUE IS OUTSIDE NORMAL WORKING LIMITS *****',/)
      CONPRE=CON
      NPRE=N
      RETURN
      END
```

TABLE C.1 Listing of subroutine MESAGE

```
1.    TYPE CRACKING PRESSURE OF RELIEF VALVE IN BAR
      1E-2

      ***** VALUE OUTSIDE NORMAL WORKING LIMITS *****

      TYPE CRACKING PRESSURE OF RELIEF VALVE IN BAR
      0.1

      ***** VALUE OUTSIDE NORMAL WORKING LIMITS *****

      TYPE CRACKING PRESSURE OF RELIEF VALVE IN BAR
      .
      .


2.    TYPE CRACKING PRESSURE OF RELIEF VALVE IN BAR
      5000

      ***** VALUE OUTSIDE NORMAL WORKING LIMITS *****

      TYPE CRACKING PRESSURE OF RELIEF VALVE IN BAR
      5000
      TYPE MAXIMUM RATED FLOW OF THE VALVE IN L/S
      .
      .
```

FIG. C.2 A typical question and answer sequence (the user's response is underlined)

APPENDIX D - INTEGRATOR CONTROL

D.1 THE SMOOTHING OF DISCONTINUOUS FUNCTIONS

Derivation of the original cubic polynomial

100. The original form of the smoothing polynomial was developed using the method of exact interpolation developed by Hermite [28], itself an extension of Lagrange interpolation. The use of the Lagrange polynomial requires the definition of a function at discrete values of the independent variable. Definition of n points will result in an exact polynomial with n coefficients, i.e. of order n-1. In addition, Hermite interpolation requires the definition of the derivative of the function at these discrete points. The number of conditions to be met is now 2n. Therefore, the resulting polynomial is of order 2n-1. The most useful form of the Hermite interpolation formula for n points is given as,

$$f(x) = \sum_{i=1}^{n} \left\{ \left[1 + 2 \sum_{\substack{j=1 \\ j \neq i}}^{n} \left(\frac{x_i}{x_i - x_j} \right) - 2 \sum_{\substack{j=1 \\ j \neq i}}^{n} \left(\frac{x}{x_i - x_j} \right) \right] \cdot \prod_{\substack{j=1 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right)^2 f(x_i) + (x - x_i) \prod_{\substack{j=1 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right)^2 f'(x_i) \right\}$$

101. We merely require a polynomial which is continuous in both function and derivative at the boundaries of an extremely small transition region between two major operating regions (figure D.1). Therefore, it follows that the polynomial we require will be of third

order, i.e. a cubic. A polynomial of lower order could not be guaranteed to satisfy our boundary conditions, and there are insufficient conditions to define the coefficients of a higher order polynomial. Expansion of the Hermite interpolation formula for n=2 yields,

$$f(x) = \left[1 + 2\left(\frac{x_1}{x_1-x_2}\right) - 2\left(\frac{x}{x_1-x_2}\right)\right]\left(\frac{x-x_2}{x_1-x_2}\right)^2 f(x_1)$$

$$+ \left[1 + 2\left(\frac{x_2}{x_2-x_1}\right) - 2\left(\frac{x}{x_2-x_1}\right)\right]\left(\frac{x-x_1}{x_2-x_1}\right)^2 f(x_2)$$

$$+ (x-x_1)\left(\frac{x-x_2}{x_1-x_2}\right)^2 f'(x_1) + (x-x_2)\left(\frac{x-x_1}{x_2-x_1}\right)^2 f'(x_2) \qquad \text{D.1}$$

In order to simplify this lengthy equation, all values of x are substituted by the non-dimensional z where,

$$z = \frac{x-x_2}{x_2-x_1}$$

Also, let

$$h = x_2 - x_1$$

These substitutions give,

$$f(x) = \left(1 - \frac{2x_1}{h} + 2z - \frac{2x_1}{h}\right)(1-z)^2 f(x_1)$$

$$+ \left(1 + \frac{2x_2}{h} - 2z - \frac{2x_1}{h}\right) z^2 f(x_2)$$

$$+ hz(1-z)^2 f'(x_1) + hz^2(z-1) f'(x_2)$$

Expanding and grouping orders of z gives,

$$f(x) = z^3\left(2f(x_1) + hf'(x_1) - 2f(x_2) + hf'(x_2)\right)$$

$$+ z^2\left(-3f(x_1) - 2hf'(x_1) + 3f(x_2) - hf'(x_2)\right)$$

$$+ zhf'(x_1) + f(x_1)$$

This equation may be readily included in a model calculation subroutine in the form shown. However, the coeffcients can often be simplified by the modeller at the time of model development.

Derivation of the revised cubic polynomial

102. Evidently, it is possible to expand equation D.1 and group orders of x rather than the non-dimensionalised z as given above. This gives an equation of the form

$$f(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

where,

$$a_3 = \frac{2}{h^3}(f(x_1) - f(x_2)) + \frac{1}{h^2}(f'(x_1) + f'(x_2))$$

$$a_2 = \frac{2}{h^3}(f(x_2)(x_2+2x_1) - f(x_1)(x_1-2x_2)) + \frac{1}{h^2}(f(x_1) + f(x_2) - f'(x_1)(2x_2+x_1) - f'(x_2)(2x_1+x_2))$$

$$a_1 = \frac{2}{h^3}(f(x_1)(2x_1x_2+x^2) - f(x_2)(2x_1x_2+x_1^2)) + \frac{1}{h^2}(-2x_2f(x_1) - 2x_1f(x_2) + f'(x_1)(x_2^2+2x_1x_2) + f'(x_2)(x_1^2+2x_1x_2))$$

$$a_0 = \frac{2x_1x_2}{h^3}(x_1f(x_2) - x_2f(x_1)) + \frac{1}{h^2}(x_2^2f(x_1) - x_1x_2^2f'(x_1) + x_1^2f(x_2) - x_2x_1^2f'(x_2))$$

103. Defining the polynomial in this form is certainly less elegant than defining y as a function of the non-dimensional z. However, elegance is not always the criterion for the production of the most efficient computer algorithm. The coefficients a_0 to a_3 may be

calculated in the calculation section of a model parameter definition subroutine. Furthermore, it is possible to use a standard subroutine to calculate these coefficients (see Appendix C.400). This relieves the modeller of the task of calculating coeffcients and no longer requires the model calculation routine to continually evaluate z.

Smoothing polynomials with variable boundary conditions

104. If the boundary conditions of the function to be smoothed can be defined at the parameter definition stage (e.g. a valve pressure/flow characteristic), then the definition of the cubic polynomial presents no problems. However, occasionally it is not possible to define the boundary conditions of the cubic. This is normally due to the fact that the functions to be smoothed are supplied by a different model. If this is the case, then the following method should be adopted.

105. Figure D.2 shows two arbitrary (i.e. externally defined) functions which must be smoothed in the independent variable range $[x_1, x_2]$. The transition function is defined by factoring the difference of the two functions by a step cubic.

i.e.

$$f_T = f_1 + k(f_2 - f_1) \qquad D.2$$

where the end conditions of the cubic are given by,

$$f(x_1) = 0 \qquad f'(x_1) = 0$$

$$f(x_2) = 1 \qquad f'(x_2) = 0$$

106. Check that the transition function satisfies the continuity requirement at the boundaries of the region.

Differentiating equation D.2,

$$f_T' = f_1' + k'(f_2 - f_1) + k(f_2' - f_1')$$

At the boundaries of the transition region, the derivative of the step cubic is zero. Therefore,

$$f_T' = f_1' + k(f_2' - f_1')$$

At the lower boundary of the transition region, the function the step cubic is zero. Therefore,

$$f_T = f_1 + k(f_2 - f_1) = f_1$$

and,

$$f_T' = f_1' + k(f_2' - f_1') = f_2$$

At the upper boundary of the transition region, the function the step cubic is unity. Therefore,

$$f_T = f_1 + (f_2 - f_1) = f_2$$

and,

$$f_T' = f_1' + (f_2' - f_1') = f_2'$$

Thus, the requirements of continuity have been satisfied.

D.2 OPERATING REGION INDICATORS

200. In order to enhance the performance of the integrator, it has been found useful to employ a method of operating region indication.

A unique region indicator exists for every model which has more than one operating region. The component model checks to ensure that regions are not omitted between successive completed integration steps. If we allowed regions to be missed, the integrator would undoubtedly take longer to converge on the next integration step.

201. In general, a model will assign one of its integer constants (ICON) to represent the region indicator. Since the coding of the model will compare the current value of the region indicator to its previous value, it is evidently important that the region indicator is initialised on the first visit to the model calculation subroutine, i.e. time = 0.

202. Figure D.3 shows an operating characteristic of a typical component model (e.g. the pressure/flow characteristic of a pressure relief valve). Figure D.4 is a flow diagram which shows the region indication logic of the component model calculation subroutine. A section of the coding which represents this component is given in table D.1. It is recommended that this method of model construction and coding should be generally adopted (including the use of Fortran-77) due to its ease of understanding and also its brevity, two qualities rarely found in a single section of coding.

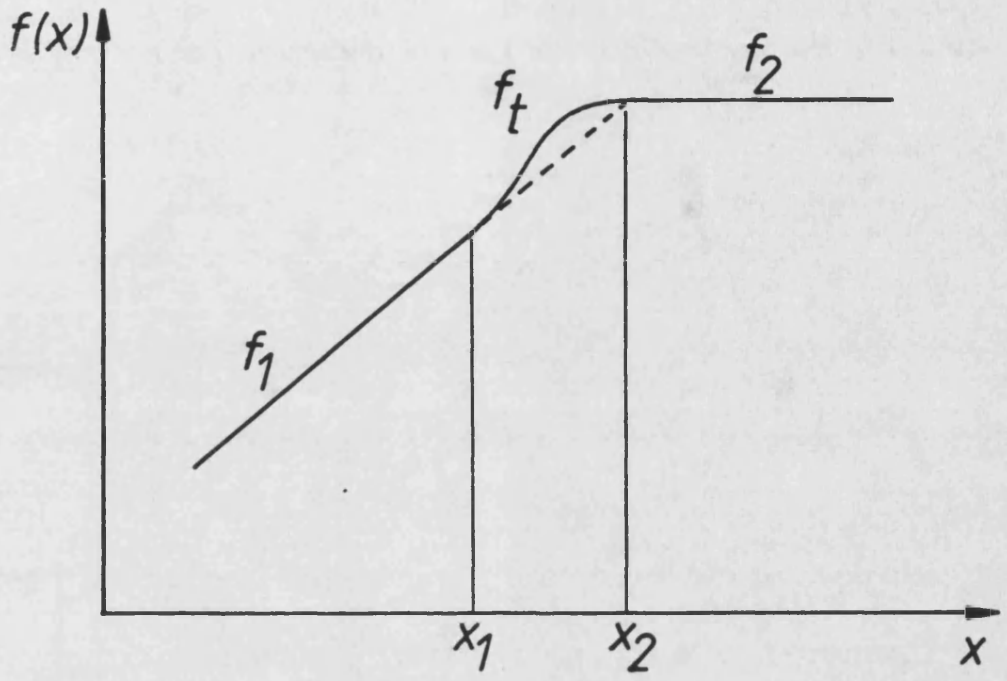


FIG D.1 A cubic transition function

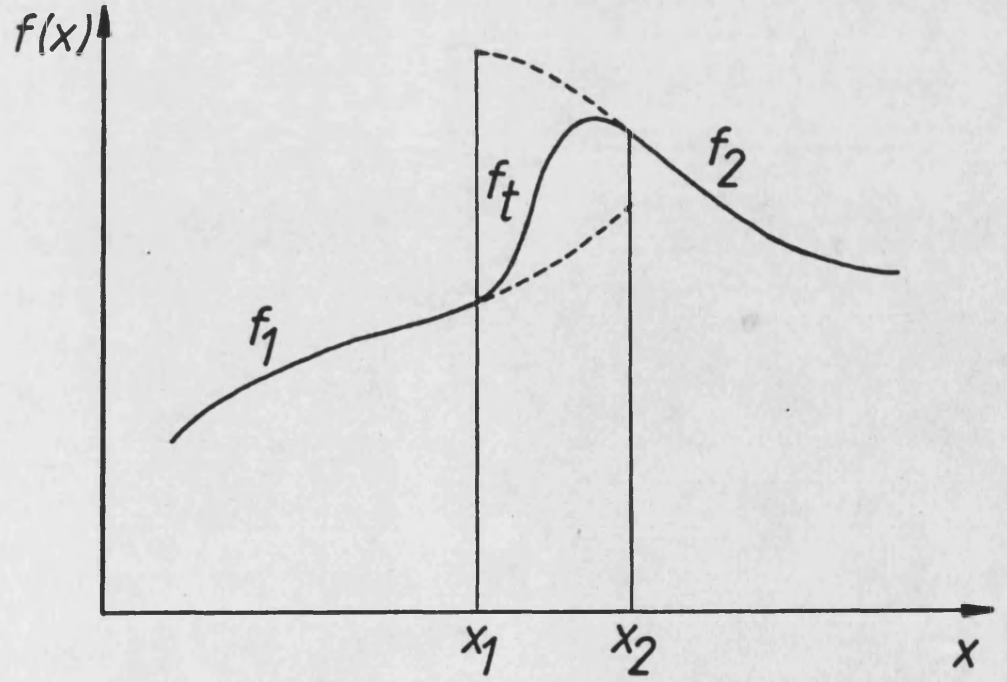


FIG. D.2 A high order transition function

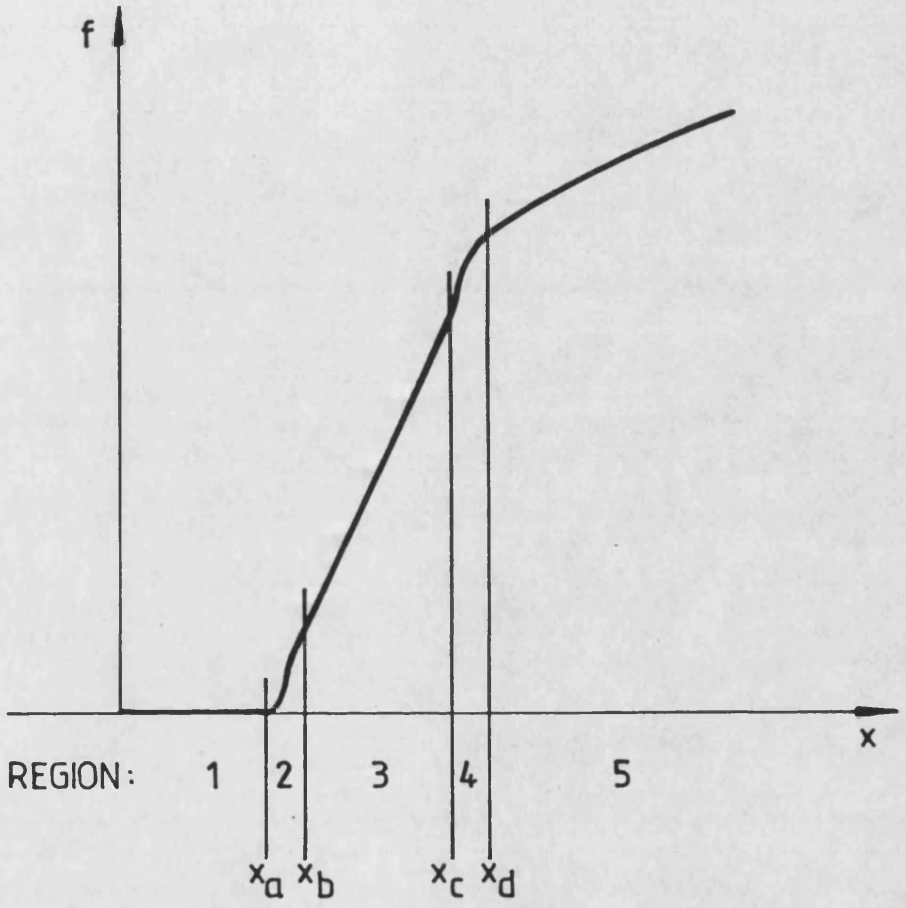


FIG. D.3 A typical operating characteristic

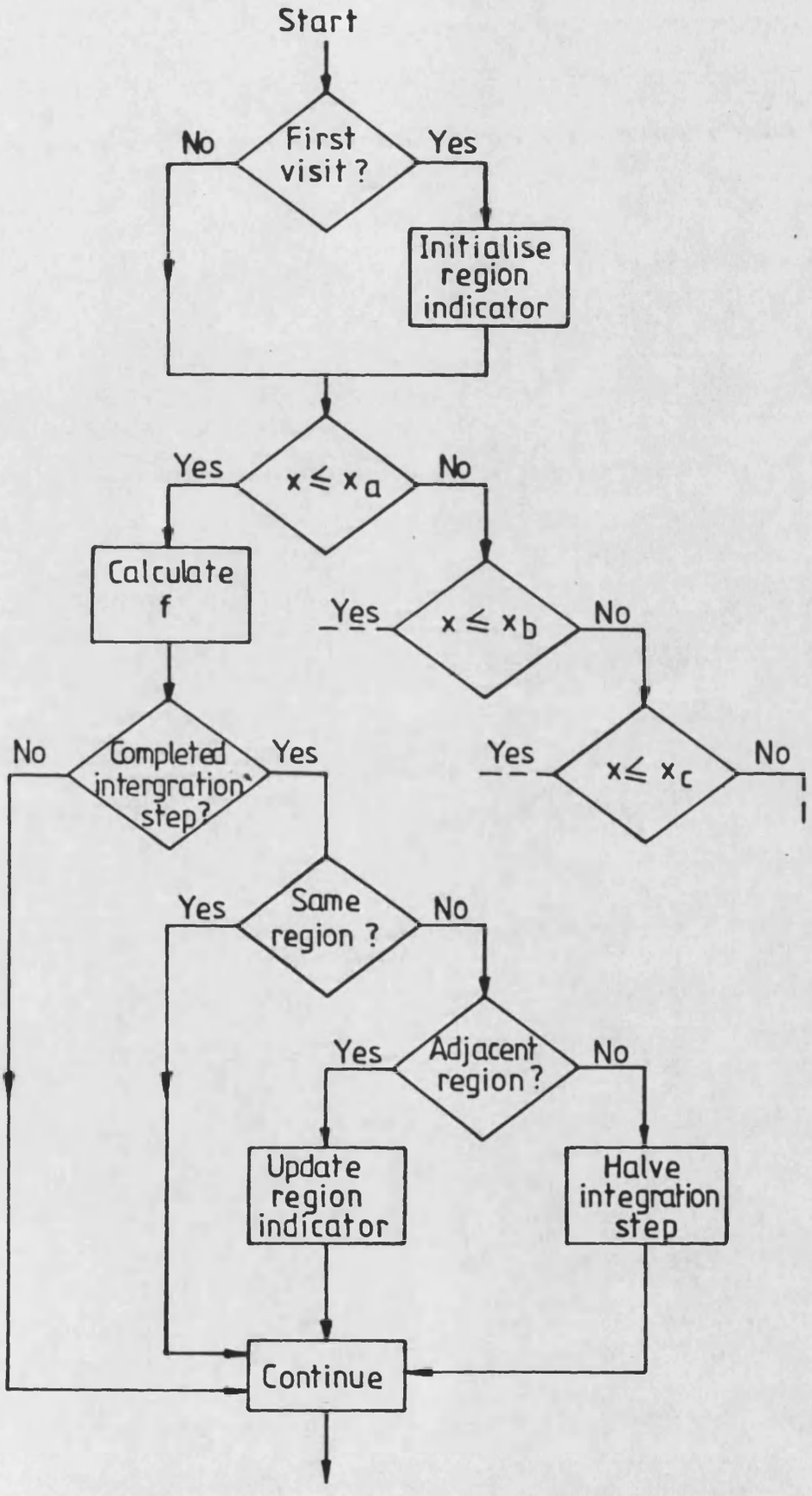


FIG. D.4 Flow diagram for the region indication logic

```
C ***** INITIALISE REGION INDICATOR IF FIRST VISIT
      IF(ICON(1).EQ.0)THEN
            ICON(1)=1
            IF(X.GE.XA)ICON(1)=2
            IF(X.GE.XB)ICON(1)=3
            IF(X.GE.XC)ICON(1)=4
            IF(X.GE.XD)ICON(1)=5
      END IF
C ***** REGION 1
      IF(X.LT.XA)THEN
            IREG=1
            F=0.0D0
C ***** REGION 2
      ELSE IF (X.LT.XB)THEN
            IREG=2
            F=CON(1)*X*X*X+CON(2)*X*X+CON(3)*X+CON(4)
C ***** REGION 3
      ELSE IF (X.LT.XC)THEN
            IREG=3
            F=CON(5)*(X-XA)
C ***** REGION 4
      ELSE IF (X.LT.XD)THEN
            IREG=4
            F=CON(6)*X*X*X+CON(7)*X*X+CON(8)*X+CON(9)
C ***** REGION 5
      ELSE
            IREG=5
            F=CON(10)*DSQRT(X)
      END IF
C ***** CHECK REGION HAS NOT BEEN OMITTED
      IF(LIMIT.EQ.2)THEN
            IF(IREG.NE.ICON(1))THEN
                  IF(IREG+1.EQ.ICON(1).OR.IREG-1.EQ.ICON(1))THEN
                        ICON(1)=IREG
                  ELSE
                        LIMIT=3
                  END IF
            END IF
      END IF
```

KEY: LIMIT Integer to define state of integrator
set to 2 for completed calculation step
set to 3 if model requires interval halving
IREG Temporary region indicator

TABLE D.1 THE CODING OF A CALCULATION SUBROUTINE REPRESENTING THE MODEL DESCRIBED BY FIGURE D.3

APPENDIX E - FUTURE PROJECTS

E.1 THE MODEL GENERATOR

100. The remaining facet of HASP which does not conform to the overall requirement of the package is the component model library. An aim of the package is to allow the user to simulate any hydraulic circuit in a simple manner. Evidently, the number of components in the library can never be totally comprehensive. Manufacturers of hydraulic components continually design new hardware. Also, there is the problem of the definition of the load on the hydraulic circuit to be simulated. No models can ever account for all the possibilities likely to arise. Therefore, it follows that the user will be forced to write new component models. Having accepted this fact, the aim of the developers should be to ensure that this process is as painless as possible.

101. Standardisation of modelling techniques, albeit of a flexible nature, has already been introduced. Coding layout, operating region indication and the manner in which discontinuities are dealt with have all been described. Two resulting effects are observed. Firstly, the production of draft models is reasonably quick provided the modeller writes his coding using the techniques described. Secondly, techniques such as the derivation of the cubic polynomial, the writing of region indication logic and the writing of entries into the component attributes data file COMPON.DAT tend to be

laborious and/or extremely error prone.

102. It is evident that an interface between the mathematical model and the computer coding is both required and feasible. The repetitive nature of the model coding lends itself to some automated procedure of coding production. Take, for example, an instantaneous characteristic model which has three primary operating regions. One would expect that the modellershould describe the nature of these three operating regions in terms of three simple equtaions. However, the modeller of HASP must also derive the cubic polynomials required as transitions between the three primary regions. He must also incorporate sufficient coding to ensure that no region is skipped on successive time steps. The resulting coding is thirty two lines long (if it is written in Fortran-77, uses the region indication logic described in Appendix D.200 and uses the dtandard utility CUBIC). It is evident that a model generator could write twenty nine of those lines automatically.

103. Figure E.1 is a schematic of the general structure of a feasible model generator. The figure shows the interaction between the user, the generator and the internal files. The generator would produce three files for every component model and would also automatically include an entry in the component attributes file COMPON.DAT using software similar to the COMPON.DAT editor described in Chapter 3, paras.352 to 367. As at present, a parametric definition subroutine and a calculation subroutine would be produced. In addition, a file called component.ATT would be produced to store

all the attributes of the model. This file would contain sufficient information to allow the associated component model to be regenerated. This allows the user to edit existing models that had been created using the model generator.

104. Figure E.2 shows the structure of the model generator in terms of its constituent routines and table E.1 lists the tasks of each of these segments.

E.2 ONLINE CIRCUIT DESCRIPTION

200. Preparation of the data to describe a circuit to be simulated can also be a laborious and time consuming task, a procedure which may be gain from some online assistance in addition to the HELP utility described in Chapter 3, paras.311 to 312. The computer can aid by allowing the user to construct linking diagrams on the screen. The user would simply move the cursor in order to position a component on the screen. The facility would then analyse the diagram that has been constructed, automatically number the links and produce the system description data file required by the program generator.

201. It is likely that a program of this kind would require a specific type of terminal. The use of joysticks and light pens for cursor movement should be avoided. Also the graphics facility employed should be of a widely available nature. The program developed to show the feasibility of the utility described above employs the special character and line drawing set available on all

computer terminals which emulate the capabilities of the popular and widely available Digital VT100 series.

202. Figures E.3 to E.11 show a typical entering and positioning process. Initially, the user is invited to select a component or to type ? if assistance on selection is required. Having done so, he is then instructed to position the cursor at some point on the screen then press <ENTER> (figure E.3). The component attributes data file COMPON.DAT is interrogated in order to produce a block which represents the component required accounting for the number of external links, internal links and signals (Figure E.4). The user is again asked to enter a component name or ? for help. Having defined two or more component models, he is also given the option to type L in order to link adjacent models (figure E.5). Assuming he decides to link the two models shown, he must position the cursor near the two ports to be connected at press <ENTER> on each occasion (figure E.6). The component model blocks are then redrawn with the external link in position and automatically numbered (figure E.7). This information is also stored in character and integer arrays which will produce the system data.

203. Should the user type ? in order to summon assistance on model selection, the program will list all classes of models and invite the user to select a particular class (figure E.8). Having done so, the component attributes data file COMPON.DAT is searched for all components in that particular class and this information listed on the terminal (figure E.9). The user may then select a specific

component in order to display a short introduction and useful information such as the number and type of links (figure E.10). The coding used to complete this part of the graphical input utility is identical to the routine PGHELP described in paras.311 to 312.

204. Figure E.11 shows a typical completed linking diagram. With bigger circuits, the user must define sections (termed pages) of the circuit at a time then assemble the pages to form a complete linking diagram.

205. Figure E.12 shows a schematic of the structure of the graphical input utility. It should be emphasised that the program remains partly unwritten. The terminal setup and the setup of all US ASCII and DEC line drawing characters has been completed. Also, sections of the graphical construction and cursor movement routines are complete. These routines do not produce a working package but sufficient coding has been developed to indicate the feasibility of the program and the manner in which it should be written.

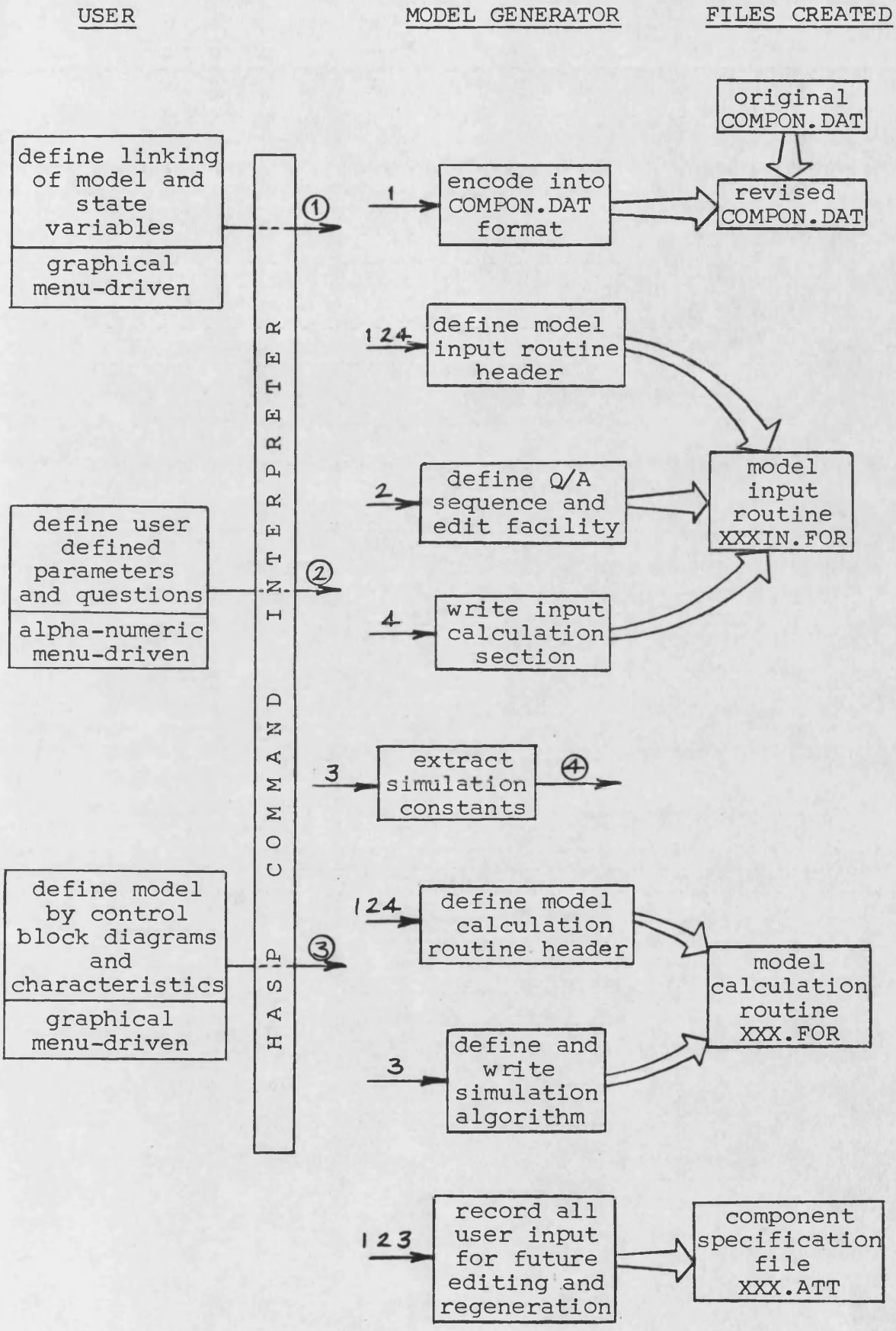


FIG. E.1 The structure of the Model Generator

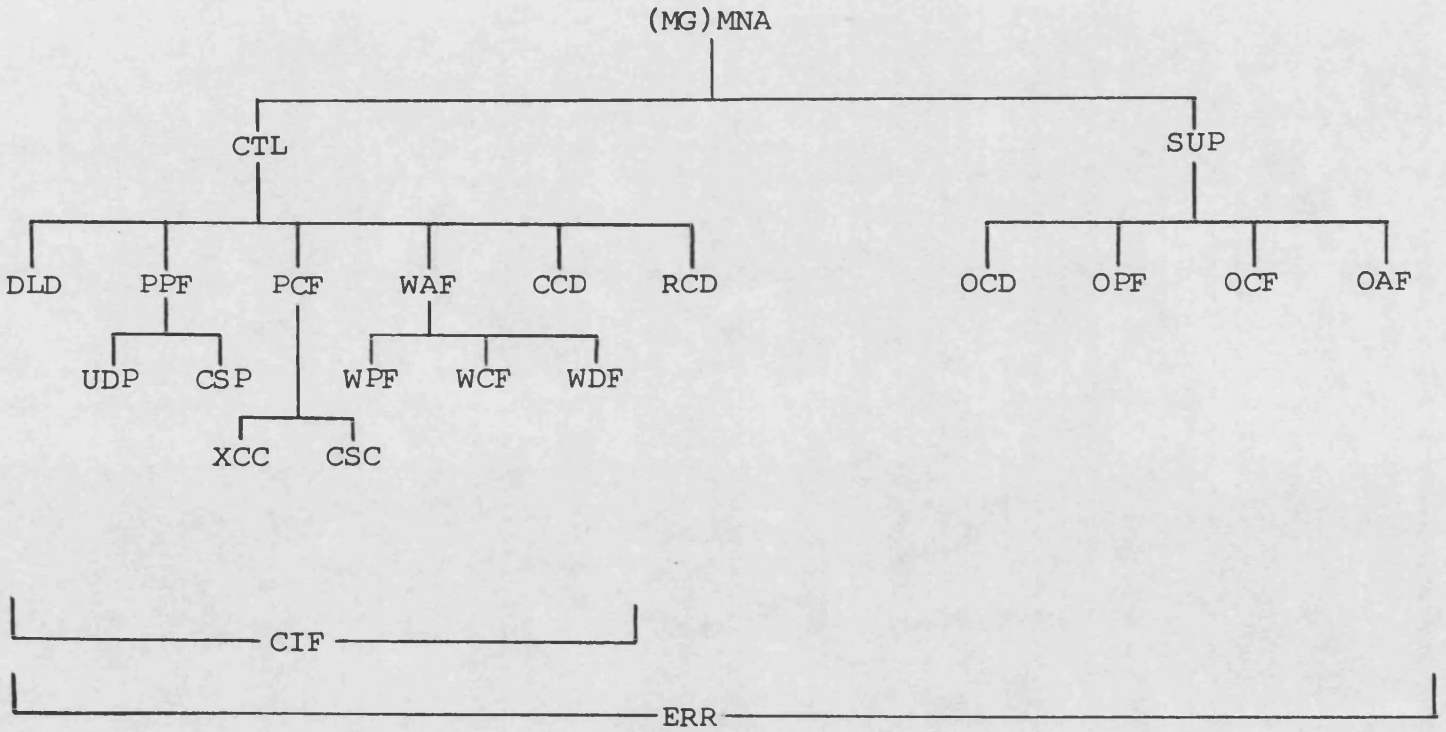


FIG. E.2 Call tree of the Model Generator

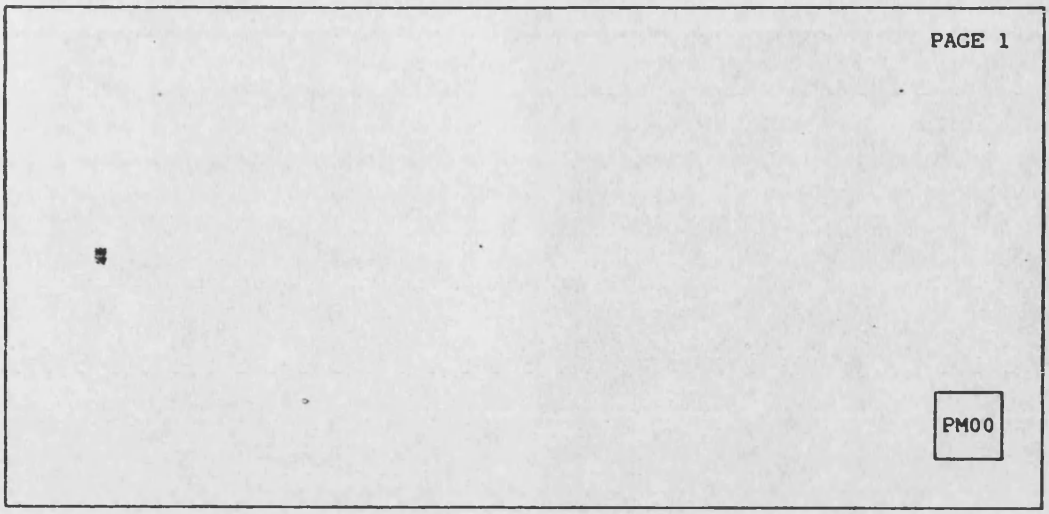


FIG. E.3 Online circuit description - Positioning first component

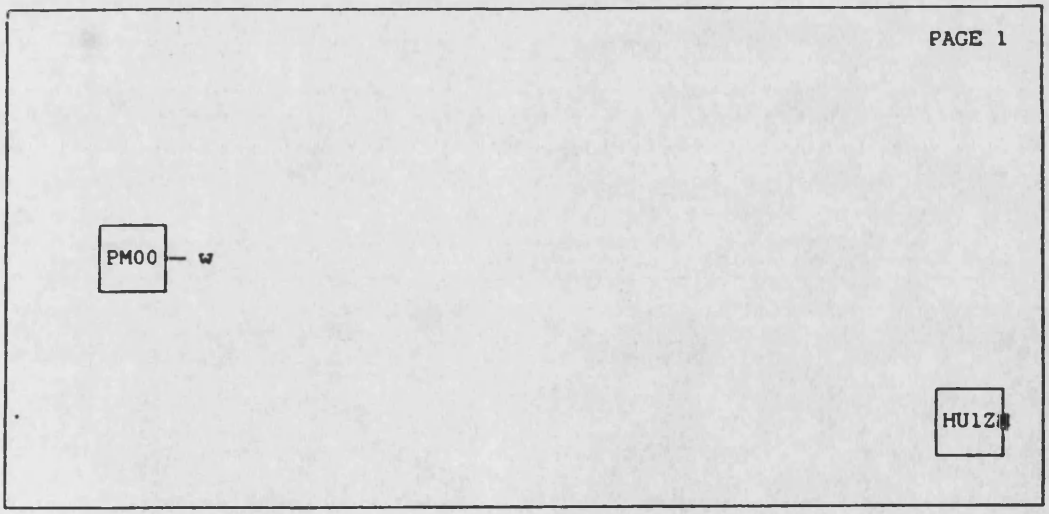
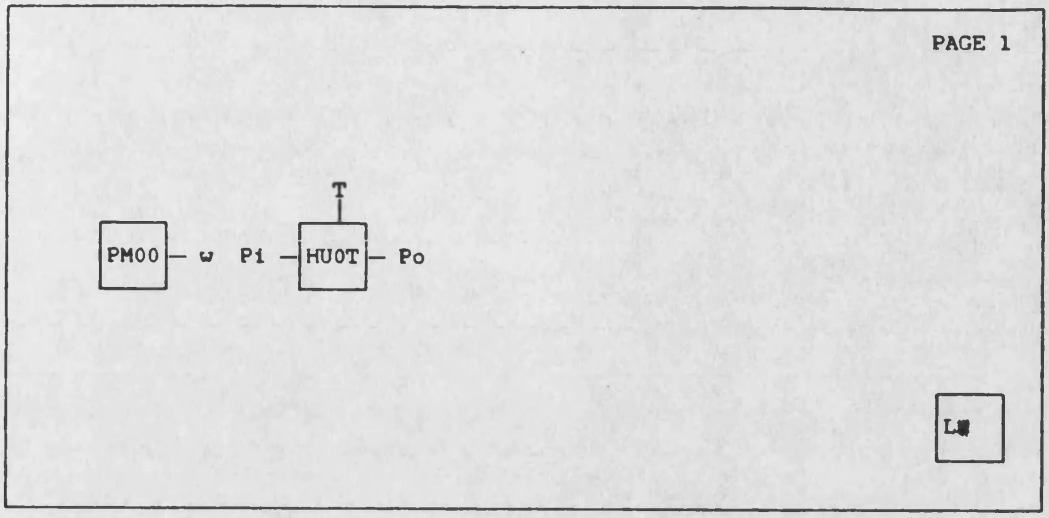
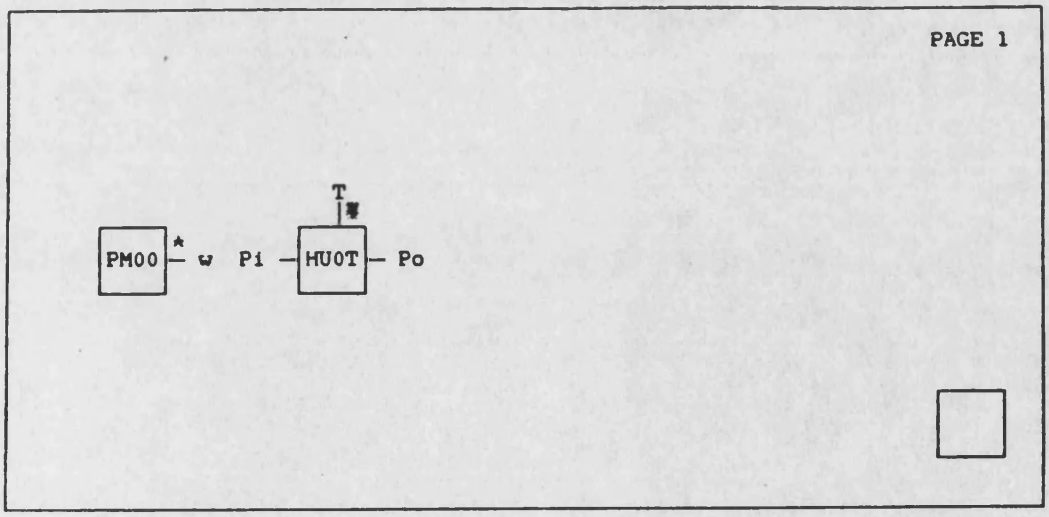


FIG. E.4 Online circuit description - Definition of the second component



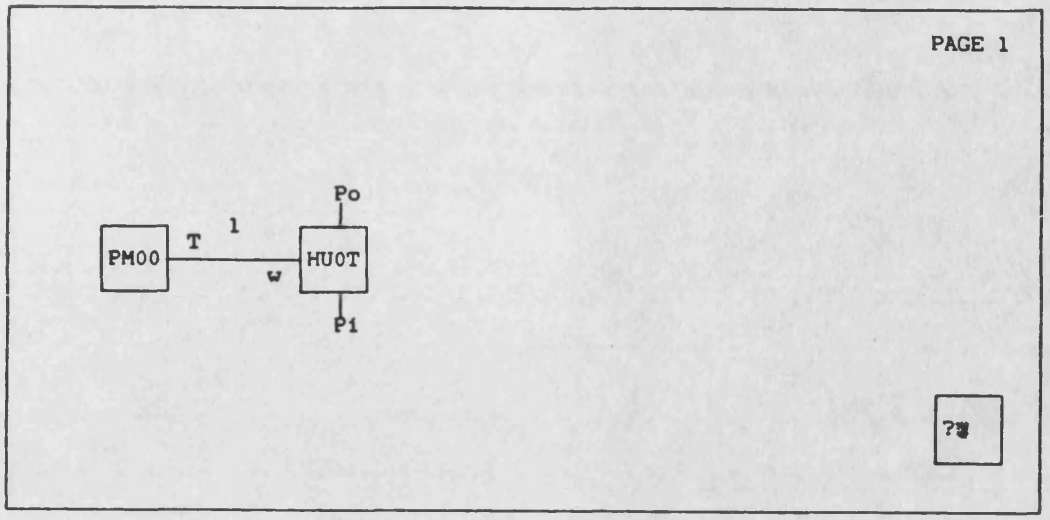
Type mnemonic of component, L to link or ? for help

FIG. E.5 Online circuit description -
Request linking of the two components



Position cursor at both ends of link and press <ENTER>

FIG. E.6 Online circuit description -
Definition of component linking



Type mnemonic of component, L to link or ? for help

FIG. E.7 Online circuit description - Request for assistance

CLASS LIST:	AC	represents	Accumulators	HELP 1
	AL		Actuator/loads	
	AM		Electronic amplifiers	
	DC		Directional control valves	
	DE(F)		Duty cycles - Effort (flow) sources	
	FC		Flow control valves	
	HE		Heat exchangers	
	HU		Pump/motor units	
	LR		Rotary loads	
	PC		Pressure control valves	
	PI		Pipes	
	TK		Hydraulic tanks	
CLASS: PU				

Type class for which information is required (press <RETURN> to proceed)

FIG. E.8 Online circuit description - Request for information on pump models

```
PIPE COMPONENT MODELS:                                        HELP 2

PI03   Frictionless pipe with cavitation - User defines total volume
PI05   Frictionless pipe with cavitation - User defines pipe dimensions
PI06   Frictionless pipe with cavitation and volume transfer
PI5T   Experimental version of PI06
PI2Z   Friction pipe - slightly modernised version of old PIP2
PIPE   Frictionless pipe with cavitation - simplified version for demos.



  PI05
```

Type component for which information is required (press <RETURN> to proceed)

FIG. E.9 Online circuit description - Request for information on model PI05

```
                                                              HELP 3
PI05   Frictionless pipe with cavitation - User supplies pipe dimensions

PI05 is a dynamic pipe model which determines the fluid pressure level
when provided with flowrates by adjacent models. The model accounts for
air release when the pipe pressure is low by calculating the proportion
by volume of the air released from solution and the effect of this free
air on the bulk modulus of the fluid. This effect will adequately
represent cavitation in the majority of cases.

     Number of external links:   1 to 8        Inputs:   FFFFFFFF
     Number of internal links:   none          Outputs:  EEEEEEEE
     Number of signals:          none
     Number of state variables:  1
```

Press <RETURN> to proceed

FIG. E.10 Online circuit description - The information provided on model PI05

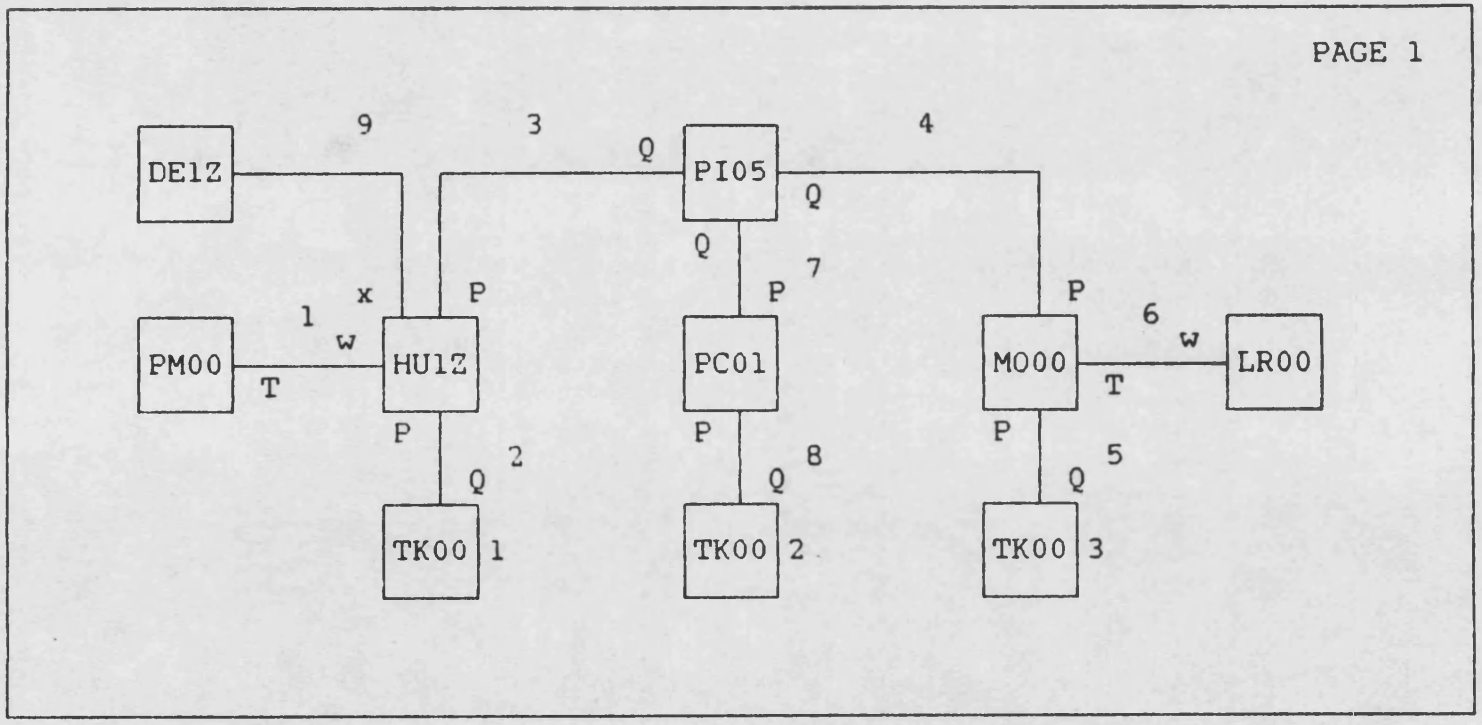


FIG. E.11 Online circuit description - A completed linking diagram

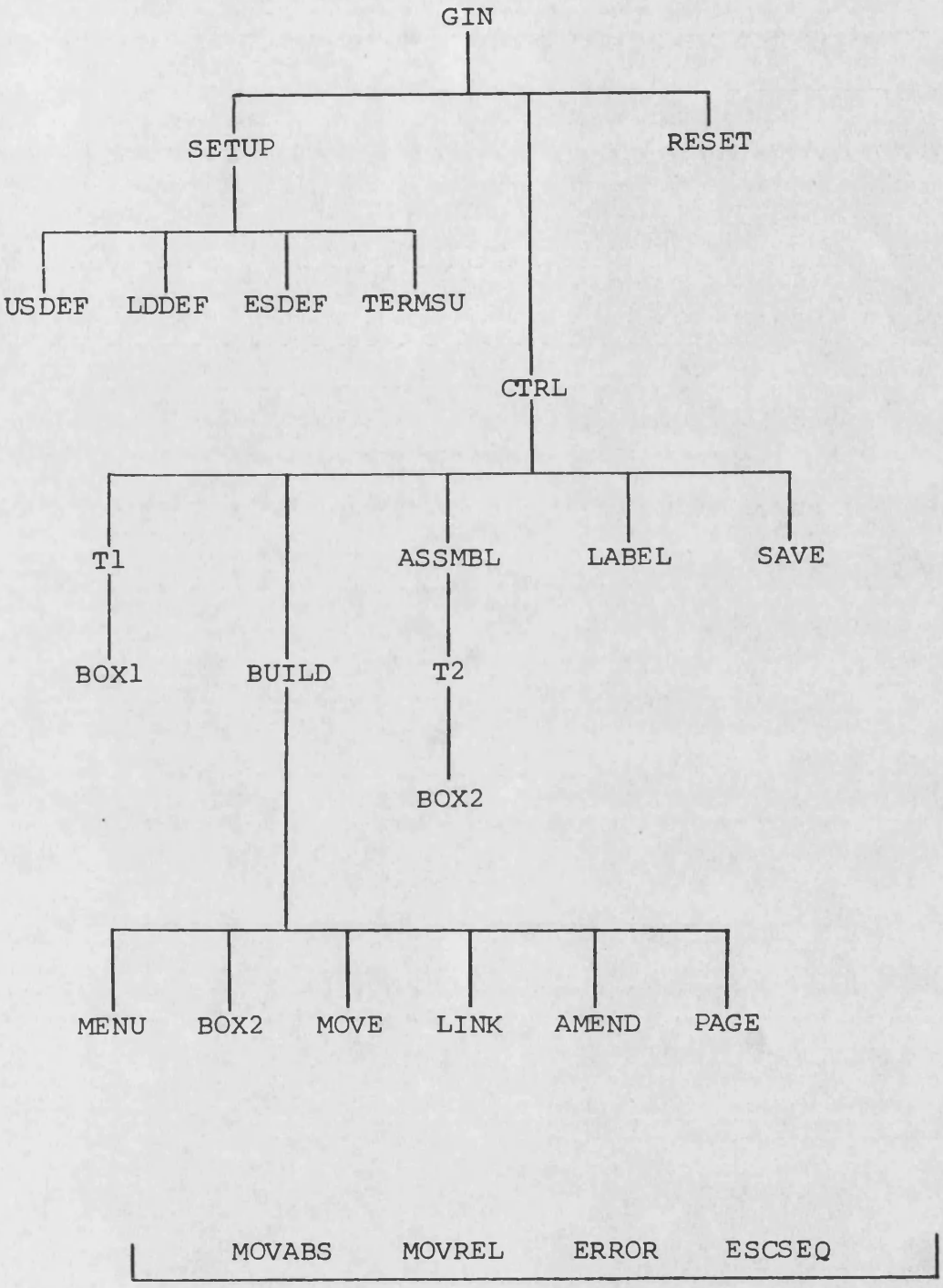


FIG. E.12 Call tree of the Graphical Input facility

NAME	TASK
MGCCD	Create a new component attributes file COMPON.DAT
MGCIF	Translates input strings into meaningful equations
MGCSP	Defines simulation constants based on user defined parameters
MGCTL	Controls the utility routines
MGCSC	Defines cubic smoothing and region indication
MGDLD	Defines the model linking diagram
MGERR	Writes error diagnostics
MGMNA	The main segment
MGOAF	Opens the information file ****.ATT
MGOCD	Opens the compenent attibutes file COMPON.DAT
MGOCF	Opens the calculation file ****.FOR
MGOPF	Opens the parametric definition file ****IN.FOR
MGPCF	Defines the calculation subroutine
MGPPF	Defines the parametric definition subroutine
MGRCD	Revises an existing entry in COMPON.DAT
MGSUP	Controls the setup of the files
MGUDP	Creates a list of user defined parameters
MGWAF	Controls the writing of all files
MGWCF	Writes the calculation subroutine
MGWDF	Writes the component definition file
MGWPF	Writes the parametric definition subroutine
MGXCC	Extracts simulation constants from the algorithm

TABLE E.1 The tasks of the model generator routines

NAME	TASK
AMEND	Amends an existing diagram
ASSMBL	Assembles individual pages of the diagram
BOX1	Draws the large screen box
BOX2	Draws the small component box
BUILD	Builds individual pages of the diagram
CTRL	Controls the utility routines
ESCSEQ	Writes an escape sequence to the terminal
ERROR	Writes error diagnostics
ESDEF	Defines the standard escape sequences
GIN	The main segment
LABEL	Labels the links
LDDEF	Defines the VT100 special character set (line drawing)
LINK	Links the models as requested by the user
MENU	Gives component information and instructions
MOVABS	Moves the current position by an absolute value
MOVE	Moves the component to a user requested position
MOVREL	Moves the current cursor position by a relative value
PAGE	Sets up a new page for the diagram
RESET	Resets the terminal to its normal mode
SAVE	Creates linking data for the program generator
SETUP	Controls the setup of constants and the terminal
TERMSU	Sets up the terminal in line drawing mode
T1	Draws the title box
T2	Draws assembled pages of the diagram
USDEF	Defines US ASCII characters

TABLE E.2 The tasks of the graphical input routines