University of Sheffield

Department of Control Engineering

An Introduction to The Design of a C.A.D. System for Control System Design

S. Bennett

Research Report No. 8

December 1971

Summary

The user requirements for a computer-aided-design facility are outlined and an introduction to the ideas of program and data structures is presented. The facilities available on the COMPAC 4020 are discussed and related to the requirements for a C.A.D. system. The problems of implementation are considered with particular reference to providing a system which is immediately useful and yet capable of continual enhancement. A set of rules for guidance in writing programs suitable for inclusion in such a system is given.

The frequency response design techniques are used to illustrate the development of the program and data structures suitable for a C.A.D. facility.

1. Introduction

The term "computer aided design" (C.A.D.) is often applied to a wide range of computer programs and techniques, and, in a sense, any program used to assist with design is a C.A.D. program. It is more normal, however, to restrict the term C.A.D. to systems where the user interacts 'on-line' with a suite of design programs and this is the usage adopted in the report.

The purpose of the report is to present, for discussion, a number of ideas relating to the development of a C.A.D. scheme for control system design. The first part of the report discusses the general approach to C.A.D. systems in terms of the program and data structures required, while the later parts discuss how these requirements relate to the facilities available on the CONPAC 4020 on which the system is to operate.

2. Basic Requirements for a C.A.D. System

Users of C.A.D. systems may have widely differing backgrounds and experience and therefore different requirements. The novice will require step by step guidance and will probably wish to follow the design procedure built into the program; the experienced user does not require detailed guidance, and will probably wish to follow his own design procedures. There is a need, therefore, for a flexible and versatile system which can be used in a simple or complex way, depending on the experience of the user.[1,2]

To achieve flexibility the C.A.D. system must be divided into a set of modules (program segments) and simple methods of inter-connecting the modules provided. The beginner need not be aware of the modular structure, since the system should have available supervisory routines to control module inter-connection; the experienced user should, however, be able to make connections as required.[3] The modular form of structure is of particular importance to organisations in which methods are continually changing, since it makes modification of the system simple; it also enables a user to easily add special segments to the system.[4,10] In addition to flexibility, an important feature for the regular user is the provision of user files enabling a user to retain from session to session a partial or complete design solution; the system will therefore require file handling program segments.

In many C.A.D. systems there is a tendency to overwhelm the user with 'conversational' output. The user may need guidance, but should have the option to suppress this guidance; 'conversation' which cannot be suppressed should be kept brief. It is preferable in many cases to reduce the amount of 'conversation' by providing the user with a written instruction sheet.

A problem which has to be faced by all designers of C.A.D. systems is that of finding a balance between the facilities offered and the complexity of the system. The solution, or rather the avoidance, of this problem lies in finding a program structure which is basically simple, but can be made to offer extended facilities by the ingenious or experienced user.

3. Program Structure

The detail of the program structure, e.g., the function and size of the various modules, will depend on the particular application and on the characteristics of the computer on which the scheme is to be implemented. The general form of the structure will, however, conform to a modular and hierarchical pattern as shown in figs. 1a and 1b (page 4).

Figure 1a shows the basic modules of any C.A.D. system, a common data area - the 'data base' - which is shared between all the modules, the algorithm or algorithms, and the input and output sections, the running and interconnection of these modules being controlled by an 'executive' or 'supervisor' module. This structure is adequate for a simple system containing one or two algorithms, but as the system size increases, further divisions are required to avoid increasing the complexity of the executive program.

This extension is shown in fig 1b and is essentially an increase in the number of control or supervisor segments and a division of these segments between a number of priority levels. The number of levels chosen is in a sense arbitrary (fig 1b shows 5 levels), the important feature of the system is that control, when a program segment terminates, always passes to a higher level, rather than to another named program segment, as would happen if the structure was arranged as a ring or chained system. For example, if segment INPUT1 (fig 1b) were running, on termination it would pass control to the active segment at level 3, it would not, however, know which of the segments sharing level 3 was active, or if it had actually been called from level 2. In a chained or ring structure this segment would have passed control to a preset segment, say ALGORITHM 1, which would in turn transfer control to another predetermined segment.

The hierarchical structure does not preclude the use of chained structures within it, and in operation in its simplest form, it simply inserts an additional segment between each segment in the chain. There are obvious disadvantages in this technique - the additional time overheads in running an extra segment - the advantages are in increased flexibility and ease of use. With the chained or ring structure the links have to be specified before the problem is run. With the hierarchical system the links can be specified as the problem is running, i.e., the user can interact with the system at each stage in the calculation.

3.1 Executive, Supervisor and Control Segments

These routines all perform control functions and the different names are used mainly to distinguish the levels at which they operate.

The executive segment normally performs functions associated with setting up the system for a given user. Typically, the user logs in, i.e., identifies himself to the system, the executive then arranges for the appropriate files for that user to be loaded. After loading the files it awaits a command from the user and on receiving the command takes the appropriate action, probably the turning on of a supervisor segment.

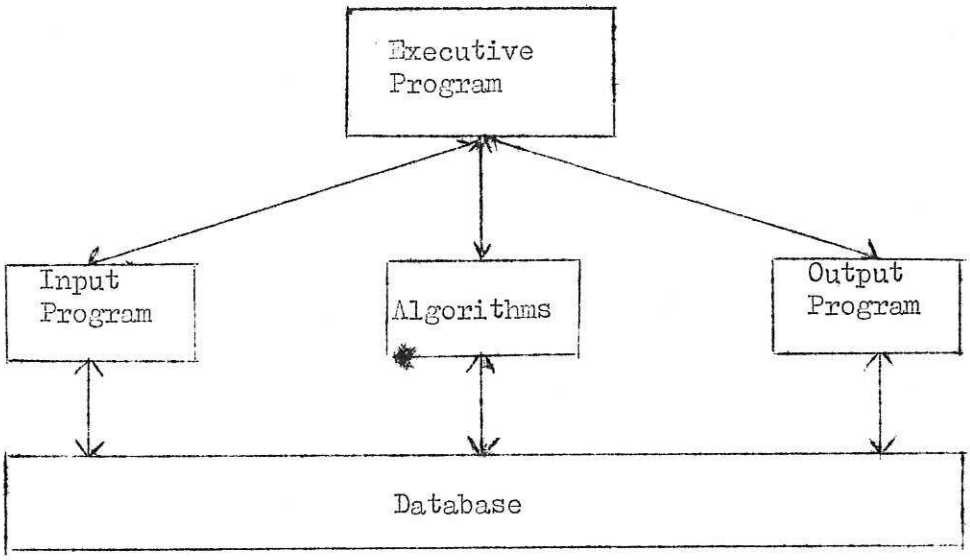


FIG.1a

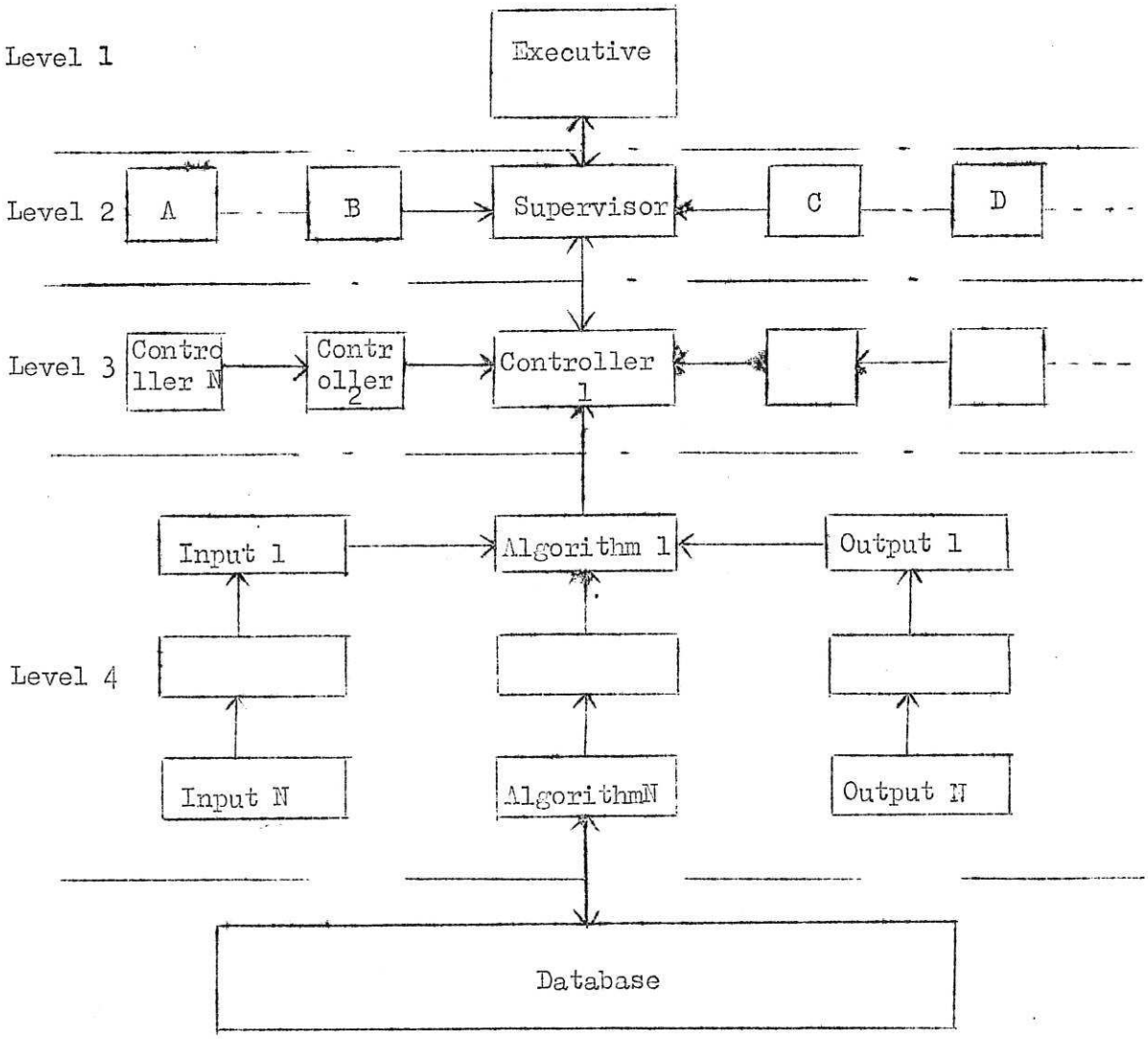


FIG.1b

PROGRAM STRUCTURE

The system may have one or more supervisor segments. A basic supervisor will, say, accept commands from the user and turn on the appropriate control or working segments, thereby enabling the user sitting at a teletype to direct the system through various stages of the design. Additional supervisors can be incorporated into the system to direct the system, i.e., the user can have his own supervisor which controls the running of the control and working segments and hence instead of typing in the various directives the user demands his own supervisor.

The control segments are used for two purposes, (a) to control the running of algorithms or input/output routines, which have been split into several segments for the convenience of the system rather than the user; (b) to link together work segments which are frequently required in a preset sequence. The control segments will be related to design techniques commonly used.

3.2 Work' Segments

The work segments run at the lowest level in system and are the segments which perform the calculation or input and output, i.e., they are similar to subroutines in normal batch processing computer systems. The function of the segment should not necessarily be related to a given design technique but should be, if possible, a function common to several techniques.

3.3 Database

With any form of program segmentation the problems of sharing data between segments arises. The use of large areas of COMMON in core to solve this problem is expensive and particularly wasteful in an interactive situation where a user, whilst thinking of the next stage in the design may be occupying several thousand words of core. To avoid the need for large areas of COMMON core some form of file structure is necessary.

In this type of structure the data for the problem is held on backing store in a number of files, a directory to the files is held either as a master file, or in a COMMON area of core. A program can, therefore, obtain the data it requires by reference to the file directory and hence data is only held in core for the time during which it is being manipulated.

A hierarchical program structure as outlined above is complicated, and it may be argued too complicated, its advantages, however, are that it provides the basis for a versatile and expandable design system. The structure involves the system in repeated transfers from backing store to core store, with a consequent increase in running time, a situation which would not be tolerable in a batch processing system. But in an interactive multi-programming system where human response is slow compared to transfer times, reduction in average core utilization is more important than a reduction in computation time.

4. A.C.A.D. System on the CONPAC 4020

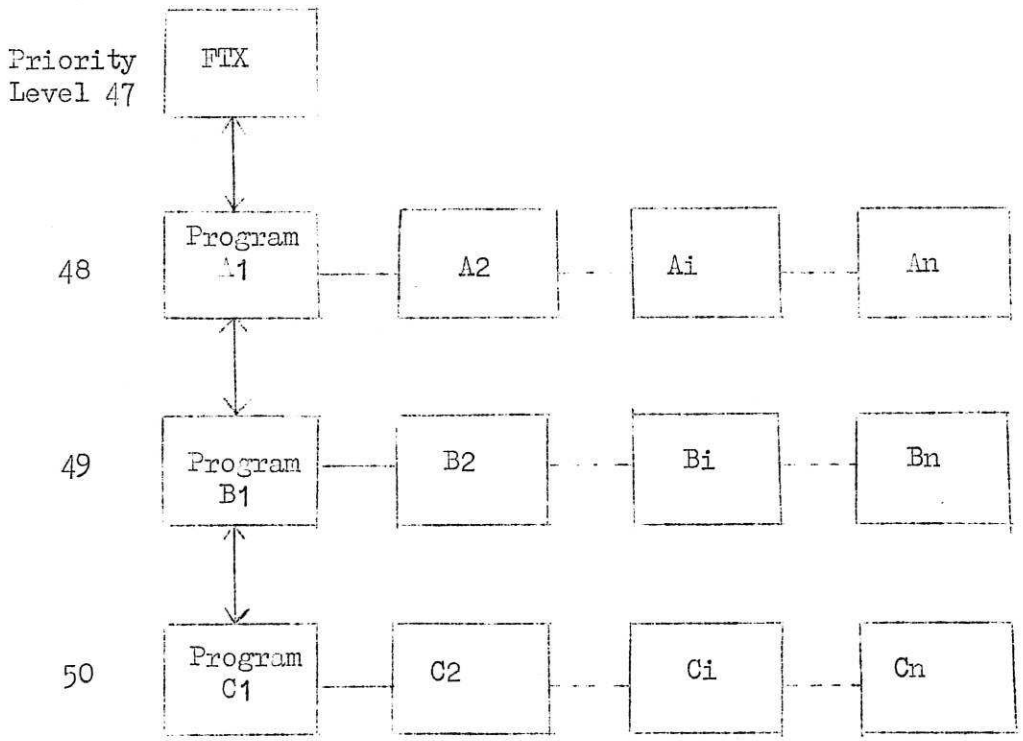
4.1 RTMOS-CONPAC 4020 Operating System

The RTMOS (Real Time Multiprogramming Operating System) as implemented on the CONPAC 4020 computer in the Department, has provision for 50 distinct priority levels of real time programs. A total of 206 active real time programs can share the 50 priority levels.

The first 50 programs are held in the real time library, the rest are held in the extended real time library. Programs in the extended real time library are run by using the RT LINKAGE subroutine. A program in the real time library running at, say, level 30, can, by using RT LINKAGE, replace itself by a program from the extended library, or run at another priority level any program from the extended library. In this way, users can create simple executive programs and run them at any priority level.

The real time programs run under the control of RTX (Real Time eXecutive) and one real time program, running at level 47, is FTX (Free Time eXecutive). This program controls programs running at levels 48,49 and 50 and provides facilities for on-line compilation and testing of programs and for background (off-line) jobs, these facilities form the CONPAC 4020 "Free Time System."

The structure of the Free Time System is shown in fig.2. Any program from the free time library can be run by FTX at level 48, by using the F.T. LINKAGE subroutine, this program, A1 say, can replace itself at level 48 by



FREE TIME SYSTEM FIG.2

any other free time program Ai(i=2,n),Bi(i=1,n), Ci(i=1n) or can run any free time program at priority level 49. Similarly any program running at priority level 49 can replace itself at level 49 or can run a program at level 50. Programs running at level 50 can only replace themselves. All programs on termination return control to the program running at the next highest level.

4.2 C.A.D. System Structure on a CONPAC 4020

The structure of the free time system is similar to the desired structure of a C.A.D. system outlined in section 3. It is easy to see how the supervisor, control and work segments could run at levels 48, 49 and 50 respectively. The linkage between the program segments can be handled using the FT LINKAGE subroutine. An example of a simple supervisor for use with the free time system is given in appendix 1.

There are a number of limitations to using the free time system for a C.A.D. system, one is that the present operating system limits the free time library to 64 programs with a total size of 40,000 words. Since the UMIST C.A.D. system occupies approximately 350,000 words, it is apparent that this size restriction would be a serious limitation of the system. The major limitation and disadvantage of using the free time system, is that it is limited to only one user at any one time. This means that whilst a design is in progress no compilation or testing of programs can take place.

These limitations can be avoided by building the C.A.D. system as part of the real time system, since this makes available approximately 150 program segments and 230,000 words of storage. Also, whilst a user is carrying out a design, background jobs (compilation, etc.) can be taking place under the control of the free time system.

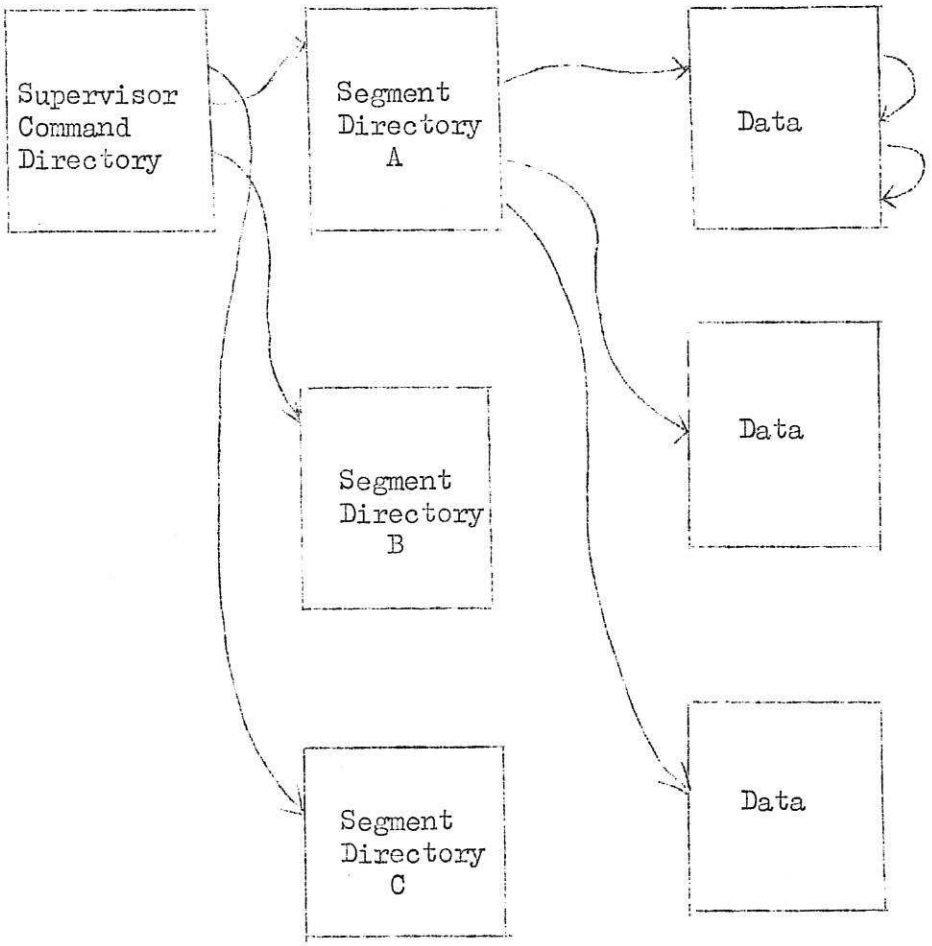
Although it is not as readily apparent as with the free time system, the C.A.D. structure outlined in section 3 can be implemented using the real time system. An example of a simple supervisor program for a C.A.D. system is given in appendix 2.

4.3 Structure of the Database

The organisation of the database presents, perhaps, the major problem in any C.A.D. system, and as was discussed in section 3.3, this organisation will have to be based on files and sub-files. The CONPAC operating system has facilities for manipulating data files, these are referred to in GE nomenclature as data tables and are held in a data table library. The library has provision for 128 tables with a total size of 100,000 words.

The data can probably be best organised in the way shown in figure 3.(page8). Associated with the supervisor programs is a directory containing the supervisor commands and pointers to the tables containing the data required for the segment. There will not necessarily be a directory for each segment, since many segments will operate on similar sets of data and will, therefore, share a common directory.

This type of data structure is flexible and by adding additional directories it will be possible to provide each user with his own data file and will enable him to keep data relevant to several design attempts.



FILE STRUCTURE FIG.3

Typically a user will require some 10 - 15 K words of storage for this purpose [4] and since on the CONPAC disc file there are approximately 100 K words of unused storage the system should be capable of handling 8 - 10 users. A user will be able during the 'login' procedure [6] to overlay the contents of his own data files into the working files.

The detail data structure is, of course, dependent on an analysis of the particular type of problem being handled [3] and different problems may be suited best by different types of data structure, e.g., in control system problems frequency domain and time domain work may benefit from a different data structure. The structure suggested above is chosen because it is suited to the hardware and software of the CONPAC 4020 and it is not necessarily ideal from the point of view of the application, it will, however, be implicit in any structure adopted. It is possible to order the data for the problem in a manner suited to the problem, however, underneath this structure will be the basic structure and any major deviation from this structure will increase overheads [5]. It is, therefore, possible if the application warrants it, to impose a different structure on top of the existing structure. The main programs in the system will, however, communicate with the basic data structure, because if an on-line system is to grow there must be some standardisation. For the CONPAC system it is convenient to use data tables as files since part of the file handling requirements are then already available in the operating system.

An example of a file system suitable for use on the CONPAC 4020 is shown in appendix 3.

5. Implementation of a C.A.D. system for control system design

Although the design and implementation of a comprehensive design facility is a lengthy task, by careful planning such a facility can be built up slowly and be useful during the period in which it is being built. The major requirement for this approach, as for any multi-user on-line system, is program compatibility. It is not necessary for all programs to inter-relate, or to be able to operate on any given set of data, but each program should have access to a public directory (or directories) which contain information about the program links, data requirements, etc. [7] For example, when using frequency response techniques, a program may require data in polar form, but the data is at present in cartesian form, it is convenient to the user if the program checks the public directory, discovers that the data is in cartesian form and

automatically runs a conversion program. Furthermore, once conversion programs are written, it does not matter which form of data additional programs require.

The full specification for compatibility of programs will only emerge as each field of application is studied. The important first steps are, however, to obtain the various program segments in a form in which they can be used to build up a system. In order for this to happen programmers must agree to follow the few simple rules listed below:

(i) Split the program into functional segments.

(ii) Keep input/output operations separate from calculation.

(iii) Do not allocate array storage inside a subroutine (temporary working area can be obtained by using redimension subroutine).

(iv) Keep conversation to a minimum.

(v) Make full use of existing segments.

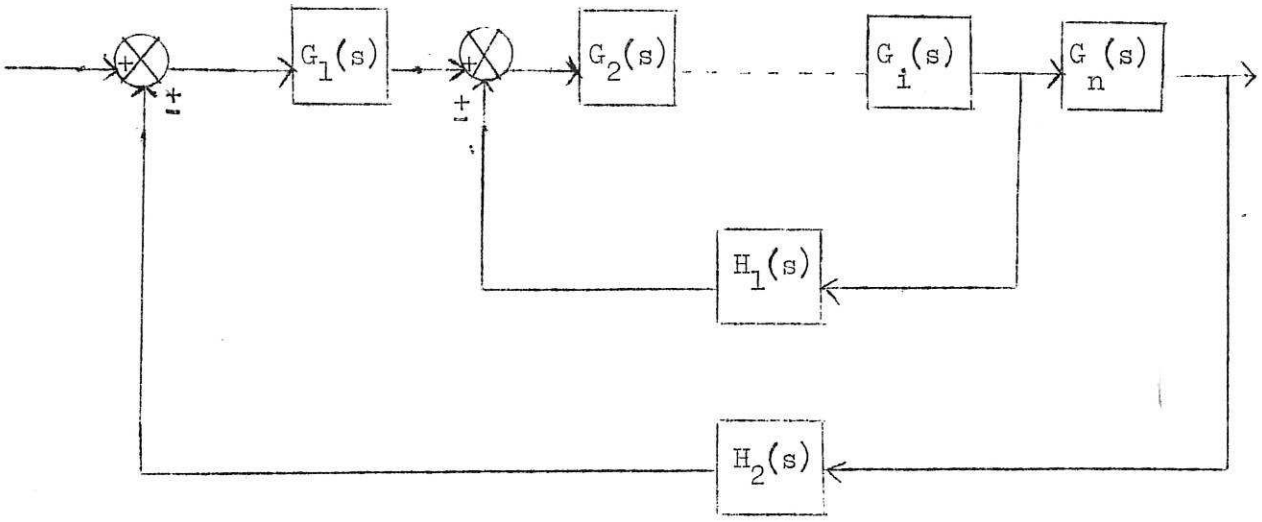
(vi) Provide a brief written description of each program segment.

Rule (i) is a basic rule of programming, small segments are easier to ' debug' and test, and a large library of well-proven, general purpose, simple programs is of much more use than a collection of unwieldy special programs. However, small functional segments for calculation are useless if they contain input/output statements, since another user may wish to use the segment in an iterative program, hence rule (ii). There is a tendency, in conversational programming, to overwhelm the user with instructions, this is bad because (a) the conversation occupies storage and (b) the user rapidly becomes bored by the repeated instructions. It is reasonable to expect a user to follow a few simple written instructions.

5.1 Development of a set of programs for control system design using frequency response techniques

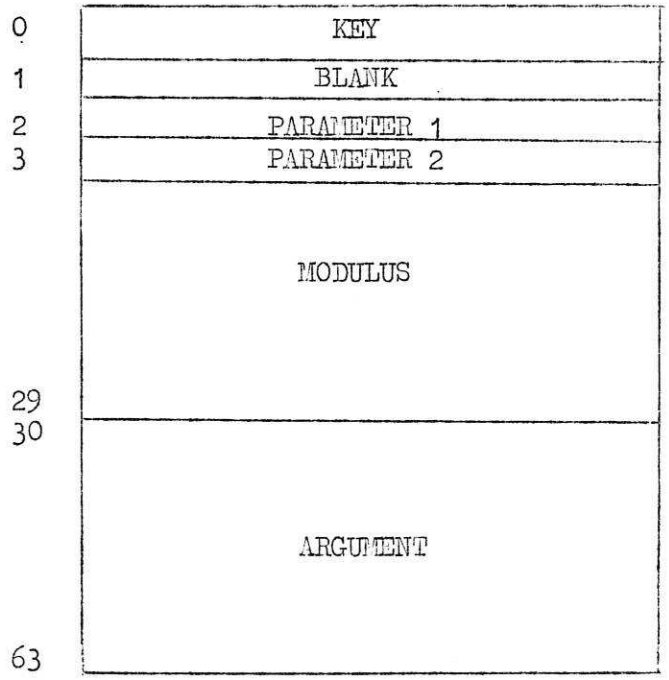
As an illustration of how a design facility can be developed, control system design using the techniques of Bode, Nyquist and inverse Nyquist will be considered.

It is assumed that the starting point for the design is a system which can be represented as shown in figure 4. The input to the programs will be the transfer functions of the various blocks and it is required to find the necessary parallel or series compensating elements to satisfy some performance criterian.



General Block Diagram of System FIG.4.

Since the system has a block structure and associated with each block is a frequency response, the overall frequency response being a simple combination of the individual responses, it is appropriate and convenient to retain the block structure in the organisation of the data storage in the computer. This can be done by associating each block with a data table with, say, the format as shown in figure 5.



Data Table Format FIG.5

This format is based on the assumption that 30 sample points within the frequency range specified are adequate, if more sample points are required the size of the table will have to be increased. This format allows a maximum of two parameters to describe the block transfer function, the frequency range is held in common area, the Key indicates the type of block and details are given in appendix 3. Typical block types will be as follows:

Gain	K
1st order lag	$\frac{1}{1+TD}$
2nd order lag	$\frac{\omega_o^2}{D^2+2\omega_o D+\omega_o^2}$
1st order lead	$1 + TD$
Integration	$1/D$
Differentiation	D

Further block types can be added as the system expands, e.g., a table corresponding to a block whose transfer function is unknown, but whose frequency response has been determined experimentally may be included, the Key will indicate to the frequency response calculation program that it is to ignore this block.

To simplify the problem of inserting remaining blocks, it is preferable not to associate a given block with a fixed table 50, say, but to allocate tables dynamically and record block number-table number associations in a separate directory as shown in figure 6.

	KEY		
	MINIMUM FREQUENCY		
	MAXIMUM FREQUENCY		
	TECHNIQUE CODE		
	TOTAL RESPONSE TABLE NUMBER		
	UNUSED		
31	NUMBER OF BLOCKS		
	I	TABLE NUMBER	BLOCK NUMBER
		"	"
		"	"
		"	"
		"	"

I = 0 block in use
I = 1 block not in use

Frequency Response System Directory FIG.6

By using this additional directory, blocks can be added or removed from the system without moving data from table to table, all that is required is to change the table entries. Also, by the use of the additional flag I, blocks can be ignored, without being removed from the system. The frequency response system directory can also include details of the frequency range to be covered, the particular technique requested (Bode, Nyquist, Inverse Nyquist) and the table number where the total system response is held.

Having decided on a data structure it is now necessary to identify the program segments required for the system. These can be listed as follows:

(This is not a complete list, e.g., programs for handling feedback compensator blocks are not included).

(a) Input

(b) Frequency response - polar co-ordinates

(c) Inversion of frequency response

(d) Polar to cartesian co-ordinates

(e) Polar to log polar - to log cartesian

(f) Cumulative frequency response

(g) Scaling

(h) Stability calculator

(i) Display

(j) Teletype output

The input program will enable the various parameters to be entered. (This input program could be common to any system requiring transfer function information in block form, e.g., a simulation package). It will allow the user to opt for a step-by-step procedure whereby the computer provides a guide to the inputs required or if the user is experienced a command procedure whereby the user declares the inputs he wishes to make. This second option also enables parameters to be changed without the need to re-enter all the parameters. The program will have a number of other options as follows:

(i) 'NULL' set all tables to zero

(ii) 'READ' read problem description from paper tape

(iii) 'PRINT' print problem description on teletype

(iv) 'PUNCH' punch problem description on paper tape in a form suitable for re-entry via READ

(v) 'TYPE' allows problem description to be entered via teletype

The frequency response program will, using the system directory, determine the number of blocks in the system and calculate for each block in turn the frequency response and store it in the appropriate table. It will determine the form of calculation required from the data table key. Programs (c), (d) and (e) are self-explanatory, program (f) enables the total system frequency response to be built up block by block working backwards

from the final block. The display and output programs will be a set of programs allowing a number of options.

The above programs form the 'work' segments (see section 3.2), operating at a higher level will be the supervisor segments, BODE, NYQUIST, INVERSE NYQUIST, which will call the work segments in the appropriate manner and order for the above techniques.

It is interesting to note that, e.g., the frequency response segment could, in fact, run on the 1907, with the other segments being run on the CONPAC. This would normally be unnecessary, but could be done if additional accuracy was required.

6. Conclusions

There is a temptation, because of the size of the task, not to attempt to design and implement a comprehensive C.A.D. facility as outlined above, but to use simple individual design programs. This is in many circumstances a satisfactory solution to the problem of providing design facilities, particularly in the initial stages of development. However, in a small real time system the programs, if they are to remain separate, must remain simple, enhancement of each program can result in rapidly overloading the system and much duplication of effort. It is worthwhile, therefore, at the outset to examine the requirements of a comprehensive facility and to design the individual programs so that they can form the basis of a larger system.

The general requirements of a C.A.D. facility have been described and related to the hardware and software characteristics of the CONPAC 4020 computer. It has been shown how such a facility could be run under either the free time executive or real time executive of CONPAC, and it is concluded that it is preferable to operate under the real time executive. The method of implementation of a design facility has been considered, using the frequency response techniques as an illustration and a set of rules is given for writing design programs in order that they might be incorporated into a comprehensive facility at a future date.

Further work is now required to study in detail the requirements of the various design techniques and the implications of operating a joint CONPAC 4020 - 1907 system. This work can be divided into several sections all of which can be considered concurrently. The sections are as follows:-

(i) The development of design programs and in particular design algorithms, written as subroutines (the programs should be written in accordance with the rules outlined in section 5).

(ii) A study of the data structures suitable for the various design packages - single input single output systems, multiple input multiple output systems, simulation, 1907 communication, etc. A suitable structure

for frequency response work on single input single output systems has been outlined above.

(iii) The development of an executive program to control the packages and to provide facilities for user files.

(iv) The provision of standard input/output programs and a data manipulation package.

(v) The provision of graphics routines.

It should be stressed that the average user of a C.A.D. facility need not understand the detailed programming underlying the facility, but should have an awareness and understanding of the basic structure of the system. It is only through this awareness that the full benefits of the system can be obtained and its limitations exposed.

7. Acknowledgements

Discussions with Mr. J. B. Edwards and Mr. J. C. Wells on the implementation of the frequency response design system are gratefully acknowledged.

8. References and Bibliography

1. SIAS, F. R., and COLEMAN, T. G.: "Digital Simulation of Biological Systems using Conversational Languages", Simulation, 1971, 16, 3, pp. 102-111.

2. GRAY, J. O., and YUNG, T. F.: "Flexible Computer-aided Design Programs for Control Systems", Proc. I.E.E., 1971, 118, 3/4, pp. 618-621.

3. TAYLOR, F. E.: "A Guide to Multi-access Systems in Engineering", Computer Aided Design, 1971, 3, 2, pp. 29-34.

4. FRIED, B. D.: "On the User's Point of View", in "Interactive Systems for Experiemental Applied Mathematics", Academic Press, 1968, pp. 11-21.

5. STAMPER, R. K.: "Logical Structure of Files", in "Data Organisation for Maintenance and Access", B.C.S. Conference Proc., 1970, pp. 28-54

6. BUSCH, K. J., and LUDERER, G. W.: "The Slave Interactive System", in "Interactive Systems for Experimental Applied Mathematics", Academic Press, 1968, pp. 225-240.

7. WIESEN, R. A., YNTEMA, D. B., FORGIE, J. W., and STOVE, A. N.: "Coherent Programming in the Lincoln Reckoner", in "Interactive Systems for Experimental Applied Mathematics", Academic Press, 1968, pp. 167-177.

8. LUNTZ, R., MUNRO, N. and McLEOD, R. S.: "Computer-aided Design of Multivariable Control Systems", 4th U.K.A.C. Control Convention, I.E.E. Conference Publication No. 78, 1971, pp. 59-65.

9. EWING, D. K.: "Implementation of Data Structures for Engineering Design Problems", N.E.L. Report No. 429, 1969, pp. 20-27.

10. MICHIE, D. and WEIR, S.: "Application of Burstall's Control Routine to Conversational Statistics", Department of Machine Intelligence and Perception, University of Edinburgh, Memorandum: MIR-R-39, 1968.

BIBLIOGRAPHY

LEFKOVITZ, D.: "File Structures for On-Line Systems" Spartan, 1969.

KLERER, M. and REINFELDS J.: eds.: "Interactive Systems for Experimental Applied Mathematics", Academic Press, 1968.

GEC-AEI: "RTMOS System Manual"
"Free-time System Manual"
"Free-time System Programming Application Guide"

9. Appendix 1

Free Time Executive Program

This simple executive program which is shown listed below, accepts as input 6 character commands, each command has associated with it a free time program. The executive turns on the appropriate program, which then runs, on completion the program returns control to the executive. This executive is available on the COMPAC and is proving useful in enabling users to develop program segments.

```
C  FREE TIME EXECUTIVE FTXSBA

50     FORMAT(£SB FTX TEST £,/,£READER-I1£)        7
51     FORMAT(I1)                 7
52     FORMAT(£PRINTER-I1£)       7
53     FORMAT(£ENTER SBX£)        7
54     FORMAT(2A3)                7
55     FORMAT(£SEGMENT DOES NOT EXIST£)        7
56     FORMAT(£ERROR£,5X,I1)      7
100    PRINT (0) 50               7
       READ(1) 51,NR              7
       PRINT(0) 52                7
       READ(1) 51,NP              7
C BEGINNING MAINLINE              7
1      PRINT (NP) 53              7
       READ(NR) 54,NANA,NANB      7
        LDA NANA                  6
        STA *+8                   6
        LDA NANB                  6
        STA *+7                   6
       CALL FT LINKAGE(1,I)       7
       FORMAT(£DUMMYA£)           7
       IF(I) 3,4,5                7
3      PRINT(NP) 55               7
       GO TO 4  7
5      PRINT(NP) 56,I             7
4      CONTINUE                   7
       STOP     7
       END      7
```

10. Appendix 2

Real-time executive program

The real time system structure is more flexible than the free time system structure and hence the program designer has more freedom. For example, (a) more program priority slots are available, 45 compared to 4 for the free time systems, (b) programs in the real time system do not need to run in a fixed hierarchical pattern.

It is convenient, however, for a C.A.D. facility to use a limited number of priority levels, say 4 levels (30,31,32,33). The programs to run at these levels are held in an extended real time library (reference numbers 100-256). One function of the executive program is therefore to accept as input a program name, determine the appropriate program number and priority level, and run the program.

A real time program called by RT LINKAGE does not automatically return control to its calling program when it terminates, to return control it must, prior to termination, turn on that program, or more accurately, the program occupying that level. In the simplest structure each program will turn on the program occupying a given level, e.g., the program running at level 33 always turns on the program at level 32, etc. As an alternative a more

flexible structure can be achieved if each program turns on a program at a level which is indicated by the last entry in a push down stack (if the stack is empty the executive program is run). In this way, a completely arbitrary sequence of program segments can be run.

A simple real time executive program could be as follows:-

```
C  REAL TIME EXECUTIVE RTXSBA
       DIMENSION NA(10)
C  GET DEVICE NUMBERS FOR INPUT/OUTPUT
       CALL DEVICE
C  GET COMMAND
1      N=NRA
       NA(1)=2
       NPACK=1
       CALL STRING(NA,N,NPACK)
C  FIND PROGRAM NUMBER AND PRIORITY LEVEL
       CALL FINDNO(NA,NLEVEL,NNUMB,IE)
       IF(IE) 20,21,20
C  ANALYSE ERROR
20     CALL ERROR(IE)
C  TURN PROGRAM ON
21     CALL RT LINKAGE(NLEVEL,NNUMB,I)
       IF(I) 22,23,22
22     CALL ERROR(I)
       GO TO 1
50     STOP
       END
```

Each program segment will include the following coding if it is to be turned on following execution of another program segment.

```
       .......
       .......
C  RUN ANOTHER SEGMENT
        LDA PRIONO
        STA N
       CALL PSTACK(N)
       CALL RT LINKAGE(N,NN,IE)
       .......
       .......
```

where PSTACK places the current priority level in the stack.

Each program segment will also include the following coding:-

```
C  TERMINATION OF PROGRAM
       CALL GSTACK(N)
        LDA N
        STA PROG
        LDA TIME
        SPB TPNCO1
PROG   CON 0,30
       .......
TIME   CON G,0
       .......
```

where GSTACK obtains from the stack the number of the program which is to be turned on.

11. Appendix 3

File Structure

The first word of each file is a "key-word" which indicates the type and contents of the file. The format of the key-word is as follows:-

23	22 21 20 19 18	17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0	a	SUB-KEY SEE BELOW

a is the basic key, and has a value in the range 0-31. A typical use of the basic key is as follows:

a = 0 system files, e.g., references to user files
= 1 frequency domain program files
= 2 time domain program files
= 3 simulation program files
etc.

The arrangement of the sub-keys depends on the type of file, the sub-keys for the frequency domain program files are as follows:

23	22 21 20 19 18	17 16 15 14 13	12	11	10 9 8 7 6	5 4 3 2 1 0
0	0 0 0 0 0 1	b	c	d	e	f

where:

b = 0 system description file
= 1 block file containing block parameters and frequency response in polar co-ordinates
= 2 block file as 1, but also containing cumulative frequency response
= 3 total system response file, polar co-ordinates
= 4 as 3, but in cartesian co-ordinates
= 5 as 3, but in log-cartesian
etc.

c = 0 block in forward path
c = 1 block in feedback path

d = 0 normal frequency response
d = 1 inverse of frequency response

e specifies block type
= 0 dummy block
= 1 1st order lag
= 2 2nd order lag
= 3 1st order lead
= 4 2nd order lead
= 5 integration
= 6 differentiation
= 7 transport delay
= 8 gain
etc.

f is not used at present.