PROGRAMMING AND SIMULATION OF CONTROL ALGORITHMS USING COMPUTER GRAPHICS.

Thesis by

Lim Jit Wee

For the degree of

DOCTOR OF PHILOSOPHY

submitted to the

Department of Electrical and Electronic Engineering

THE UNIVERSITY OF ASTON IN BIRMINGHAM

October 1981

The University of Aston in Birmingham

'PROGRAMMING AND SIMULATION OF CONTROL ALGORITHMS

USING COMPUTER GRAPHICS.'

Ph.d. Thesis, Lim Jit Wee, October 1981

Summary

A system for the programming and evaluation of process control algorithms has been developed. A graphical block diagram language was adopted to provide an easy means of programming via a graphic terminal. A number of pre-defined blocks are provided and programming is achieved by connecting the required blocks together. The graphic blocks correspond to software modules performing the required control functions.

The pictorial program is then compiled to give a machine-independent program structure table. This can then be linked with the block routine code to provide the target processor's control program to realise a required control algorithm.

A similar graphic approach was adopted for simulation of the controlled processes, so that the performance of the control algorithm may be evaluated. The use of standardised software modules and computer graphics simplifies programming and the maintenance of system documentation.

The interaction and communication between the two sub-systems (control scheme and process model) supports the testing and evaluation of control algorithm. The data structures for effective graphics and efficient execution of the blocks are specified. The various functions necessary for the compilation of the graphical program are investigated and analysed.

KEYWORDS : control algorithm programming, simulation,

graphical language, compilation,

data structure

ACKNOWLEDGEMENTS

The Author wishes to express his appreciation and gratitude to his supervisor, Dr. G. K. Steel, for his invaluable assistance and guidance throughout the period spent on this research and to Professor H. A. Barker for his initial supervision.

Thanks are also due to the staff of the department, colleagues and friends for their encouragement and advice. Special thanks to Dr. C. Y. Linn for his assistance and co-operation.

Finally, the Author wishes to thank his parents and the rest of the family for their patience, support and kind understanding throughout the project.

LIST OF CONTENTS

Summary	i
Acknowledgements	ii
List of contents	iii
List of illustrations	viii
List of tables	xi

CHAPTER

1. General Introduction and Research Objectives	1
2. Process Control	4
2.1 Introduction	4
2.2 Programming languages	5
2.3 Choice of programming approach	8
2.4 Levels of process control	9
2.5 Generation of control program code	10
2.6 Execution on dedicated control processor	14
2.7 Block types in process control	16
2.7.1 Implementation of interface blocks	20
2.7.2 Minimal basic set of blocks	21
2.8 Implementation of blocks	22
2.8.1 Integrator block	22
2.8.2 Delay block	23
2.8.3 PID block	24
2.8.3.1 Introduction	24
2.8.3.2 Variations of algorithms	26
2.8.3.3 Further considerations	28

3. Graphics Systems 33

3.1 Graphics display 33

3.2 Graphical input mechanism 35

3.3 Graphics software 36

3.4 Graphics and data structure 37

3.5 Graphic data structure 38

3.6 Graphic picture structure and programming methodology 39

4. Data Structure 42

4.1 Graphic data structure 42

4.1.1 Introduction 42

4.1.2 Graphic block (GB) record 43

4.1.3 Graphic information (GI) record 46

4.1.4 Macro expansion (MX) record 48

4.1.5 Non graphic data (NGD) record 49

4.1.6 Graphic text (GTXT) record 50

4.1.7 Input list (IL) record 52

4.1.8 Relation between graphic records 53

4.2 Graphic menu 54

4.3 Run time data structure 56

4.3.1 Introduction 56

4.3.2 Run time block records 57

4.3.3 Upwards link (ULN) record 58

4.3.4 Run time type records 59

4.3.4.1 RT simple type (RST) record 59

4.3.4.2 RT composite type (RCT) record 60

4.3.5 Run time text (RTXT) record 61

4.3.6 Run time value table (RVT) 62

5. User Interface and Graphic Editor 78

5.1 User interface design 78

5.1.1 Introduction 78

5.1.2 Command languages 79

5.1.3 Feedback considerations 81

5.1.4 Information display 82

5.2 Graphic editor 84

5.2.1 Introduction 84

5.2.2 Graphics hardware description 84

5.2.3 Pick function 86

5.2.4 Editing of the picture 87

5.2.5 Connections to and from junction 90

5.2.6 Text editing 91

6. Graphic Compiler 93

6.1 Introduction 93

6.2 Transformation of graphic type to run time type 94

6.3 Expansion of macro block and subpicture 95

6.4 Error checking by graphic compiler 96

6.4.1 Missing connections 96

6.4.2 Illegal data type connection 96

6.4.3 Algebraic loops of non-retrospective blocks 98

6.5 Sequencing the blocks for execution 99

6.6 Allocation of data tables for blocks 100

6.7 Initialization of the run time data table 103

6.8 Listings and messages 103

6.8.1 Examples of listings and warnings 104

7. Compilation Activities (1) Picture Validation 106

7.1 Run time treatment of composite blocks 106

7.2 Closed loops of blocks 109

7.3 Algebraic Loop detection 111

7.3.1 Introduction 111

7.3.2 Implementation of loop detection scheme 114

8. Compilation Activities (2) Sequencing 117

8.1 Processing order amongst control scheme blocks 117

8.2 Sequencing functional block 119

8.2.1 Introduction 119

8.2.2 Sequencing method implementation 120

8.2.3 Data structure used in sequencing 122

8.2.4 General comments on sequencing 123

9. Compilation Activities (3) Data Manipulation 125

9.1 Allocation of storage for run time block 125

9.1.1 Introduction 125

9.1.2 Data structure for retrospective block 126

9.1.3 Data structure for non-retrospective block 128

9.2 Initialization of functional blocks 129

9.2.1 Introduction 129

9.2.2 Initialization criteria 130

9.2.3 Input-output initialization 131

9.2.4 Steady-state initialization 132

9.2.5 Further considerations 134

9.2.5.1 Steady-state initialization 134

9.2.5.2 External terminal initialization 136

10. Simulation 137

10.1 Review of digital simulation languages 137

10.2 State variable representation 139

10.3 Integrator block in simulation 140

10.3.1 Introduction 140

10.3.2 Features of single step and multi-step approaches 143

10.3.3 Error estimation 145

10.3.4 Control of integration step size 147

10.4 Relations between integrator and other blocks 148

10.5 Distinction betwwen H and Tc 149

10.6 Block types in simulation 150

10.6.1 Minimal set of basic blocks required 151

10.6.2 Composite block implementation 152

10.7 Delay block 152

10.8 Derivative block 153

10.9 Simulation and the graphic compiler 155

10.9.1 Sequencing of blocks 156

10.9.2 Storage allocation for blocks 157

11. Testing of control algorithms 159

11.1 Introduction 159

11.2 Interaction supervisor 160

12. Conclusion and results 164

12.1 Programming implementation of system 164

12.2 Conclusion 165

Appendices 167

List of references 172

LIST OF ILLUSTRATIONS

2.1 Generation of program code 12

2.2 Scheme of downloading to dedicated controller 14

2.3 Memory map of dedicated control processor 15

2.4 Examples of DDC functional blocks 16

2.5 Delay block implementation 24

2.6 Basic PID - block diagram 25

2.7 Interactive PID - block diagram 27

2.8 Incremental PID - block diagram 28

2.9 Set point derivative elimination - PID 31

3.1 Simplified model of interactive process 33

3.2 Heirachical structure of picture 41

4.1 Graphic block (GB) record 63

4.2 Graphic information (GI) record 64

4.3 Macro expansion (MX) record 65

4.4 Non-graphic data (NGD) record 65

4.5 Attributes of the flags in NGD record 66

4.6 Graphic text (GTXT) record 67

4.7 Graphic type specific text records 67

4.8 Graphic block specific text records 68

4.9 Input list (IL) record 69

4.10 Relation between graphic records for simple block 70

4.11 Relation between graphic records for composite block 71

4.12 Graphic menu data structure 72

4.13 Run time block records 73

4.14 Upwards link (ULN) record 74

4.15 Example of run time composite block record of 3 simple blocks 74

4.16 Run time simple type (RSTYPE) record 75
4.17 Run time composite type (RCTYPE) record 76
4.18 Run time text (RTXT) record 77
5.1 Equipment used in project 85
5.2 Simplified model of graphic editor 87
6.1 Simplified model of graphic compiler 94
6.2 Example of an algebraic loop 98
6.3 Relation between run time data file and block record 102
7.1A Subroutine and effect on run time data 108
7.1B Macro and effect on run time data 108
7.1 Run time treatment of composite block 108
7.2 Square root function diagram, algebraic loop 109
7.3 Non-linear filter diagram, non-algebraic loop 111
7.4 Loop detection scheme 113
8.1 Processing order amongst control scheme blocks 118
8.2 Groups of interconnected nodes 120
8.3 Block sequencing example 122
8.4 Data structure used in sequencing 123
8.5 Program modules and data flow 124
9.1 Run time data file structure 127
9.2 Structure of LEAD/LAG function block 129
9.3 Structure of first order lag block 131
9.4 Initialization block diagram 134
10.1 Runge-Kutta 4th order integration rule 142
10.2 Adam-Moulton 4th order predictor-corrector 142
10.3 Runge-Kutta-Merson 4th order integration rule 146
10.4 Distinction between H and Tc 149
10.5 Composite block implementation - simulation 152

10.6 Derivative block and difference equation approach 153

10.7 Derivative function from integration 154

10.8 Execution order amongst simulation blocks 156

10.9 Data allocation for integrator 158

11.1 Interaction between control scheme and model 162

11.2 Interfacing to actual process 163

11.3 Data structure for interaction communication 163

B.1 Graphical details of integrator block 169

B.2 Graphic data records for integrator block 170

B.3 CORAL 66 procedure for integrator block 171

LIST OF TABLES

2.1 Transformation of integral action to discrete form 22

5.1 Graphic editor command characters 88

CHAPTER 1

GENERAL INTRODUCTION

During the last two decade the advent and proliferation of the minicomputers and microprocessors has reduced the cost of digital processing power. With lower cost and higher reliability of computing elements, digital control is becoming widely adopted. This is mainly possible due to the advancement in technology and mass manufacturing of programmable hardware. It is this programmable capability that makes the computing system very versatile, thus very responsive to the changes in requirements.

In this project computing power is exploited to create a system which allows programming to be carried out by graphics or drawings. This means of programming will eliminate the major hindrance to the use of computers - the need to acquire detailed knowledge of the computer hardware and programming languages. This system permits the user to describe the problem to the computer system by drawings, a very natural and effective man-machine means of communication.

The main objectives of this research are in the areas of

(1) designing and implementation of a computer graphics package. The package provides an efficient means of entering graphical information into the data base in

the form of a graphical programming language (GPL). The data structure required for efficient handling of graphic information is identified.

(2) the compilation of the graphical programs to executable programs representing process control algorithms. The graphic data are combined with pre-fabricated software modules. Use of such standardised modules results in more reliable software.

(3) to test the working of the control algorithms and allow the interactive modification of control parameters. Reliable control software is obtained by testing the interaction with a simulated model of the process. The graphical programming approach is extended to include simulation algorithms. The interaction ensures the satisfactory performance of the control algorithms before loading them to dedicated control processor or controller.

The thesis is divided into the following chapters. Chapter 2 deals with process control aspects - general requirements and the implementation of some functional blocks. Chapter 3 is concerned with the general graphics system - display and input mechanisms. In chapter 4, the details of the data structures used for graphics purposes and the execution phase are discussed. A tailored-purpose data structure for graphics (GDS) is chosen to ensure efficient graphics operation and memory storage utilisation. The execution phase data structure (RDS) is concerned more

with the efficiency of execution. Chapter 5 covers the man-machine interface of the GPL system and gives a general description of the graphic editor and its capabilities. In chapter 6, 7, 8, 9 the graphic compiler of GPL and its activities are analysed and discussed. Chapter 10 deals with simulation - general requirements and the distinction between simulation to control algorithm functions. In chapter 11, the testing and evaluation of the control programs are considered. Chapter 12 concludes the thesis with a review of results.

CHAPTER 2

PROCESS CONTROL

2.1 INTRODUCTION

The application of digital computers to process control is now well established. Digital process control has been in use in industries for a long period of time, the first control computer being installed in 1958 [EDWARDS 1972]. However due to the high cost and relatively unreliability of early computers, the spread of digital process control was much smaller than expected.

During the last two decades, with the technological advancement and extensive manufacture of computers and other LSI (large scale integration) components, the cost of digital processing hardware has been decreasing at a very rapid rate [MUSSTOPF 1979A], [SCIAM 1977]. Low cost, and high component quality, in turn lead to more and wider applications. The reliability of digital hardware has been further enhanced by the usage of distributed computing systems, with local data processing and remote communication, and fault-tolerance computer architecture using redundancy [RZEHAK 1978], [DEPLEDGE 1981].

The field and depth of computer applications in control are well covered by [HEALEY 1975] and [DUYFJES 1977]. One major obstacle to even wider application of digital

control is the programming aspect. Process control programming is a comparatively specialist activity, since programming skill must be supplemented by thorough awareness of control engineering and the process itself. The lack and cost of personnel with the above qualifications severely hinder the spread of computer process control.

In efforts to overcome this problem, the philosophy of control software design has been undertaken in many approaches (discussed in the following section). The general trend is to allow the user himself to describe the process and the control requirements to the control computer, bypassing the use of specialist programmers. The general development and trends of process control software are well highlighted by the following publications [PIKE 1970], [PIKE 1972], [SCHOEFFLER 1972], [IECI 1968] and [IECI 1969].

2.2 PROGRAMMING LANGUAGES

The classifications of programming languages will be discussed below with reference to process control requirements. Programming languages can be generally divided as follows [MUSSTOPF 1979B], [STEUSLOFF 1979] :

(A) Assembly/machine language. This is the lowest level of programming, requiring detailed knowledge of the specific machine used. The advantages are efficiency in execution and storage space utilisation. The main

disadvantages are that (1) errors can be easily made and are difficult to "debug" and (2) the software is not "machine portable". So assembly language is normally used to overcome very stringent restrictions on memory space and processing requirements.

(B) High Level sentence-type Language (HLL). High level languages improve on assembly languages in terms of programmer productivity, documentation, maintainability and portability. HLL programs are easier to write and debug. The portability allows the transfer of programs to other machines. One sentence in HLL is usually the equivalent of many "sentences" of assembly language. However HLLs do not normally produce very efficient machine language programs. HLL can be classified as :

(1) General Purpose. The most commonly used are FORTRAN, COBOL, BASIC and PASCAL. Extensions of FORTRAN for control applications are commonly carried out e.g. [IPW/EWICS 1981]. Certain HLLs are designed for "real-time applications" such as RTL/2 [BARNES 1975], PEARL [FREVERT 1975] and CORAL 66 [HALLIWELL 1977], [WOODWARD 1974]. Being general purpose in nature such languages contain features that may not be necessary in process control, leading to inefficiency. Control software programming is still a difficult task requiring time and skill.

(2) Process-oriented (problem-oriented). Here, in order to ease the programming effort, familiar engineering terms are introduced into the language. A problem-oriented language is closely related to the application field. Examples of such languages are AUTRAN [GASPAR 1968], BATCH [PIKE 1970], ACCOL [BRISTOL 1975] and PROSEL [NOBLE 1977].

(C) Fill-in-the-blanks (FIB) language. This approach eliminates the need for the knowledge of programming by the user in the normal sense. All "programming" is done by filling in the questions on pre-prepared "forms". A large number of FIB languages exist such as BICEPS [PIKE 1970], APEX [KELLY 1967] and PROSPRO [BATES 1968]. Most FIB languages are for special applications in specific fields. Such languages are usually written in a high level language.

(D) Block-oriented language (BOL). This is similar to the FIB approach except now most of the pre-defined software modules are defined in terms of "functional blocks". Each block performs a certain function such as the three-term or lead/lag control. Use of such pre-fabricated modules tends to result in more reliable control software programs. Programming here involves the interconnections of whatever blocks are required (obtaining a "control-diagram" drawing). There are two methods of communicating the structure of the drawings to the computer system, namely

(1) by labelling the input and output terminals of each block and entering the structure in alphanumeric form. Examples of such BOLs are MICRODARE [KORN 1979] and an approach by Lee [LEE 1967]. Translation from the block structure to the intermediate code may be tedious and error-prone.

(2) by entering the actual drawing using a graphic terminal. The computer system may then work out the interconnections.

The approaches in (C), (D) and (2) of (B) can be regarded as very high level languages (VHLLs). Such VHLLs have a major disadvantage in the requirement of a long implementation time but do offer the following advantages : (i) they are relatively easy to learn and to use, (ii) it is possible to adopt standards for usage, (iii) they can be modified quite easily to suit a particular application and (iv) they offer good application-oriented, readable documentations.

2.3 CHOICE OF PROGRAMMING APPROACH

One of the main features of the usage of computers is to provide an easy programming facility and a friendly user interface. To quote Wilke [WILKE 1979], "emphasis in process control packages .... has been on the facilities for operators and users, rather than on any sophistication of control". The programming language in this implementation is

a graphically based block oriented language. Graphics is a natural medium for man-machine interaction [FOLEY 1974]. The GPL (graphical programming language) allows the programming to be carried out by entering the actual "drawing" of the various blocks to the computer system using a graphic terminal. The two main advantages of using computer graphics in GPL are the ease in use and the only documentation required is the drawing. The GPL package will be further discussed in section (2.5).

2.4 LEVELS OF PROCESS CONTROL

Process control software can be generally divided into four levels of function [PIKE 1970], [SCHOEFFLER 1970], [TOCZYLOWSKI 1978]. From the lowest level upwards they are :

(1) Direct Digital Control (DDC). This level communicates with the process variables directly. Data acquisition and direct control are carried out at regular intervals.

(2) Optimization Control. This level involves supervisory functions applied to the DDC system. Optimization is based on performance criteria relating to the overall system performace.

(3) Adaptive Control. Here the process model used for the optimization control can be checked by on-line measurements and modified if necessary. Security checks

can be carried out to detect any plant malfunctions.

(4) Mangement Information. Tasks at this level supply information to management and permit overall control of the process behaviour.

Emphasis in this project is mainly with the DDC level. Process time constants and the execution speed of the computer are important considerations for DDC. In processes where the process time constants are in the order of milliseconds, DDC is not generally feasible. For many slower processes with time constants in the order of seconds or minutes DDC is readily applicable. The general applications of DDC are well surveyed by Auslander & co-workers [AUSLANDER 1978] and Varga & co-workers [VARGA 1979].

2.5 GENERATION OF CONTROL PROGRAM CODE FROM GPL

The general sequence in obtaining the control program code using the GPL is shown in figure (2.1). The intermediate steps are :

(1) the synthesis of the GPL "program" from the user block diagram concept, using the graphic editor.

(2) the compilation of the GPL program by the graphic compiler to give the program structure table (PST). This table contains all the information as to the block used and interconnections. The PST is numerical in nature, comprising various run time data structure

records (section 4.3).

(3) the program generation. The program generator "links" the PST with the appropriate run time algorithm code routine to give the run time program code. This machine-dependent program code is ready for execution on the target processor. The execution of the program code by the processor will be denoted throughout the thesis by the term "run time phase".

The GPL program and the PST obtained in (1) and (2) are machine-independent since they are only involved in the transformation of the pictorial block diagram to a numerical form suitable for the program generation section.

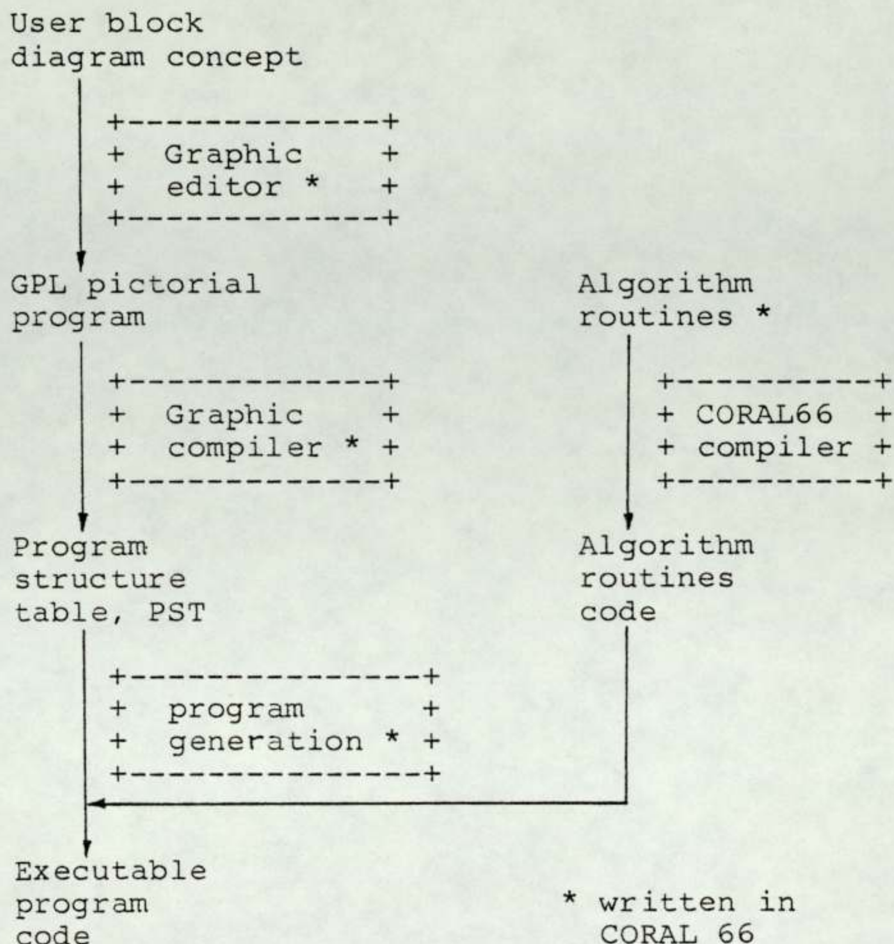


FIGURE 2.1 GENERATION OF PROGRAM CODE

The structure of the package system can be divided into four operational stages : (1) graphic editor, (2) graphic compiler, (3) program generation section and (4) run time algorithm routines. The first three stages are independent of each other and operate on input data to give the appropriate output for the next stage. If any error or ambiguity is encountered then the relevant message is displayed and the next stage is not activated.

To allow the package to be portable, every stage is written in a high level language, CORAL 66 [HALLIWELL 1977], [WOODWARD 1974]. Portability is further enhanced by separating the machine-dependent code from the machine-independent portions. To transfer the package to a different computer system (with a suitable CORAL 66 compiler), only the machine-dependent code requires modification. Most of the machine-dependent code segments are involved with the input and output activities.

The algorithm routine determines the functional characteristics of each block, relating the output to the inputs. Each routine is functionally independent, needing only the information as to the parameter list (the block's variables). The routines are written in CORAL 66 which is further compiled to give the machine-dependent code. If it is deemed necessary to modify the function of a block, then only that algorithm routine need to be changed accordingly.

The program generation is a relatively easy task compared with the graphic compilation. This is basically a "linking" function, linking the required functional type in the PST with the appropriate address of the routine code. This function is very similar to that required for any block-oriented language approach.

2.6 EXECUTION ON DEDICATED CONTROL PROCESSOR

This section considers one of the feasible scheme of execution of the run time control program code obtained from the GPL system on a control dedicated processor. The scheme involves downloading to the processor from the host computer as shown in the figure (2.2).

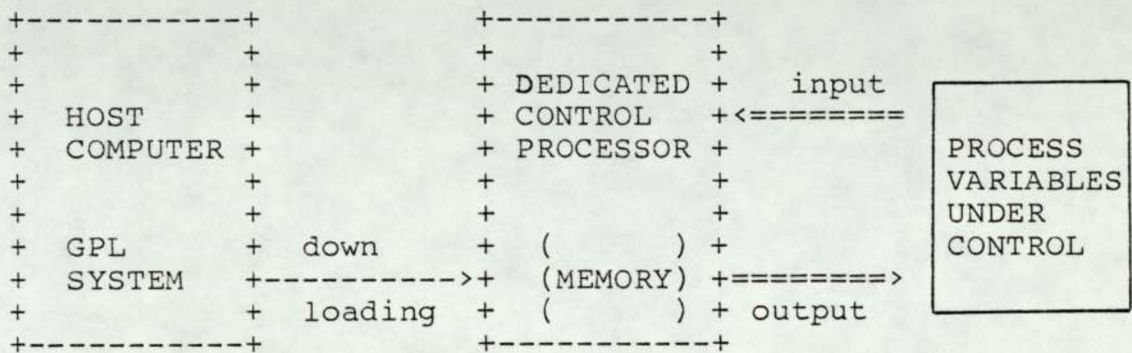


FIGURE 2.2 SCHEME OF DOWNLOADING TO DEDICATED CONTROLLER

There are many different methods of downloading, via

(1) "blowing" onto EPROM (programmable memory). The EPROM can then be plugged into the control processor for execution.

(2) transferring on to a magnetic tape, and getting the control processor to pick up the relevant information from the tape.

(3) a serial link from the host computer to the control processor. This serial link is used to pass all the required information.

The memory map of a typical control processor using a serial link is indicated in figur_ (2.3).

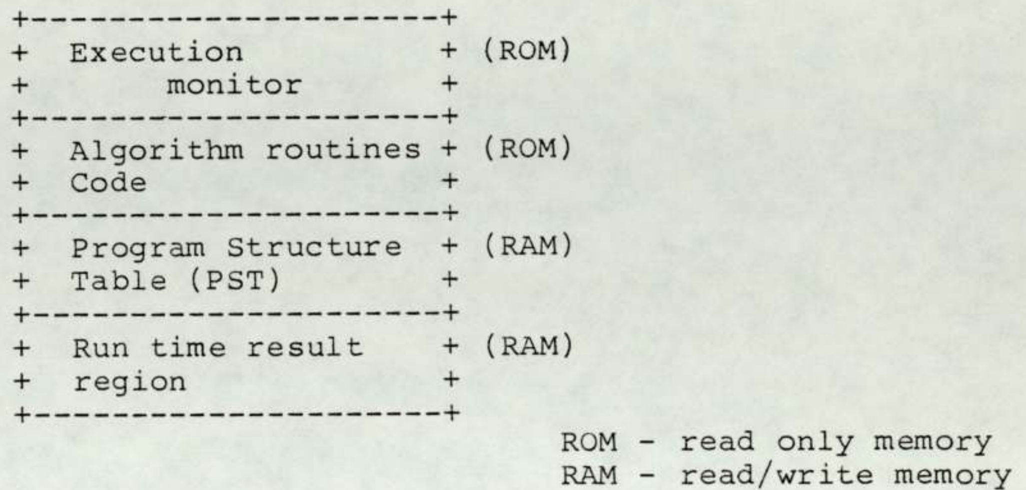


FIGURE 2.3 MEMORY MAP OF DEDICATED CONTROL PROCESSOR

The program structure table (PST) is generated by the host computer and defines the particular control algorithm employed. The monitor picks up the relevant data from each block in the PST, finds the correct functional algorithm code and executes the code. The result can be dumped onto the result region before going to the next block in PST.

The dedicated control processor described above is a simple but crude system. The processor system can be enhanced to give a higher performance such as to provide a display and modification interaction for the operator. This can be achieved by having a more sophisticated execution monitor. The execution of the control program on dedicated processor is an area not covered by the research objectives and will not be considered further in this thesis.

2.7 CLASSIFICATION OF FUNCTIONAL BLOCKS

The basic set of functional blocks simulate functions familiar to process control engineers. Some of the more frequently used blocks are indicated in the figure (2.4). The GPL permits the provision of a new block in 2 ways,

(1) by the specification of a set of numerical coordinates (for drawing the block symbol) and an algorithm procedure to define the relationship of output to input. Appendix B gives an example of this process.

(2) by taking blocks from the basic set to create a new "composite block".

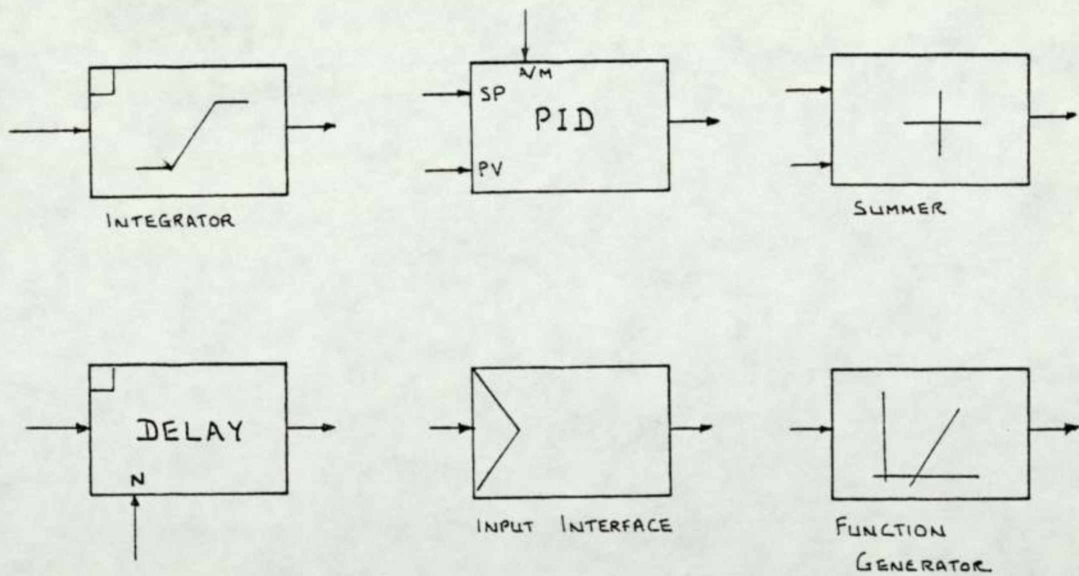


FIGURE 2.4 EXAMPLES OF DDC FUNCTIONAL BLOCKS

All the fundamental block types in the process control system can be generally classified as one of the following :

(A) NON-RETROSPECTIVE BLOCKS

These blocks have outputs as instantaneous functions of the inputs. They fall into two categories :

(1) block whose current output depends solely on the current input value. Examples include the adder and multiplier.

(2) block whose current output depends on the current input and the past value history of the input and/or output. Some examples are the leadlag function and the PID function.

Implementation of the type (2) differs from type (1) in that it requires a past value queue to be maintained.

(B) RETROSPECTIVE BLOCKS

Retrospective blocks are those whose outputs are computed based only on the past history of their inputs and/or outputs. On no account are the current inputs involved. These blocks can be processed during the run time phase in any order. Examples include the integrator, first order lag and the second order lag functions.

(C) MACRO BLOCK AND SUBPICTURE

Macro block and subpicture are "composite" blocks, which are constructed by various other blocks, retrospective or non-retrospective, as separate entities. The composite block can be regarded as the graphical equivalent of a subroutine. Once such a block is defined the user can just use it as a "black box", ignoring the internal working of the block. The composite block approach promotes the modern programming methodologies [YOURDON 1975], allowing the user to view the problem as a hierachical level of black boxes. The advantages of the composite block are

(a) It leads to more comprehensible diagrams.

(b) Frequently used configurations can be defined once and called up whenever required.

The concept of a subpicture differs from that of a macro block. A large block diagram may be partitioned into subpictures for convenience or where the display capacity requires this. A subpicture is a composite block used only once in the control scheme. Its internal structure can be freely modified at any time. A macro block is designed to be used as many times as necessary within a control scheme. A change in this block will affect all implementations and this can lead to errors in appreciating the full effect of changes. For this reason, macro blocks are regarded as fixed entities.

(D) INPUT INTERFACE BLOCKS

The input interface block is used as an interface between the process (or the model of the process) and the actual control scheme. It enables measurements from the process to be passed over to the control scheme. Generally there are two classes of input interface, the analog and digital. Device handlers will be handling all the means of measurement of the process, including the sampling interval. The process analog variable may be scaled and linearised and fed through a A/D converter to give a finite range, normally in the 12 bit representation. The actual physical address of the instrumentation will be handled by the device handler and the output sent to memory locations. These memory locations may be updated by internal transfers or DMA (Direct Memory Access) may be used. DMA allows blocks of memory locations to be updated by means external to the processing unit.

It is the function of the analog interface to "format" the integer output value of the device handler into floating point representation and to carry information about the scaling factors. This means as far as the other blocks are concerned that they are only interested in the value at the appropriate memory location and not the physical means of implementation of the instrumentation. No formatting is required in the digital interface, just storing the binary input in the appropriate memory location.

(E) OUTPUT INTERFACE BLOCK

Output interface blocks act as the interface between the control algorithm and the process (or model of the process). Similar to the input interface, there are two classes, the analog interface and digital interface. The main function of the analog interface is to format the output values from floating point representation down to the finite integer range, normally in the form of 8 bits representation. To do so, the output interfaces require the scaling factor and the upper and lower limits of the variable. Output device drivers will be handling all the physical addressing and the activation of the actuators.

2.7.1 IMPLEMENTATION OF OUTPUT AND INPUT INTERFACES

As far as this project is concerned, the interfaces are between the control algorithm and the simulation model of the process. These interfaces can be considered as message passing modules. The input interface will be given the following data :

(1) the upper limit of the incoming variable

(2) the lower limit of the variable

(3) the actual value of the variable as a percentage of the two limits, as an integer value.

Based on these values, the input interface module will format the incoming percentage integer value into the

floating point representation used in the rest of the blocks. The output interface with the following data :

(1) the floating representation of the variable

(2) the upper limit of the output variable

(3) the lower limit of the output variable

will compute the output (integer value) to be transferred to a D/A converter.

All the integer values will be over the range of a 8-bit representation (0-255), interfaces are usually limited in range and precision.

2.7.2 MINIMAL BASIC SET OF BLOCKS

The minimal basic set of the functional blocks essential for DDC is considered to be as given below :

(a) retrospective blocks - integrator, first order lag, second order lag and the delay function.

(b) non-retrospective blocks

type (1) - summer, multiplier, function generator and junction block.

type (2) - lead/lag function and PID controller

(c) interfaces - input and output blocks.

2.8 IMPLEMENTATION OF FUNCTIONAL BLOCKS

The function of each block is determined by its algorithm procedure. The implementation of the conventional blocks such as the summer, multiplier and function generator are straightforward and obvious. A few of the functional blocks can be implemented by various approaches and some of these are considered in the following sections.

2.8.1 INTEGRATOR BLOCK

There are many methods of transforming the analogue integral action into the equivalent discrete form [D'HULSTER 1979], [ROSKO 1972], see table (2.1).

method	transfer function	forward extrapolation eqn.
difference	$\frac{T}{1-z^{-1}}$	$y_{n+1} = y_n + Tx_{n+1}$
z-transform	$\frac{Tz^{-1}}{1-z^{-1}}$	$y_{n+1} = y_n + Tx_n$
Tustin	$\frac{T}{2}\frac{1+z^{-1}}{1-z^{-1}}$	$y_{n+1} = y_n + \frac{T}{2}[x_{n+1} + x_n]$

T = sampling interval
z = backward shift operator
x = input, y = output

TABLE 2.1 TRANSFORMATION OF INTEGRAL ACTION TO DISCRETE FORM

From the above table, the z-transform approach gives an extrapolation equation in which the current output can be calculated without the knowledge of the current input (i.e. retrospective in nature). The z-transform approach is adopted for the implementation to obtain retrospective integrators allowing them to be executed in any abritrary order. This choice implies that the z-transform approach is also used for any functional block involving the integral action such as the first order lag and lead/lag function (see Appendix A).

2.8.2 DELAY BLOCK

There are two approaches to the approximation of the delay block [KEY 1965]. One approach is to satisfy the mathematical transfer function having a constant gain and a phase shift proportional to the frequency (e.g. Pade approximation or Stubbs and Single's approximation [JACKSON 1960]). The other approach is to store the input and to reproduce this after the desired interval of time. The shift register is adopted to achieve digital computation when the required delay is an integer multiple of the sample interval.

The delay block is implemented as follows,

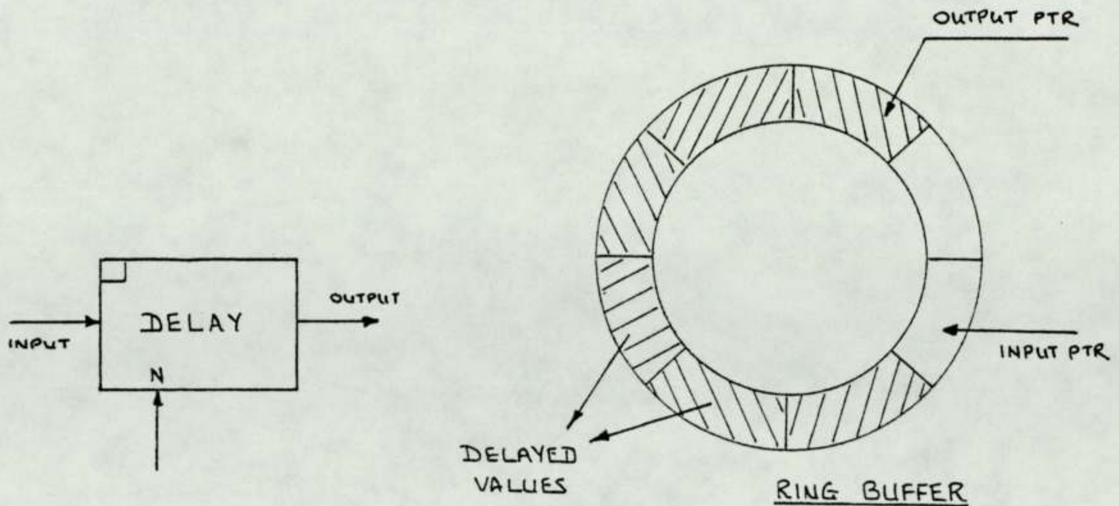


FIGURE 2.5 DELAY BLOCK IMPLEMENTATION

The input parameter N is the number of the time delay intervals to be specified by the operator. A variable-sized delay block causes difficulties during the storage allocation. So the delay block is implemented with a fixed-size M-stage ring buffer. This avoids having to shift all the actual values along since only the two pointers need updating. Precautions must be taken to ensure that N does not exceed M.

2.8.3 PID CONTROLLER

2.8.3.1 INTRODUCTION

Most process loops are controlled by the very flexible PID algorithm or one of its variants. A PID controller can be easily "tuned" (i.e. its parameters varied) to give the required performance of the manipulated variable.

As implied by the name PID controller (Proportional, Integral and Derivative controller), also widely known as the "three term" controller, the basic algorithm is as follows :

$$v = K\left[e + \frac{1}{T_i}\int e\,dt + T_d\frac{de}{dt}\right] + v_m \qquad (2.1)$$

where v = output of the PID algorithm

e = input signal to the PID

K = proportional gain

Ti = integral time constant

Td = derivative time constant

vm = manual reset output value.

Figure (2.6) shows the basic PID algorithm in diagrammatic form.

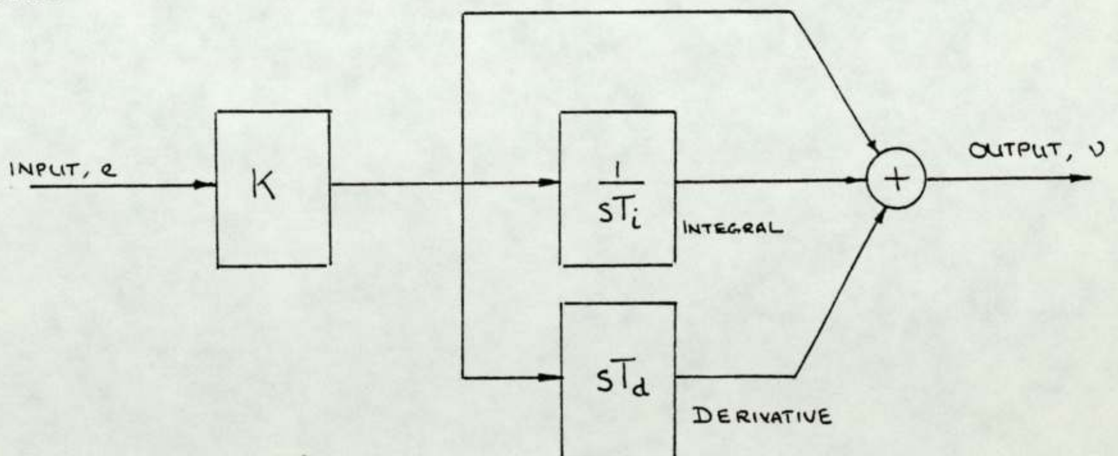


FIGURE 2.6 BASIC PID - BLOCK DIAGRAM

In the implementation of this, consideration must be given to other factors, such as the operator interface,

filtering of signals, automatic/manual transfer, bumpless parameter changes, reset windup and non-linear output requirement [ASTROM 1980]. Detail considerations will be given in later sections.

2.8.3.2 VARIATIONS OF THE PID ALGORITHM (1)

The basic PID algorithm can be modified to give several variants to provide for different operating requirements. The basic algorithm (equation (2.1) and figure (2.6)) is also known as the ideal or non-interactive PID, since all the three terms can be set independently. By approximating the integral and derivative terms, the following equation is obtained :

$$v = K\left[e_n + \frac{1}{T_i}\sum_j e_j T_s + T_d\,\frac{e_n - e_{n-1}}{T_s}\right] + v_m \qquad (2.2)$$

where Ts is the sampling interval.

With further manipulation,

change in v,

$$v(n) - v(n-1) = K(e_n - e_{n-1}) + \frac{KT_s}{T_i}e_n + \frac{KT_d}{T_s}(e_n - 2e_{n-1} + e_{n-2}) \quad (2.3)$$

(Note that the manual reset output vm is now not required.) This is known as an incremental algorithm since only the change in the output, v, is calculated.

2.8.3.2 VARIATIONS OF THE PID ALGORITHM (2)

In practice, most analog contollers are better represented by [BIBBERO 1977] :

$$\frac{v}{e} = \left(1 + \frac{1}{sT_1}\right)(1 + sT_2)\,K_1 \tag{2.4}$$

where T1 = equivalent of the integral time constant

T2 = equivalent of the derivative time constant

K1 = gain

This is known as the <u>real</u> or <u>interactive</u> PID algorithm (figure (2.7)).

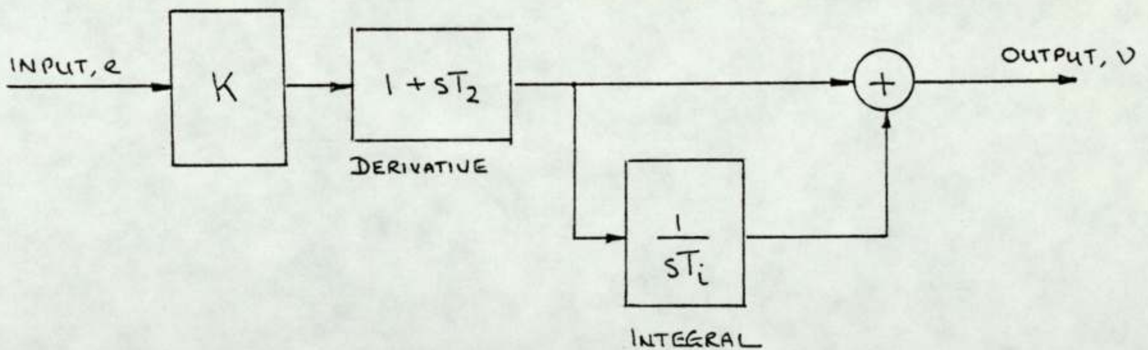


FIGURE 2.7 INTERACTIVE PID - BLOCK DIAGRAM

Manipulation of equation (2.4) gives

$$\frac{v}{e} = (1 + sT_2)\left(s + \frac{1}{T_1}\right)\frac{K_1}{s} \tag{2.5}$$

The front portion of the expression represents an incremental algorithm and the latter an integrator function, figure (2.8).

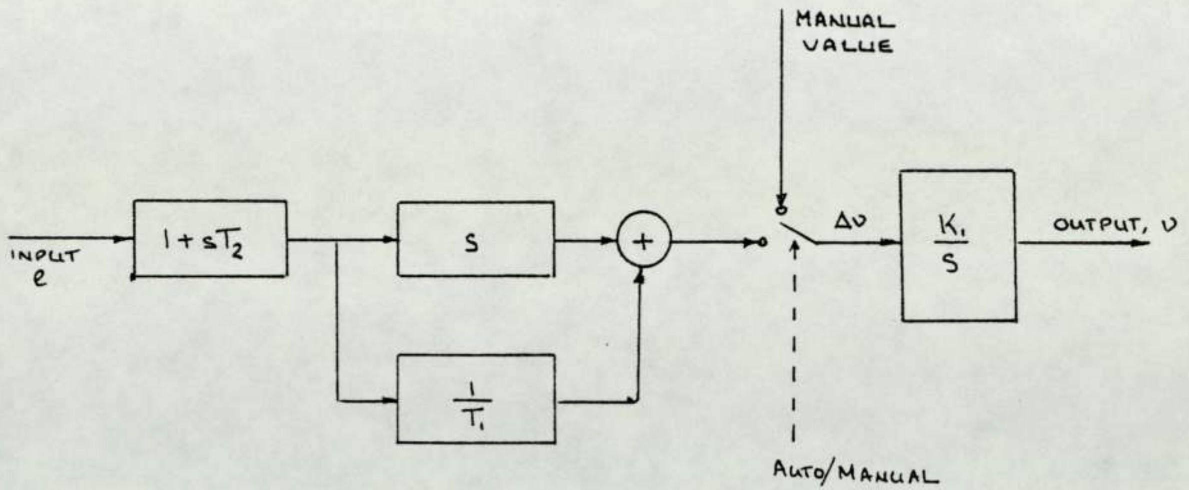


FIGURE 2.8 INCREMENTAL PID with internal integrator

Working on just the incremental portion gives :

change in v,

$$v(n) - v(n-1) = \frac{K_1(T_1+T_s)(T_2+T_s)}{T_1 T_s} e_n - \frac{K_1}{T_1 T_s}(T_1 T_s + T_2 T_s + 2T_1 T_2) e_{n-1} + \frac{K_1 T_2}{T_s} e_{n-2} \qquad (2.6)$$

Equation (2.6) is selected to be implemented as the PID incremental algorithm. Further information on the algorithm of PID can be found in [CADZOW 1970] and [SMITH 1972].

2.8.3.3 FURTHER CONSIDERATIONS ON PID

The final choice of the algorithm for the PID is partly dependent on the following factors [BRISTOL 1977] :

(1) FILTERING OF SIGNALS

In most practical cases, the input to the PID is usually preceded with low pass filter. This limits the high frequencies present (regarded as noise) in the input signal as the PID is usually only interested in

the slow changing trend of the variables under control. The low pass filter also prevents any sudden "jump" in the PID output when there is a sudden change in the input variable. Instead the PID output will "ramp" up to the new required value.

Another PID block type is provided, that with the equivalent transfer function of a PID with a low pass filter attached at the front end.

(2) INTEGRAL WIND UP OR SATURATION

The integral mode of the PID is introduced to eliminate steady state errors. As long as there is a deviation from the set point, the integral mode will give a changing control demand. Often the control demand cannot be achieved due to saturation of the actuator. This leads to a situation where the integral mode builds up to a large value i.e. integral "windup", a situation to be avoided.

For digital computation with floating point numbers, the range of the output of the PID is virtually infinite. When the output of the PID is connected to a D/A converter saturation occurs at the limits of the conversion range of the converter. Once the output limit is reached reset windup is avoided by holding the PID algorithm output at the saturation value until the computed increment requires the output to move back into the linear range.

(3) AUTO/MANUAL CONTROL

Auto/manual changeover is implemented so as to allow the operator to vary the output of the PID manually. With the incremental algorithm, the output is tracked automatically i.e. in switching over to the AUTO mode, separate initialization of the output is not required. Input to the manual value terminal in the PID function block may be from another functional block. The Auto/manual changeover is implemented by means of a flag (figure (2.8)).

(4) OPERATOR INTERFACE

Since the flexibility of the PID controller lies mainly in the ability to be "tuned" to suit a given control requirement, clear and easy access for the tuning must be provided. With the chosen algorithm it is possible to change each parameter independently. To allow this feature, the evaluation of the coefficients of the PID controller will be done during the RUN TIME, instead of being calculated earlier and storing only the results (for example T1/T2 can be calculated and stored as a single value). The penalties are slightly longer execution time and more memory storage. This allows the set point of the controller to be set by another block giving cascade control [BIBBERO 1977].

(5) BUMPLESS PARAMETER CHANGES

With the derivative mode in the PID algorithm, any changes in the set point will be differentiated, giving a large control output. This can be avoided by elimination of the set point from the derivative term. Figure (2.9) shows the set point derivative elimination.

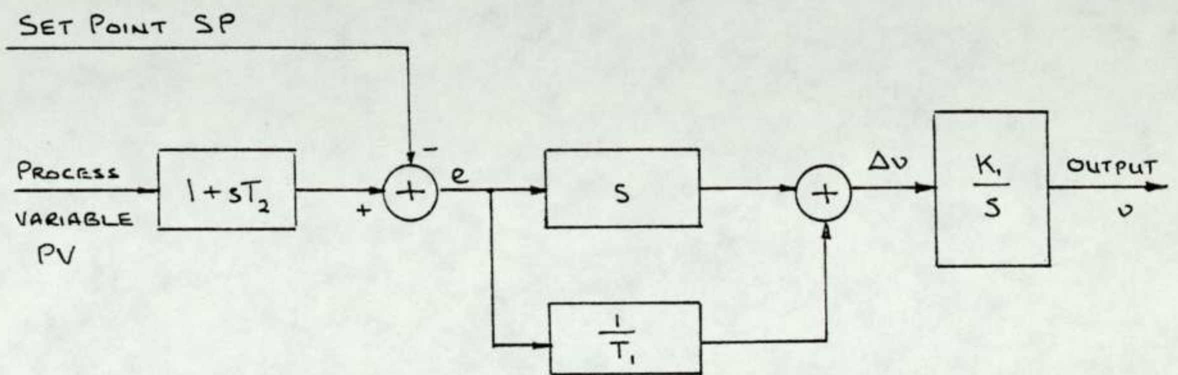


FIGURE 2.9 SET POINT DERIVATIVE ELIMINATION - PID

(6) EXTERNAL INTEGRATOR

The provision of the an external integrator for the PID allows the incremental changes to be used by other funcional blocks. For most cases this is not necessary and so the integrator is provided within the PID structure. If required, a further block functional type can be provided with external integrator.

(7) INITIALIZATION

During the steady state condition, the error samples are approximately the same i.e.

$$e(n) = e(n-1) = e(n-2)$$

In effect the derivative and proportional components of the PID are cancelled out, leaving only the integral component. For initialization the states of the internal error signal can be made equal to the first error input signal i.e.

$$e(-2) = e(-1) = e(0)$$

CHAPTER 3

GRAPHICS SYSTEM

3.1 GRAPHICS DISPLAY

Graphics devices fall into two categories -- interactive and passive. Passive devices are output-only devices (such as the graph plotter). Interactive devices (such as a graphics terminal with a light pen) permit human interaction through a variety of input mechanisms. A minimal interactive graphics workshop comprises a device for displaying the pictorial data and a device for accepting pictorial data as shown in figure (3.1).

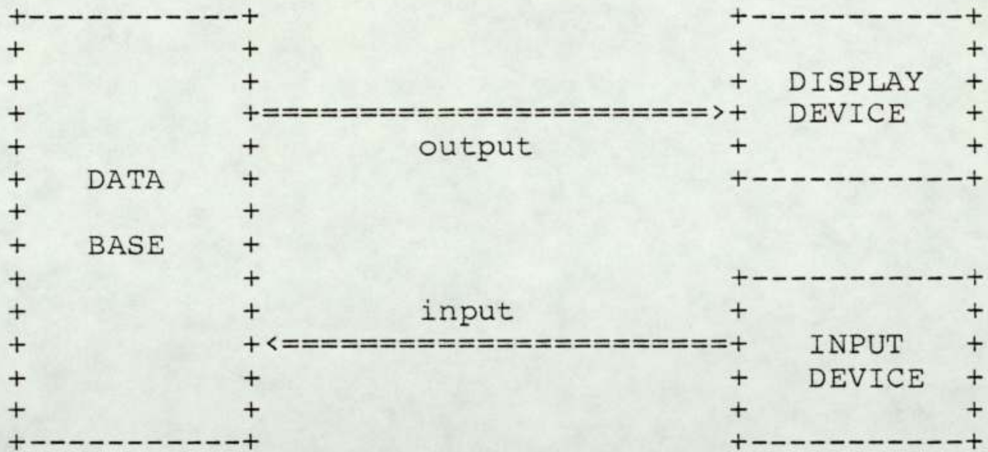


FIGURE 3.1 SIMPLIFIED MODEL OF INTERACTIVE PROCESS

Graphics displays of many kinds are used [HOBBS 1981], but three types of cathode ray tube (CRT) displays lead the field [MACHOVER 1977A], [McMANIGAL 1980].

(A) REFRESHED RASTER-SCAN DISPLAYS are similar to television CRT, requiring the generation of a matrix of intensity values which are fed to a TV monitor [MACHOVER 1977].

(B) REFRESHED DIRECTED BEAM DISPLAYS [LUCIDO 1978]. Lines are drawn by directing the electron beam across the screen. Such lines are called "vectors". Each vector is regenerated (refreshed) during the refresh cycle to give a constant picture. Unlike the raster-scan display, this only scans the paths between vector endpoints.

(C) DIRECT VIEW STORAGE TUBE DISPLAYS [PRESIS 1978]. This display uses the CRT that incorporate a means of storing displayed data and causing them to remain visible, without refreshing, once written. Lines are drawn in a similar way as refreshed directed beam display.

The different types of displays require different device drivers (software to operate the devices). The general graphics terminal requirements for various application areas have been identified by Carlson [CARLSON 1978] and Presis [PRESIS 1978]. Selection of a particular display depends on several factors including the resolution available and required, the ability to move objects about dynamically and the cost. The refresh-type terminal is, generally, more complicated and expensive than the storage

tube display. If the ability to display dynamically changing pictures is necessay, then the refresh-type display should be used.

3.2 GRAPHICAL INPUT MECHANISM

The ACM Graphics Standard Planning Committee has made its CORE proposal for a graphics standard [ACM 1979] and identifies the following 6 types of logical input devices :

(A) KEYBOARD for the typing of alphanumeric data

(B) BUTTONS for program function activation (e.g. function keys)

(C) STROKE DEVICES for the direct visual graphics entry e.g. the RAND tablet [DAVIS, ELLIS 1964] and the SKETCHPAD [SUTHERLAND 1963].

(D) VALUATORS for analog quantity entry (e.g. dials and meters)

(E) LOCATORS for position entry (e.g. joystick)

(F) PICKS for item selection (e.g. light pen and joysticks)

Of the six device types, the pick and the locator are the most useful for interactive graphics because they allow the user to interact directly with a graphical output by pointing [FOLEY, WALLACE 1974]. One form of the pick devices is the usage of the human finger to provide "touch

input" [HEROT 1978]. Design considerations for graphics input devices have been well discussed [OHLSON 1978], [NEWMAN, SPROULL 1979]. In this project, the input device is the joystick (performing the picking function) and a keyboard for alphanumeric data input.

3.3 GRAPHICS SOFTWARE

To achieve device-independence, the graphics software is usually divided into three parts -- an application program, a standard graphics package (for the manipulation of the graphical item) and a device driver [HEILMAN 1978], [NEWMAN 1978], [BERGERON 1978]. The application programs involve user written problem-solving programs, making full use of the graphics language for graphical input and output actions. The device driver is the program to activate the graphics hardware used.

Some graphics systems are built in the form of a graphics package based on an existing programming language [MEADS 1972], [GINO 1976], [CALCOMP 1974], [SIMPLEPLOT 1978]. Such graphics packages are sets of functions, subroutines (to provide the manipulation of the graphical objects) to be called by the application programs. The alternative is to choose an existing programming language and to extend and modify it to perform the graphics facilities [KULSRUD 1968], [SCHARK 1976]. The followings just a few examples of the languages being extended for graphics, Pascal [THALMANN 1981], Algol [JONES 1976] and

PL/I [SMITH 1971]. The basic graphics facilities must include means for moving the cursor (or drawing pen), drawing line vectors and writing alphanumeric characters.

3.4 GRAPHICS AND DATA STRUCTURE

An important component of the graphics programs (and other general programs) is the DATA STRUCTURE. The graphic data structure is the software representation of the model being operated upon. The choice of the data structure has an influential effect on the algorithms used. To quote Wirth, "The decisions about structuring data cannot be made without knowledge of the algorithms applied to the data" [WIRTH 1976]. The general criteria for the graphic data structure design are :

(1) adequate representation of the problem.

(2) sufficient flexibility.

(3) facilitating the extraction and manipulation of information.

(4) efficient in terms of memory storage space.

Much work has been done in the area of data structure for graphics. Sutherland's work on SKETCHPAD [SUTHERLAND 1963] has heavily influenced the development in this area. Sutherland defined a ring structure to handle a very common class of picture (called the "network graphs"). These pictures are usually interconnected in a network fashion,

and can be decomposed in lower levels of "subpictures" (other smaller pictures). Early work was also carried out by Ross and Rodriguez [ROSS, RODRIGUEZ 1963].

Sebsequent to SKETCHPAD, work has concentrated on the investigation and finding of more efficient graphic data structures. Various surveys of data structures for graphics have been carried out [GRAY 1967], [WILLIAMS 1971], [VAN DAM 1971]. Abrams [ABRAMS 1971] has discussed the advantages and disadvantages of the general purpose and tailored graphic data structures. Other workers have discussed specific data structures, for example for the drawing of lines [VAN DAM, EVANS 1967], [FRANKS 1968] and for remote computer graphics [COTTON, GREATOREX 1968].

3.5 GRAPHIC DATA STRUCTURE

Most graphical data structures are pointer-type structures, with such pointers being explicitly or associatively addressed. Programming languages designed to work with such pointers greatly facilitates the construction of the data structure e.g. PASCAL and ADA. The design of the pointer scheme is a critical part of any data structure [DODD 1969]. Some programming languages are developed for the implementing and manipulation of general-purpose graphic data structure e.g. LEAP [ROVNER, FLEDMAN 1968], L6 [KNOWTON 1969], ASP [LANG, GRAY 1968] and a system by Evans and Van Dam [EVANS, VAN DAM 1968].

It is decided to use a specially tailored data structure for the graphics in this application in order to achieve efficient data storage. The iconic or symbolic data structure (introduced by Tanimoto [TANIMOTO 1976]) for structuring pictorial data is a good basis upon which to build the tailored-purpose data structure. The scheme employs arrays whose elements are pointers to property list (table of attributes and other values) and pointers to other arrays [SHAPIRO 1978]. Linn [LINN 1979] has demonstrated the usage of such a scheme using tables to store the graphics information.

Tables containing entries of fixed size in consecutive locations can be used but the choice of the size is critical, since "overflowing" may occur, when the reserved locations are filled up and more entries are required. In this project, the iconic data structure scheme is used, with the basic form of data storage being the SINGLE LINKED LIST. All the graphical information is stored in "records" within the linked list. Discussion on the different types of records is given in section (4.1) (Graphic data structure).

3.6 GRAPHIC PICTURE STRUCTURE AND PROGRAMMING METHODOLOGY

The display diagram, composed of various block types and their interconnections and other textual information, is termed as a block-diagram or a "PICTURE". Each block diagram itself can contain other block-diagrams or graphical

pictorial entities. Such pictorial entity can be a simple block (this is the lowest level of decompositon, the very basic or fundamental "building brick"), or it can be composite in nature.

Composite blocks are of two types :

- -- the <u>subpicture</u>, which is a collection of blocks (simple or composite) to be treated as a single entity in the picture.
- -- the <u>macro block</u>, similar to the subpicture but with the restriction that its internal structure (i.e. the constituent components) may not be modified.

The main difference between the subpicture and the macro block lies in their usage. The subpicture is defined as a "once-off" entity, i.e. it is used only once in a picture. Its usage is usually for conceptual or aesthetic purposes. Therefore, it is permissible to modify its internal structure as long as the number of terminals (input and output) is left unchanged.

The macro block is intended for definition of a configuration of blocks for repeated use. The macro block definition is a "master" entity, and there can many usage of the block type ("instances") in a picture [SUTHERLAND 1963]. Modification of the macro block constituents is prohibited due to the "ripple" effect (any modifications in the master must be reflected through all the instances). It

is possible for the macro block and the subpicture to contain composite blocks themselves, resulting in a heirarchical structure. Figure (3.2) shows the heirarchical structure of the picture.

Recursive block definition is strictly not allowed, for example it is not permitted to define block A to contain block B, if block B is defined in terms of block A (explicitly or implicitly).

The composite blocks (macro block and subpicture) can be used for "information hiding". As long as the interface remains the same the internal composition may be varied without affecting the overall final results. Each composite block can be treated as a module with a single entry and a single exit. By using such composite blocks, the operator is encouraged to pursue the top-down programming methodology. The operator deals on only one level of decompositon at a time. Hence the advantages of the latest programming methodology can be reaped [YOURDON 1975].

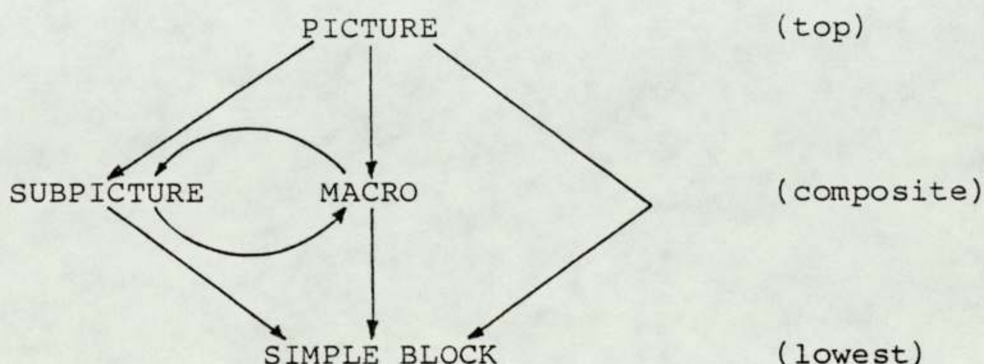


FIGURE 3.2 HIERACHICAL STRUCTURE OF PICTURE

CHAPTER 4

DATA STRUCTURE

This chapter deals firstly with the data structure adopted for effective computer graphics and secondly the data structure for efficient execution purposes during the run time phase.

4.1 GRAPHIC DATA STRUCTURE (GDS)

4.1.1 INTRODUCTION

The diagrammatical description of the pictorial scheme must be mapped (represented) onto a data structure that allows easy manipulation and modification. This data structure keeps all the information necessary to define the pictorial signal flow diagram. These include the blocks used (their types, any associated text) and interconnections.

The linked list is a versatile form of data structure. This contains of various "records" (set of values) all linked together via using linking pointers [WIRTH 1976], [HOROWITZ, SAHNI 1976]. The main advantage of the linked list is that it is efficient in memory utilisation and can be readily expanded to accommodate requirements for larger storage area. The linked list is used in this project to represent the mapping of the signal flow diagram into internal data representation. Consideration of various data structures for graphics can be found in [WILLIAM 1971] and

[GRAY 1967]. The consideration for the choice of a purpose-tailored data structure and its internal component is well discussed by Linn [LINN 1980]. One important consideration for the GDS is that it should be simple in nature and form. This allows the understanding of the "picture" by the operator in alphanumeric form, facilitating the manual entry of the graphical information in numbers (if this is ever required). The various records used in the GDS are :

(1) graphic block (GB) record

(2) graphic information (GI) record

(3) graphic text (GTXT) record

(4) input list (IL) record

(5) macro expansion (MX) record

(6) non-graphic data (NGD) record

The internal structure of each record will be discussed in the following sections with their inter-relationship.

4.1.2 GRAPHIC BLOCK (GB) RECORD

The graphic block record is diagrammatically represented in the figure (4.1). Most of the allocations within the record are explained in the figure itself, but the following requires more attention :

(A) The block number (BLKNO) is the identification number given to the block (by the operator or the system). Other records may be identified with a particular block record via its block number or the block pointer (a pointer to the start of the GB record).

(B) the GIPTR (graphic information pointer) links this block record to the appropriate GI (graphic information) record where more information about the attributes and pictorial form of the block is kept.

(C) The position of the base of the block on the screen of the graphical terminal is given by XPOS and YPOS. These, in association with the pictorial information in the graphic information record, are used for the actual drawing of the symbol of the block as well as for the identification of the block when "picked" by the joy-stick (performing the pick function).

(D) Following the TXTPTR (text pointer) is the input set of two elements for the first input terminal (if a block has no input terminal then the input set is not allocated). The input set is repeated for each input terminal of the block. A input set contains any connection information to that associated input terminal. Within the input set are two entries, the BLKPTR (block pointer) and the O/PNO (output terminal number). If the input is connected to, say the output terminal 2 of block 25, then the BLKPTR points to the start of the GB record of block 25 and the O/PNO is 2.

(E) The representation of any output connection is handled by the ILPTR (input list pointer) which follows any input set. Each output terminal of the block is allocated a ILPTR entry. When an output is connected to some other input terminals, then the ILPTR points to the appropriate input list (IL) record where the information about the connection is kept. Section (4.1.7) will deal more with the IL record. The IL record is output-oriented in the sense that only information of connection of output terminal to input terminals is kept.

As indicated in the figure, the GB record may vary in the length as this depends on the number of input and output terminals. The minimum length (minlength) of a GB record is eight (8) entries. The first input set are at location eight (8) and nine (9) after the start (or the base) of the GB record. For the Nth input the input set starts at

$$base + minlength + N*2 - 2$$

Any ILPTR (for the output) follows the input sets, the position of the Nth output being given by

$$base + minlength + NIP*2 + N - 1$$

where NIP is the total number of input for the block. The NIP can be found in the GI record.

4.1.3 GRAPHIC INFORMATION (GI) RECORD

This record (figure (4.2)) keeps most of the pictorial data of the block symbol and other information, most important of which are the number of inputs (NIP) and number of outputs (NOP).

The following elements of the record are discussed in greater details :

(A) the graphic information link (GILINK) is used to link one GI record to the next. This allows a search through all the GI records for any particular required GI record (usually identified by the type number).

(B) the type number (TYPENO) identifies the block function category. Each type number is unique in the whole system.

(C) the class number (CLASSNO) allows differentiation between the various classes of the block type provided. The four classes of type are :

(1) The simple block type (CLASSNO 0). This is the fundamental type of block and is a single block entity.

(2) The macro block type (CLASSNO 1). This is a collection of blocks (simple or otherwise) to be used as a separate single entity. It is analogous

to the "library" of general purpose subroutines in high-level programming languages. Modification of the internal structure of the macro block type is restricted as it may be used more than once in the picture.

(3) The <u>subpicture type</u> (CLASSNO 2), similar to the macro type is also composite in nature. The exception is that it is only a "one-off" block (It is only used once in a picture). So the internal structure can be modified with no restriction. The subpicture can be viewed more as a "normal" subroutine (as opposed to general purpose subroutines).

(4) The <u>picture type</u> (CLASSNO 3). This is the top in the hierarchical level of the graphical entities. A picture normally consists of the three previously mentioned block types, namely simple, macro block and subpicture. The picture can be regarded as a composite type, allowing all its information to be stored in the GI reocrds. This eliminates the need for another data structure to indicate the picture presence. Each picture is uniquely identified by the type number (in this case a large number in excess of 1000). Whenever the graphic editor is called, the GI records can be search to see if the editing is to be carried out on an existing picture or to create a new picture.

(D) The non graphic data pointer (NGDPTR), valid only for simple block record, links to the NGD (non graphic data) record where non-graphic information is stored.

(E) The MXPTR (macro expansion pointer) points to the macro expansion (MX) record, this is only valid for composite block records. The MX record keeps information about the internal structure of composite block.

4.1.4 MACRO EXPANSION (MX) RECORD

The macro expansion (MX) record is used only by the composite block type. Information about the internal structure of the block type is given in this record, represented in figure (4.3). The important elements of the MX record are as follows :

(A) MXLINK (macro block link). This is the pointer to the graphic block (GB) record of the first block in the internal structure of the composite type.

(B) Following the length element, is an input set of two elements representing the first input terminal. The BLKPTR refers to the start of the GB record of the internal structure which is providing the actual input terminal numbered (I/PNO). The relative displacement from the base of the MX record for the Nth input is given by N*2.

(C) Similar to the input, an output set (two elements) are used for each output terminal of the composite type.

The relative displacement from the base of the MX record of the output set for the Nth output is

$$NIP*2 + N*2$$

where NIP is the number of inputs for the composite type, defined in the GI record.

4.1.5 NON-GRAPHIC DATA (NGD) RECORD

The non graphic data record (figure (4.4)) contains all the information not related to the drawing of the symbol of the block. Non graphic data are those mainly used for the run time phase and the compiling phase.

The significant elements of the NGD record are as follows :

(A) The number of internal variables (NIV) is the number of variables used by the run time routine of the block during its execution.

(B) The general flag (GF), which is a collection of bits (in this case 16 bits) used to indicate attributes of the block type e.g. the first four bits are used to indicate if the block is of the retrospective or non-retrospective nature. Not all the bits are used at the present moment.

(C) The reset output flag (ROF) has 16 bits, with each bit set showing if a particular output need to be reset

during initialization or restarting of the run time computation.

(D) The logical input flag (LIF) (16 bits) is used to show if the input to a particular input terminal must be of the logical variable nature.

(E) The logical output flag (LOF) is similar to the LIF, except this shows if any output gives only logical value.

(F) The constant value input flag (CIF) is used to indicate if any input is expecting only constant value, similar to the LIF.

For LIF, LOF, CIF the input terminal numbers correspond to the bit positions in the flag word. Figure (4.5) shows the attributes of each of the flags in the NGD record.

4.1.6 GRAPHIC TEXT (GTXT) RECORD

Text can be used in a block diagram for naming, labelling and entering numerical values. Such text data are kept in the graphic text (GTXT) record (figure (4.6)). The storage of text characters requires special explanation. The first text character follows the TXTYPOS. Each text character stored in a byte (8 bits), allowing 2 characters in a word (16 bits). The length of the record is specified in complete words. A "blank space" is added to the text if the number of characters is odd. The entries TLINK, LENGTH, TXTXPOS, TXTYPOS occupy 4 words so that the number of text

character in a record is given by (L-4)*2 where L is the length of the record. If the text length is, say 6 then the total number of text characters stored is (6-4)*2 (i.e. 4 characters). The position of the text (TXTXPOS, TXTYPOS) is defined as the lower left corner of the first character in the text string.

There are seven (7) different forms of text records corresponding to the different purposes that the text serves. Graphic text can be classified under two categories :

(A) The block specific text. This is the text that may be vary with each block. The block specific text are of the following categories :

(1) the random text, mainly for commentary purposes

(2) the block name

(3) the engineering unit for the terminals

(4) the constant text, for entering numerical values

(5) the tag label, for tagging terminal for later identification.

(B) The type specific text. This is fixed for a block type and includes :

(1) the block type function

(2) the block terminal name

Figure (4.7) and figure (4.8) show all the types of text records. The various forms of text record are differentiated by their TXTXPOS elements. The normal range of the TXTXPOS and TXTYPOS is limited by the size of the screen (in our case 1024 and 780 respectively). So identification of the various type of text record is possible by using excessively large values for TXTXPOS in combination with the TXTYPOS. The normal random text is identified by its TXTXPOS and TXTYPOS having values less than 1024 and 780 respectively. If the values of TXTXPOS and TXTYPOS are 3000 and 0 then, this is a block name text record. The values used for identification of the text record are all given in the figure (4.7) and figure (4.8).

4.1.7 INPUT LIST (IL) RECORD

The input list record (figure (4.9)) keeps the information about the interconnection between the output of a block and the input terminals of blocks. The IL record is acessed by the ILPTR (input list pointer) entry in the GB record. If the first (1st) output of block, say block number 24 is connected to the input terminal numbered 2 of another block, say 31, then the ILPTR entry for the 1st output of block 24 is updated to link with a IL record. In this IL record, the BLKPTR entry points to the start of the GB record of block 31 and the I/PNO is 2.

When the output terminal of a certain block is connected to more than one input terminal, then the ILLINK entry is used to link all the appropriate input list records together. The example in section (4.1.8) will further show the use of the IL record.

4.1.8 RELATION BETWEEN THE RECORDS

By combining together and relating the various records, an extremely efficient and and flexible data structure is obtained. The figure (4.10) shows the records required for a simple block with all their possible inter-relations. The block record is linked to the GI record (of the correct TYPENO) by GIPTR. All block specific text is handled by the block text pointer (TXTPTR) and all the output connections by the input list records.

Figure (4.11) shows all the records required to represent a composite block with 3 constituent blocks. Here the MX record is used. A "picture" (the top level) is considered as a very special composite block type and thus is similar to figure (4.11).

4.2 THE GRAPHIC MENU

The graphic menu is a display of all the different block types that is provided in the system. The menu shows the shape, size and the number of inputs and outputs of the each block. The type number is displayed next to the block picture as additional information. The operator uses the graphic menu to select the required block type to be used during synthesis of the pictorial program. The blocks in the graphic menu are drawn in their normal size to assist the user in laying out the picture. Whenever a selection is made from the graphic menu, the selected item is redrawn to provide the visual feedback.

The data structure for the menu is different from that of the general graphics being simpler in nature. The menu data structure is shown in figure (4.12). It is separate from the graphic data structure, the only link being the pointer to the GI record. The menu is divided up into pages, where each page can have several block types (usually of a similar nature). For example one page can be used especially for the retrospective blocks. Each page in the menu is identified by the menu page number (MSPPNO). When using the menu to aid in the selection of the block types, pages of the menu can be "skipped" over, i.e. unwanted pages not displayed.

The menu data structure is in two forms, a menu page record and a page block record. The menu page record contains the menu page number (MSPPNO) and a pointer (MPBLINK) to the first page block within the page. All the menu pages are linked via the MPLINK element. The page block record contains the position of the base of the block within the menu in BXPOS and BYPOS. The GIPTR links to the GI record where all the drawing coordinates are stored. All page blocks within a page is linked by their MBLINK.

Pages in the menu gives the flexibility and the ability to add on at some later stages, if necessary, new functional block types. The flexibility even allows the graphical menu to be modified, for example to discard unwanted block types or to group frequently used block types onto one page.

4.3 RUN TIME DATA STRUCTURE (RTD)

4.3.1 INTRODUCTION

The run time data structure serves a different purpose from that of the graphic data structure. The graphic data structure is designed for easy manipulation and modification of the graphical items on display screen and for the extraction of data during the actual drawing phase. The run time data structure is more concerned with the execution of the functional blocks within the picture. Here the efficiency with regards to the referencing of the input and output connections is important. The run time data structure contains only the essential information for the execution, hence the graphical information can be removed. Consideration as to the basic run time data structure is given by Linn [LINN 1980]. The run time data records are as follows :

(A) The RSB (run time simple block) record

(B) The RCB (run time composite block) record

(C) The ULN (upwards link) record

(D) The RSTYPE (run time simple type) record

(E) The RCTYPE (run time composite type) record

(F) The RTXT (run time text) record

(G) The RVT (run time value table).

The following sections will discuss each record in greater detail.

4.3.2 RUN TIME BLOCK RECORDS

There are two forms of block records in the run time data structure, one for simple block, and another for the more complex composite block (figure (4.13)). The following elements of the record are discussed :

(A) The ELINK (execution link) is used to link all the RSB records in the order in which the blocks will be processed during the execution phase.

(B) The LBLKNO (local block number) is the block identification number (provided by the operator or the graphic system) during the synthesis of the signal flow block diagram phase. This is not necessarily unique, due to the usage of composite blocks.

(C) The GBLKNO (global block number) is the block identification number provided by the system during the conversion from the graphic data to the run time data. This GBLKNO is unique for each block during run time.

(D) The RVTPTR (run time value table pointer) is used to indicate the starting location of the run time data file of each block. The data file is stored in the RVT. The data file consists of entries for the input and output values and any internal variables necessary for the execution of the block.

(E) The IPPTR (input pointer) is used to repesent the connection between the blocks. One IPPTR entry is allocated for each input terminal of the block. This entry is used as a pointer to the RVT, pointing to the location in which the appropriate ouput value is stored. (This output is that which is connected to the input terminal.)

(F) The MXPTR (macro expansion pointer), valid only for the composite block record, points to the first run time block record of the internal structure of the composite block.

The composite block record has no allocation for any IPPTR entry. The composite block during run time will be expanded down to simple blocks. Since all the simple blocks will contain the interconnection information in their IPPTR entries, it is not necessary to allocate IPPTR for the RCB (run time composite block) record.

4.3.3 UPWARDS LINK (ULN) RECORD

This is only used in conjunction with the RCB record. Figure (4.14) shows a ULN. The only element of interest is the CBNPTR (composite block record pointer) which is used to point back to the RCB record. The ULN record is used to indicate the end of the internal structure of the run time composite block. Figure (4.15) shows an example of a run time composite block with the ULN record. The ULN is

necessary as it provides the means of termination of the constituent records and the link back to the RCB itself.

4.3.4 RUN TIME TYPE RECORDS

The run time type records are used to store all the necessary run time (RT) information about the block type. As there are two categories of blocks, the simple and the composite, two different forms of type records are required. They are :

(1) the RT simple type (RSTYPE) record.

(2) the RT composite type (RCTYPE) record.

4.3.4.1 RUN TIME SIMPLE TYPE (RSTYPE) RECORD

The RSTYPE record keeps all information for only the simple blocks, figure (4.16). The RSTYPE record is fixed in length, having 14 entries. Some of the entries are discussed below :

(A) the TLINK (type link). This is a pointer for linking all the RT type records together. It provides the routing for a search of any required type number (TYPENO).

(B) the TXTPTR (text pointer) is used to point to any associated RT type specific text records.

(C) the various flags GF, ROF, LIF, LOF, CIF are identical to that in the NGD (non-graphic information) record and

have been discussed in section (4.1.5).

(D) The CLASSNO (class number). This entry is always zero for the RSTYPE record.

4.3.4.2 RUN TIME COMPOSITE TYPE (RCTYPE) RECORD

The RCTYPE record differs from the RSTYPE record since it must contain more information about the internal structure of the block. Figure (4.17) shows a RCTYPE record. Most of the elements in the record are similar to those in the RSTYPE record. The following entries are different :

(A) CLASSNO (class number). The class number is used to differentiate between the various class of the block type, namely macro or subpicture.

(B) MXPTR (macro expansion pointer). This is a pointer to the first block in the internal structure of the composite block type.

(C) the input set of LBLKNO (local block number) and I/PNO (input terminal number). The definition of the input terminal of the composite block type in relation to the actual input terminal of the block in the internal structure is handled by this input set. An input set is allocated for each and every input terminal. For a particular input terminal set, the LBLKNO refers to the local block number of the block within the composite type ; and the I/PNO is the numbered input terminal of

that block.

(D) the output set of two entries, LBLKNO and O/PNO (output terminal number). This defines the output terminal of the composite type in terms of the constituent block (see discussion of input set in (C)).

4.3.5 RUN TIME TEXT (RTXT) RECORD

The format of the run time text differs from that of the graphic text record. The run time text keeps only the necessary data such as the block name, type name, engineering units and the terminal names. The random text in the graphic system is now not required. The general form is shown in the figure (4.18).

The main element is the TEXT, which contains all the text characters. Each text character is allocated a byte (8 bits), and the most significant bit (MSB) is used as a termination flag. When the character is the last of the text string, then its MSB is set.

The run time text can be classified as follows :

(A) type-specific text. This record stores the text specifically related to the block type i.e. the block type function text and the terminal name text. The type-specific text is fixed for all blocks of the same type and is given during the definition of the block type.

(B) block-specific text. This record stores text relating to each specific block in the block-diagram. For example this may contain the block name and any engineering units to be associated with the terminal data values.

Figure (4.18) shows the two run time text records.

4.3.6 RUN TIME VALUE TABLE (RVT)

The RVT is a floating point array where values of variables and parameters of the run time simple blocks are stored. These are grouped together to give a "module" of data file for each block. Further discussion on the allocation of the storage location can be found in section (9.1) (allocation of storage for run time block).

GRAPHIC BLOCK (GB) RECORD

```
+-----------------------------------------------
+ (GLINK) (LENGTH) (LLINK) (BLKNO) (GIPTR)
+-----------------------------------------------

                -------------------------------
                     (XPOS) (YPOS)  (TXTPTR)
                -------------------------------

                 -----------------------------------------+
                   (BLKPTR)(O/PNO) .. (ILPTR1)  (ILPTR2)    +
                 -----------------------------------------+
                   (     input     )    (output1) (output2)
```

LEGEND

(GLINK) global link to the next graphic block (GB) record linking all the block records.
(LENGTH) length of the record.
(LLINK) local link to graphic block record which is in the same "picture level" (hierarchical level).
(BLKNO) block identification number.
(GIPTR) graphic information (GI) record pointer.

(XPOS)] position of the base of block
(YPOS)] x & y coordinates.

(TXTPTR) pointer to associated block-specific text (GTXT) record.

(BLKPTR) pointer to block (GB) record.
(O/PNO) output terminal number.

(ILPTR1) pointer to the input list (IL) record (output 1).
(ILPTR2) pointer to the input list (IL) record (output 2). When the output is connected, then the ILPTR is pointed to the appropriate IL record.

```
+-----------------------+
+   (BLKPTR) (O/PNO)    +   INPUT SET
+-----------------------+
```

This is the output terminal of graphic block to which the input terminal of the present block is connected to.

FIGURE 4.1 THE GRAPHIC BLOCK RECORD

GRAPHIC INFORMATION (GI) RECORD

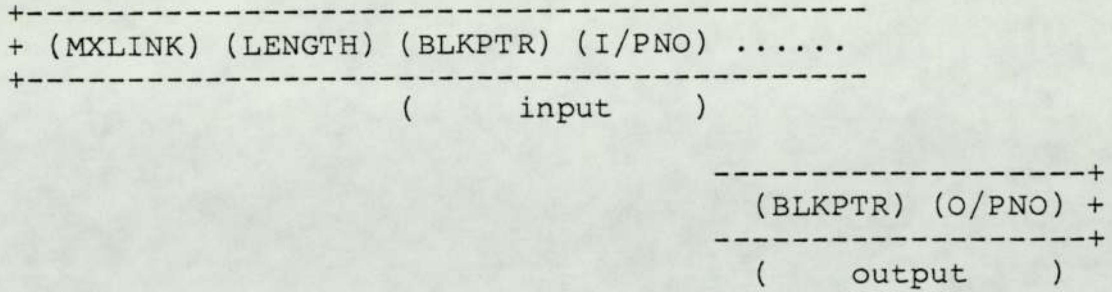
```
+------------------------------------------------------
+ (GILINK) (LENGTH) (TYPENO) (CLASSNO) (NIP) (NOP)
+------------------------------------------------------

    ------------------------------------------------------
      (MXPTR) (TXTPTR) (NGDPTR) (BLKNOXPOS) (BLKNOYPOS)
    ------------------------------------------------------

              ----------------------------------------------+
                 (I/P COORDS) (O/P COORDS) (PIC COORDS)  +
              ----------------------------------------------+
                 (...pictorial drawing coordinates...)
```

LEGEND

```
(GILINK )      link to the next graphic information (GI)
               record linking all the GI records.
(LENGTH )      length of the record.
(TYPENO )      type number, identification of block function.

(CLASSNO)      class number, various classes :
                         0 - simple type
                         1 - macro type
                         2 - subpicture
                         3 - picture

(NIP       )   number of inputs.
(NOP       )   number of outputs.

(MXPTR     )   pointer to the macro expansion (MX) record,
               valid for composite type only.
(TXTPTR    )   pointer to associated type specific text
               (GTXT) record.
(NGDPTR    )   pointer to the non-graphical data (NGD)
               record.

(BLKNOXPOS )   ] starting position of the block
(BLKNOYPOS )   ] number, relative to the base of block.

(I/P COORDS)   ] coordinates of the input & the output,
(O/P COORDS)   ] relative to the base of the block.

(PIC COORDS)   coordinates for the drawing of the symbol of
               the block.
```

FIGURE 4.2 GRAPHIC INFORMATION RECORD

MACRO EXPANSION (MX) RECORD

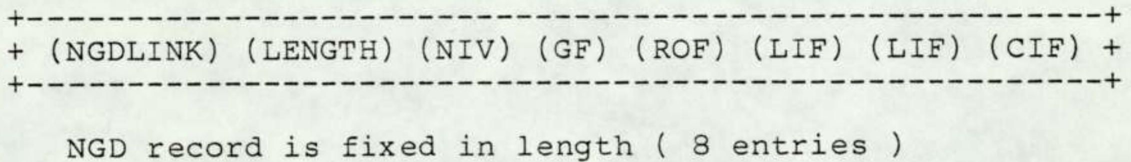


LEGEND

(MXLINK) pointer to the first GB record of the "internal structure" of the composite block
(LENGTH) length of the record
(BLKPTR) pointer to the block record (GB)
(I/PNO) input terminal number
(O/PNO) output terminal number

FIGURE 4.3 MACRO EXPANSION (MX) RECORD

NON GRAPHICAL DATA (NGD) RECORD



NGD record is fixed in length (8 entries)

LEGEND

(NGDLINK) pointer to the next NGD record
(LENGTH) length of record
(NIV) number of internal variables
(GF) general flag to indicate attributes of block type
(ROF) reset output flag
(LIF) logical input flag (16 bits) with each bit set to represent that the corresponding input terminal is logical in nature.
(LOF) logical output flag (16 bits)
(CIF) constant input flag (16 bits), each bit showing if the corresponding input terminal is expecting constant values.

For the LIF, LOF, COF the terminal number corresponds to the bit position in the computer word .

FIGURE 4.4 NON GRAPHIC DATA (NGD) RECORD

ATTRIBUTES OF THE FLAGS IN THE NGD RECORD

GENERAL FLAG (GF)

```
 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+ ..............................+X  X  X  X +
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

1st four bits used to indicate nature of block

- retrospective
- non-retrospective
- input interface
- output interface

LOGICAL INPUT FLAG (LIF)

```
 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+ ............................ 0  1  0  0  1 +
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

1st and 4th input are logical in nature, i.e. they are expecting only binary (logical) values.

LOGICAL OUTPUT FLAG (LOF)

```
 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+ ............................ 0  1  0  0  1 +
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

1st and 4th output are logical in nature, i.e. the outputs are only binary (logical) values.

CONSTANT INPUT FLAG (CIF)

```
 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+ ............................ 0  1  1  0  0 +
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

3rd and 4th input are expecting only constant values

FIGURE 4.5 ATTRIBUTES OF THE FLAGS IN NGD

GRAPHIC TEXT (GTXT) RECORD - GENERAL FORM

```
+-----------------------------------------------+
+ (TLINK) (LENGTH) (TXTXPOS) (TXTYPOS) (( TEXT )) +
+-----------------------------------------------+
```

LEGEND

```
(TLINK  ) pointer to next text (GTXT) record
(LENGTH ) length of the record
(TXTXPOS) ] starting position of the text relative to
(TXTYPOS) ] the base of the block.
((TEXT )) location where the TEXT is stored.
```

FIGURE 4.6 GRAPHIC TEXT RECORD

TYPE SPECIFIC TEXT RECORDS

BLOCK TYPE NAME TEXT RECORD

```
+---------------------------------------------+
+ (TLINK) (LENGTH) (TXTXPOS) (TXTYPOS) ((TEXT)) +
+---------------------------------------------+
                   (=2000)    (=0)
```

TERMINAL TEXT RECORD

```
+-----------------------------------------------+
+ (TLINK)(LENGTH) (TXTXPOS) (TXTTERMNO) ((TEXT)) +
+-----------------------------------------------+
                  (=2000)
```

NOTE : the values of TXTXPOS and TXTYPOS given in bracket are the dummy values of the entries used to identify the various different text records. So if the TXTXPOS=2000 and TXTYPOS=0, this is a block type name record.

LEGEND

```
(TLINK     )   link to the next graphic text (GTXT) record.
(LENGTH    )   length of the record.
(TXTXPOS   )   x-coordinate of starting position of text.
               Also used to differentiate between various
               form of GTXT record. The normal range of
               txtxpos is 0-1024.
(TXTYPOS   )   y-coordinate of starting position of text.
(TXTTERMNO)    terminal number to which the terminal text is
               associated.
```

FIGURE 4.7 GRAPHIC TYPE SPECIFIC TEXT RECORDS

BLOCK SPECIFIC TEXT RECORDS

RANDOM TEXT RECORD

```
+-----------------------------------------------------+
+ (TLINK) (LENGTH) (TXTXPOS) (TXTYPOS) ((TEXT)) +
+-----------------------------------------------------+
```

BLOCK NAME TEXT RECORD

```
+-----------------------------------------------------+
+ (TLINK) (LENGTH) (TXTXPOS) (TXTYPOS) ((TEXT)) +
+-----------------------------------------------------+
                   (=3000)   (=0)
```

ENG UNIT TEXT RECORD

```
+-----------------------------------------------------+
+ (TLINK) (LENGTH) (TXTXPOS) (TXTO/PNO) ((TEXT)) +
+-----------------------------------------------------+
                  (=3000)
```

CONSTANT TEXT RECORD

```
+-----------------------------------------------------+
+ (TLINK) (LENGTH) (TXTXPOS) (TXTI/PNO) ((TEXT)) +
+-----------------------------------------------------+
                  (=4000)
```

TAG TEXT RECORD

```
+--------------------------------------------------------+
+ (TLINK) (LENGTH) (TXTXPOS) (TXTTERMNO) ((TEXT)) +
+--------------------------------------------------------+
                   (=5000)
```

NOTE : values of the TXTXPOS, TXTYPOS given in bracket are dummy values of the entries used for the identification of various text record type. So if TXTXPOS=3000 and TXTYPOS=0, this is a block name text record.

LEGEND

(TLINK) link to the next text (GTXT) record
(LENGTH) length of the record
(TXTXPOS) x-coordinate of starting position of text, also used to differentiate between various form of GTXT record. For random text, the txtxpos is always less than 1024.
(TXTYPOS) y-coordinate of starting position of text
(TXTO/PNO) output terminal number to which the engineering unit text is associated.
(TXTI/PNO) input terminal number to which constant value text is associated.
(TXTTERMNO) terminal number to which terminal text is associated.

FIGURE 4.8 GRAPHIC BLOCK-SPECIFIC TEXT RECORDS

INPUT LIST (IL) RECORD

```
+-------------------------------------+
+ (ILLINK) (LENGTH) (BLKPTR) (I/PNO)  +
+-------------------------------------+
```

IL record is fixed in length (4 entries)

LEGEND

(ILLINK) pointer to the next input list (IL) record used when the output of a block is connected to more than one input terminal
(LENGTH) length of record
(BLKPTR) pointer to the block record
(I/PNO) input terminal number

```
+------------------+
+ (BLKPTR) (I/PNO) +
+------------------+
```

This set refers to the input terminal (given by I/PNO) of the block number (found in BLKPTR).

FIGURE 4.9 INPUT LIST RECORD

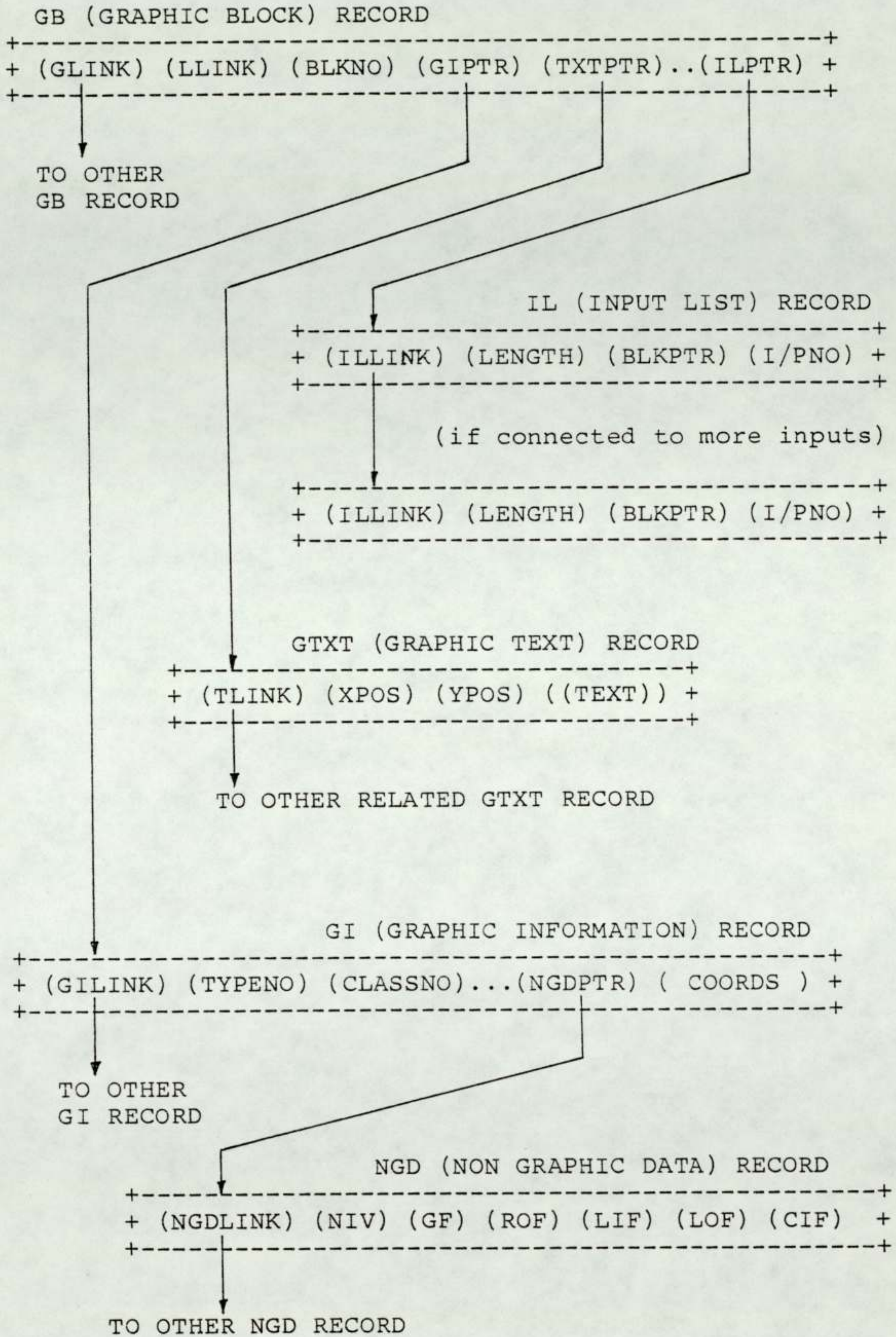


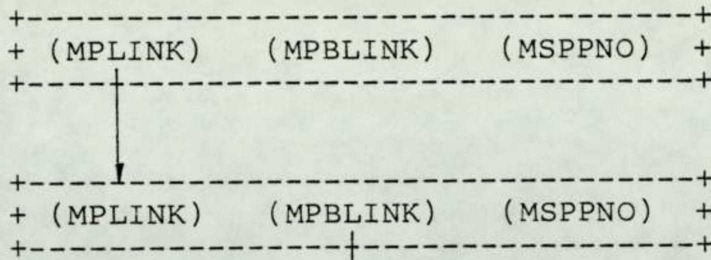
FIGURE 4.10 RELATION BETWEEN RECORDS FOR SIMPLE BLOCK

```
GB RECORD ( composite block )
+-------------------------------------------------------+
+ (GLINK) (LLINK) (BLKNO) (GIPTR)      (ILPTR) +
+-------------------------------------------------------+

                                                      GI RECORD
+----------------------------------------------------------------+
+ (GILINK) (TYPENO) (CLASSNO) (NIP) (NOP)  (MXPTR) +
+----------------------------------------------------------------+

  TO OTHER
  GI RECORD

                                                      MX RECORD
        +--------------------------------------------------------+
        + (MXLINK)  (BLKPTR) (I/PNO)...(BLKPTR) (O/PNO) +
        +--------------------------------------------------------+
                 (     input      )  (     output     )

                                        GB RECORD
        +---------------------------------------+
        +(GLINK)   (LLINK) (BLKNO) (GIPTR) +
        +---------------------------------------+

                                        GB RECORD
        +---------------------------------------+
        +(GLINK)   (LLINK) (BLKNO) (GIPTR) +
        +---------------------------------------+

                                        GB RECORD
        +---------------------------------------+
        +(GLINK)   (LLINK) (BLKNO) (GIPTR) +
        +---------------------------------------+
```

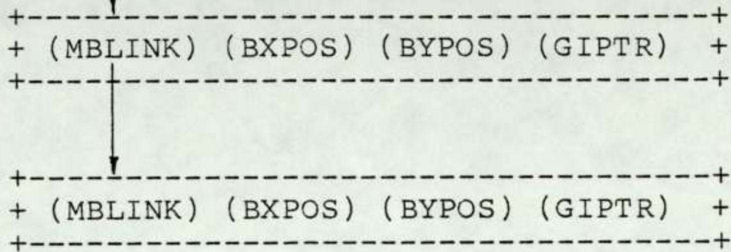
LEGEND
GI = GRAPHIC INFORMATION
MX = MACRO EXPANSION
GB = GRAPHIC BLOCK

FIGURE 4.11 RELATION BETWEEN RECORDS FOR COMPOSITE BLOCK

MENU PAGE RECORD



PAGE BLOCK RECORD



LEGEND

(MPLINK)	menu page link, pointer to the next page in menu.
(MPBLINK)	menu page block link, pointer to the first block in this page.
(MSSPNO)	menu page number, identification purpose
(MBLINK)	menu block link, pointer to next block in the same page.
(BXPOS)	] position of the block within this page.
(BYPOS)	]
(GIPTR)	pointer to the GI (graphic information) record.

FIGURE 4.12 GRAPHIC MENU DATA STRUCTURE

RUN TIME DATA STRUCTURE

RSB (runtime simple block)

```
+---------------------------------------------------------
+ (LINK) (LENGTH) (LLINK) (ELINK) (LBLKNO) (GLBKNO)
+---------------------------------------------------------

        ----------------------------------------------+
         (TYPEPTR) (TXTPTR) (RVTPTR) (MXPTR)  (IPPTR1) +
        ----------------------------------------------+
```

RCB (runtime composite block) fixed length of 10 entries

```
+---------------------------------------------------------
+ (LINK) (LENGTH) (LLINK) (ELINK) (LBLKNO) (GBLKNO)
+---------------------------------------------------------

              ----------------------------------------+
                (TYPEPTR) (TXTPTR) (RVTPTR) (MXPTR)    +
              ----------------------------------------+
```

LEGEND

```
(LINK   ) general link for all run time block records.
(LENGTH ) length of each block record.
(LLINK  ) local link, use for linking local blocks in
          composite block internal structure.
(ELINK  ) execution link, linking blocks in the proper
          processing order (valid for simple block only).

(LBLKNO ) local block number, identification purposes.
(GBLKNO ) global block number, identification purposes.

(TYPEPTR) pointer to run time type record.
(TXTPTR ) pointer to any run time block-secific text (RTXT)
          record of the block.

(RVTPTR ) pointer to the run time value table ( RVT ) which
          contain the data file for this block.
(MXPTR  ) applies to composite block, pointer to first block
          of the internal structure. For simple block
          record, this is a null entry.

(IPPTR1 ) first input pointer to the run time value table
          (RVT) location of the output to which it is
          connected to. If input is not connected then IPPTR
          is null.
```

FIGURE 4.13 RUN TIME BLOCK RECORDS

UPWARDS LINK (ULN) RECORD - for composite block only

```
+----------------------------------------------+
+ (LINK) (LENGTH) (LLINK) (CBNPTR) (GBLKNO)    +
+----------------------------------------------+
```

LEGEND

(LINK) pointer to link to next run time block record.
(LENGTH) length of the record (fixed of 5 entries).
(ILLINK) null entry.
(CBNPTR) pointer back to the composite block (RCB) record, used to terminate the internal structure of a composite block.
(GBLKNO) null entry.

The (LENGTH) and the (GBLKNO) may be used to identify the ULN record from the other run time data records.

FIGURE 4.14 UPWARDS LINK RECORD (ULN)

```
RCB RECORD
+-------------------------------------------------------------+
+(LINK)(LLINK)(LBLKNO)(GBLKNO)(TXTPTR) (RVTPTR) (MXPTR) +
+-------------------------------------------------------+-----+

                                                  RSB RECORD
   +---------------------------------------------------------+
   +(LINK) (LLINK) (LBLKNO) (GLBKNO) .. (MXPTR) (IPPTR) +
   +---------+-----------------------------------------------+

                                                  RSB RECORD
   +---------------------------------------------------------+
   +(LINK) (LLINK) (LBLKNO) (GLBKNO) .. (MXPTR) (IPPTR) +
   +---------+-----------------------------------------------+

                                                  RSB RECORD
   +---------------------------------------------------------+
   +(LINK) (LLINK) (LBLKNO) (GLBKNO) .. (MXPTR) (IPPTR) +
   +---------+-----------------------------------------------+

                     ULN RECORD
   +-----------------------------+
   +  (LINK) (LLINK)  (CBNPTR)  +
   +-----------+--------+--------+
```

LEGEND
RCB - RUN TIME COMPOSITE BLOCK
RSB - RUN TIME SIMPLE BLOCK
ULN - UPWARDS LINK

FIGURE 4.15 EXAMPLE OF RUN TIME COMPOSITE BLOCK OF 3 SIMPLE BLOCKS

RUN TIME TYPE RECORDS

```
RUN TIME SIMPLE TYPE (RSTYPE)
+-------------------------------------------------------
+ (TLINK) (LENGTH) (CLASSNO) (TYPENO) (SWIX) (TXTPTR)
+-------------------------------------------------------

   ------------------------------------------------------+
    (NIP) (NOP) (NIV) (GF) (ROF) (LIF) (LOF) (CIF)      +
   ------------------------------------------------------+
                        ( ........ flags ......... )
```

LEGEND

```
(TLINK  ) link to the next run time type record
(LENGTH ) length of this record, fixed at 14 entries.
(CLASSNO) class number of type, used for differentiate
          various classes of type, namely
                 0 -- simple type
                 1 -- macro block type
                 2 -- subpicture type
                 3 -- picture type

(TYPENO ) type number identification
(SWIX   ) switch index, for execution uses only for simple
          type
(TXTPTR ) txtptr, pointer to any associated type-specific
          text (RTXT) record

(NIP    ) number of input
(NOP    ) number of output
(NIV    ) number of internal variables

(GF     ) general flag (set of bits) for indication of
          attributes of the type e.g. retrospective & non-
          retrospective

(ROF    ) reset output flag
(LIF    ) logical input flag - bit is set to indicate the
          corresponding input of a block type must be
          logical (binary) in nature
(LOF    ) logical output flag, as above applying to the
          output
(CIF    ) constant input flag - bit indication that the
          input expects a constant value.
```

FIGURE 4.16 RUN TIME SIMPLE TYPE RECORD

```
RUN TIME COMPOSITE TYPE (RCTYPE)
+---------------------------------------------------
+ (TLINK) (LENGTH) (CLASSNO) (TYPENO) (MXPTR)
+---------------------------------------------------

       ---------------------------
          (TXT) (NIP) (NOP)
       ---------------------------

         -----------------------------------------------+
            (LBLKNO) (I/PNO)   .....  (LBLKNO) (O/PNO) +
         -----------------------------------------------+
              (.. input ..)             (.. output ..)
```

LEGEND

```
(TLINK  ) link to the next run time type record.
(LENGTH ) length of this record.
(CLASSNO) class  number  of  type,  used  for  differentiate
          various classes of type, namely
               0 -- simple type
               1 -- macro block type
               2 -- subpicture type
               3 -- picture type
(TYPENO ) type number identification.
(TXT    ) txtptr, pointer  to  any  associated  text  (RTXT)
          record.
(NIP    ) number of input.
(NOP    ) number of output.
(MXPTR  ) pointer to the first block record in the  internal
          structure of the composite type.
(LBLKNO ) local  block  identification,  in  the   internal
          structure.
(I/PNO  ) input terminal number.
(O/PNO  ) output terminal number.
```

```
+-----------------+
+ (LBLKNO) (I/PNO) + input set to define the input
+-----------------+ terminal of the composite type.

+-----------------+
+ (LBLKNO) (O/PNO) + output set to define the output
+-----------------+ terminal of the composite type.
```

Which input or output terminal of the composite to which the above sets refer to depends on their relative location in the type record.

FIGURE 4.17 RUN TIME COMPOSITE TYPE RECORD

RUN TIME TEXT (RTXT) RECORD - GENERAL FORM

```
+--------------------------------+
+ (TLINK) (LENGTH)  ((TEXT))     +
+--------------------------------+
```

LEGEND

(TLINK) pointer to the next associated RTXT record.
(LENGTH) length of the record in complete words.
((TEXT)) text characters. Each character is stored in a byte.

TYPE-SPECIFIC TEXT

```
+----------------------------------------------------+
+                                  .             .   +
+ (TLINK) (LENGTH)  BLOCK FUNCTION TERMINAL1 NAME    +
+----------------------------------------------------+
```

BLOCK-SPECIFIC TEXT

```
+----------------------------------------------------+
+                           .       .                +
+ (TLINK) (LENGTH)  BLOCK NAME ENG UNIT              +
+----------------------------------------------------+
```

NOTE : The dot above the character indicates that the most significiant bit (MSB) of the byte is set.

FIGURE 4.18 RUN TIME TEXT RECORD

CHAPTER 5

USER INTERFACE AND GRAPHIC EDITOR

This chapter deals with the design and considerations of the man-machine interface and gives a general description of the graphic editor and its facilities.

5.1 USER INTERFACE DESIGN

5.1.1 INTRODUCTION

The user interface design of a graphics system is of utmost important and a main contributer to the success of the system. A poorly designed interface is difficult to learn and to use. Considerations for designing of the user interface of a graphics system [NEWMAN 1979], [GOOD 1981] includes :

(1) the command language

(2) the feedback

(3) the information display

Each consideration will be further dealt with below having regards to the user interface used in this project. (The display terminal used is a direct view storage tube display with a joystick control device and a keyboard.)

5.1.2 COMMAND LANGUAGES

The command language should be designed to be as simple as possible and logically consistent so that it is easy for the user to learn. The followings are some methods of communication on which the command language may be based,

(A) KEYBOARD DIALOGUE

This is the simplest style of command language. The graphics system "prompts" the user to supply all the necessary information by printing question messages. The choice of answer may also be restricted to a set of responses offered to the user together with the question.

(B) KEYBOARD COMMAND LANGUAGE

An example for the graphics system is the command to delete a block, say block number 25, from the display as follows :

DELETE BLOCK 25

This form of command language requires much less code than the keyboard dialogue. The processor must only recognize a limited vocabulary of commands. The user is however confronted with the task of memorising or keeping a record of the command set.

(C) FUNCTION KEYS

The commands are given with the aid of a set of function keys. Each function key can be assigned a specific function, such as DELETE or CREATE. It is possible to assign certain alphanumeric keys to act as function keys. An example is the character key D to act as the DELETE function key.

(D) MENU-DRIVEN LANGUAGE

This is a very general and flexible style of command language for the following reasons :

(1) The menu displays plainly on the screen the full range of the available options. A well designed menu can even be made to display different list of options during different stages of using the graphics system.

(2) The menu can be easily changed e.g. to include new commands. When the command menu is displayed on the screen, the required command can be selected from the menu by use of the joystick.

In implementation the function keys approach was adopted. The menu-driven language would be preferred but with the storage tube display terminal, the writing speed is limited. A trained user can operate much more quickly using single key strokes rather than having to wait for a menu to be displayed. However menu-driven facilities are also

included, particularly to describe the available block types (GRAPHIC MENU). The graphic menu lists much information about the block type, including the shape and the size of the block symbol and the number of inputs and outputs.

5.1.3 FEEDBACK CONSIDERATIONS

Feedback is an important ergonomical factor to be taken into account during user interface design. In the graphics system, visual feedback serves to assure the user that the system is responding to his command. One essential form of feedback is the "selection feedback" whenever some form of menu is used. When a choice is made by the user, selection feedback (e.g. highlighting or inversion) indicates that the system is responding to the selected item. Whenever the user selects a block, that block is redrawn to provide the visual feedback. (Highlighting and inversion are not permitted in storage tube display.)

Another form of feedback is the "command feedback". This serves to indicate to the user that the system is responding to his non-visual commands (commands not affecting the display directly). One example is the saving of the system present status and data. The command feedback also prevents the user from giving a command when the system is not ready to receive it (the system may be busy doing some other functions). In this project, the command feedback is provided by the changing the shape of the cursor on the screen. Whenever the system is ready to accept commands, the

cursor is a blinking pointer (looking like an arrowhead). When the system is not ready for commands, a blinking alpha cursor (a shaded rectangle) appears.

5.1.4 INFORMATION DISPLAY

This section concerns the effectiveness in displaying information. The important question is "how should the information be presented on display in the most effective manner to promote the interaction between user and the graphics system ?". Problems in information display generally relate to overall layout or the representation of the object.

(A) OVERALL LAYOUT

Here utilization of the screen area is considered with regards to the picture display and the menu. The screen could be divided up into "windows", allowing the menu and the picture to appear simultaneously. Since the screen area is not very large, it is decided to use the whole screen for the picture display area. The graphic menu will be drawn separately upon user request.

(B) OBJECT DISPLAY

The graphical representation of the object item is chosen on the basis that (1) it must reinforce the user conceptual view of the item and (2) the symbol preferably is one that the user is accustomed to. In this implementation, the symbology used is "borrowed"

from the commonly used and popular control block diagram representation.

5.2 THE GRAPHIC EDITOR

5.2.1 INTRODUCTION

The graphic editor enables interactive communication between the graphics system and the user for the synthesis of the GPL (graphical programming language) programs. The editor is used to create a new picture or to modify an existing picture. A picture is defined as a collection of related graphical entities which are displayed together. The facilities provided by the graphic editor are dependent on the graphics hardware used. The following section will give a brief description of the graphics hardware used.

5.2.2 GRAPHICS HARDWARE DESCRIPTION

The display terminal used is the TEKTRONIX 4051 terminal, of the 4050-series storage CRT (cathode ray tube) display type [TEKTRONIX 1976A]. This is a popular and commonly used storage terminal which has been emulated by refresh type terminals. It has a drawing area of 19 cm by 15 cm and has the addressing capability of 1024 x 780. A 4051 data communication interface [TEKTRONIX 1976B] (a RS-232 compatible interface) is used to connect the display terminal to a minicomputer as the host computer. The minicomuter used is the TEXAS INSTRUMENTS 990/10 model [TEXAS INSTRUMENTS 1978]. The interface acts as a doorway, allowing streams of characters to flow to and from the graphics drawing hardware in the 4051 terminal. This

interface allows the 4051 terminal to emulate a TEKTRONIX 4012 computer display terminal. Figure (5.1) shows diagrammatically the hardware equipment used in the project.

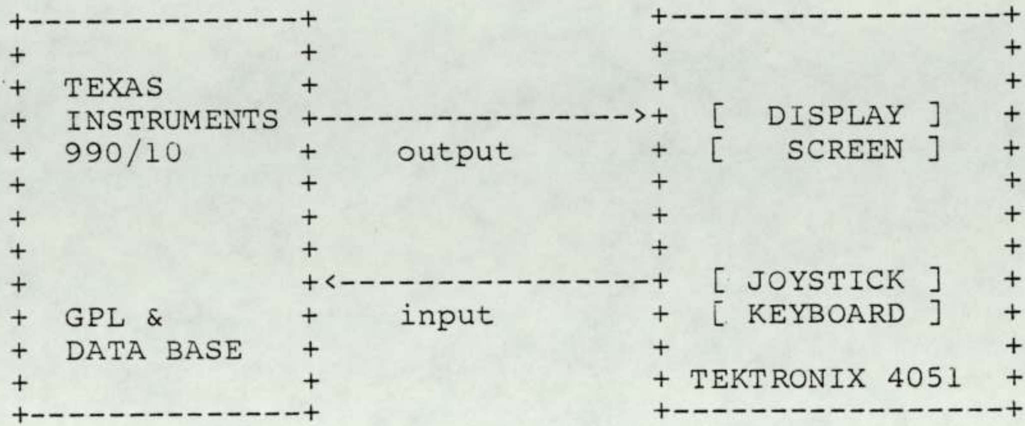


FIGURE 5.1 EQUIPMENT USED IN PROJECT

By putting the data communication port into the terminal mode, the 4051 terminal becomes interactive and three other submodes are allowed [TEKTRONIX 1976B]. The ALPHA SUBMODE in which the incoming characters are displayed as lines of text. In this mode, the terminal works like an alphanumeric terminal except that the characters can be positioned anywhere on the screen. The GRAPHIC SUBMODE in which the incoming characters are decoded as screen locations, which are used for vectors (lines) drawing, allowing picture to be drawn. The GIN SUBMODE in which the location of the graphic cursor can be sent out through the data communication port, with the character typed on the keyboard. The graphic cursor is controlled by a joystick device (TEKTRONIX 4952) [TEKTRONIX 1976B], which is used to move the cursor to any desired position on the screen.

Although storage displays do not permit dynamic movement of image (e.g. "dragging"), they do generally provide better resolution and are less expensive than the refresh types [PRESIS 1978]. Because of the storage display of the 4051 terminal, deleted blocks and connections do not disappear from the screen until the picture is redrawn. Redrawing of a picture may take a long time (depending on the complexity) due to the slow speed of drawing, so a picture is only redrawn if requested by the user. Futhermore, if necessary, the editor can be made to redraw the picture after each alteration by setting a software switch. The graphic editor can be easily adapted to handle a refresh type terminal with a light pen, by making the refresh type terminal emulate a storage tube display terminal. The basic editing actions will remain unchanged.

5.2.3 PICK FUNCTION

The pick function, using the joystick and the GIN submode, allows the operator to "pick" graphical entity on the srceen. The picking function is performed by comparing the cursor position to the position of the graphical entities. Identification algorithms for the pick function are discussed in the paper by Weller and co-workers [WELLER 1980]. General picking selection techniques are discussed by [NEWMAN 1979]. The pick window is defined as the area around the cursor within which the graphical item is chosen. The pick window is of a fixed size in this case. Any ambiguity of which graphical entities picked (when there are

more than one in the window) is resolved by picking the one nearest to the cursor location. The selected item is redrawn to provide the visual feedback to the operator.

5.2.4 EDITING OF PICTURE

This section on the graphic editor concentrates mainly on the creation of a new picture or modification of an existing picture, using elements from a menu of basic block types. A simplified model of the graphic editor is shown in figure (5.2).

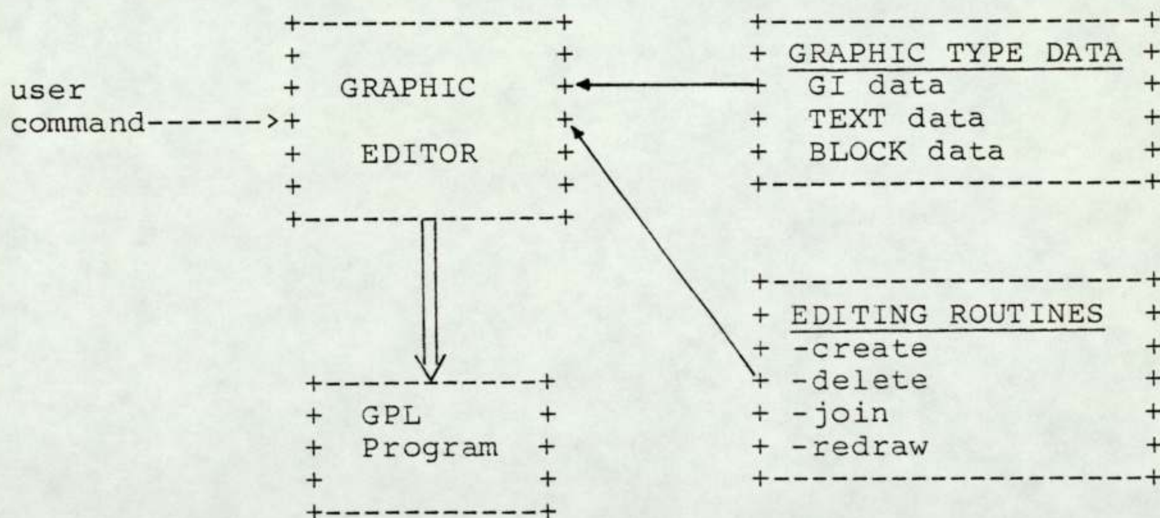


FIGURE 5.2 SIMPLIFIED MODEL OF GRAPHIC EDITOR

Editing is initiated by giving the appropriate keyboard command and by giving the picture type number (unique for each picture). An example of the editing keyboard command set is given in the table (5.1).

GRAPHICAL EDITOR COMMANDS

```
P = picture naming              J = join terminal
R = redraw of picture           W = draw guidelines

? = dump GDS                    C = create block
: = save GDS                    D = delete connection
X = open trace file             B = delete block
Z = close trace file            * = create junction
G = get trace flags             Q = quit edit
O = open or "unfold" composite block
```

SELECT

```
S = select
+ = next menu page      ) used for controlling of
- = previous menu page  ) display of menu page
```

TEXT EDITING

```
T = select block for text edit
B = block name text edit
N = block number edit
R = random text edit
C = constant input edit
E = engineering unit edit
F = finish text edit
```

TABLE 5.1 GRAPHIC EDITOR COMMAND CHARACTERS

If the picture type number cannot be found in the GI (graphic information) records, then a new (empty) picture is created. Otherwise the existing picture will be drawn on the display screen. Addition of a block within a picture involves the selection of the required block type. This selection can be carried out in three ways :

(1) by selecting a block of the required type (already present in the picture).

(2) by selecting through the graphic menu (a display of all the available block types).

(3) by specifying the required type number. (This avoids the delay due to drawing of the graphic menu.)

The cursor can then be located in the required position and the block created by a keyboard character command.

The editor can be used to connect terminals of blocks. Terminals are "picked" by the cursor (controlled by the joystick) and validity checks applied before the connection of the two terminals is completed. The validity checks concentrate mainly on the correctness of the intended connection in the signal flow sense, i.e. an output to an input, or vice versa. Incorrect connections include attempts to connect two outputs (or inputs) together and the input terminal being already connected to another output. Any attempted incorrect connection will be reported back to the operator via a message on the display screen and the attempted editing action ignored. The difficulty involved in usage of the junction block is discussed in section (5.2.5).

Deletion of a connection between blocks is relatively easy to implement. The simplest case is deletion at the input terminal, since there can be only one connection to an input terminal. Deletion at an output terminal requires more care, since it may be connected to several inputs. Deletion of a block requires even more attention (see section on recursive deleting in [SUTHERLAND 1963]). When a block is

deleted, then all connections to and from it must be deleted as well. The editor removes the appropriate block record and updates all the connection information.

The editing of the picture (with subpictures and macro blocks) is enhanced by the possibility to "expose" or "unfold" the internal composition of the composite blocks (i.e. to show the internal structure). The GB (graphic block) records and the GI (graphic information) records are scanned for the next lower level of the picture to find the required composite block. Validity check will prevent a macro block from being modified by any attempted editing.

The creation of a composite block is very similar to the creation of a picture, except for the need to modify the class number entry in the GI (graphic information) record to the appropriate class number. (The class numbers are 1 for macro block, 2 for subpicture and 3 for picture.)

5.2.5 CONNECTION TO AND FROM JUNCTIONS

A junction block is normally used for aesthetic purposes since it just passes the signal along. A junction block in diagrammatical representation is just a big "dot", and its input and output terminals are coincident. Whenever a junction is "picked" for connection, the terminal which is chosen cannot be immediately identified as either the input or the output. This has to be worked out from other connection already made to the junction. A more complex

situation arises in the case of a multi-segment connection line (formed by connecting several junctions only). Here the whole length of the multi-segment connection must be scanned to determine the flow of the signal.

5.2.6 TEXT EDITING

Text associated with the block type (type specific text) may not be modified while creating a picture. Type specific text are text of the block function name and terminal names and can only be modified by recreating the block type.

Block specific text (text associated with a particular block) can by edited using the editor. Text editing is initiated for example by first selecting the block in which the text is to be edited. The complete capabilities of the text editing include :

(1) addition or removal of commentary random text.

(2) the changing of the block number subject to the restriction that the new block number is unique within the picture.

(3) addition or removal of the engineering unit text to be associated with the terminal data value.

(4) the association of constant numerical value to terminals.

(5) the changing of the block name.

(6) the tagging of any terminal with a label so as to enable easy indentification during the execution phase.

These changes are incorporated in the graphic text data structure (see section (4.1.6) for a description on the graphic text data structure).

CHAPTER 6

THE GRAPHIC COMPILER

6.1 INTRODUCTION

After the graphical programming language (GPL) program is configured, it is passed over to the graphic compiler. The main output from the graphic compiler is the program structure table (PST), a numerical data representation of the picture. The PST comprises various inter-related run time data structure records (section 4.3). The functions of the graphic compiler can be outlined as the followings :

(1) data transformation of type specific data from graphic to run time requirements.

(2) expansion of the macro blocks and subpictures.

(3) error checking of the picture, which is divided generally as :

(a) missing essential connections.

(b) illegal data type connections.

(c) "algebraic loops" of non-retrospective blocks.

(4) sequencing the blocks for execution.

(5) allocation of data tables for the blocks.

(6) initialization of the data tables.

(7) listings and messages.

A simplified model of the graphic compiler is shown in figure (6.1).

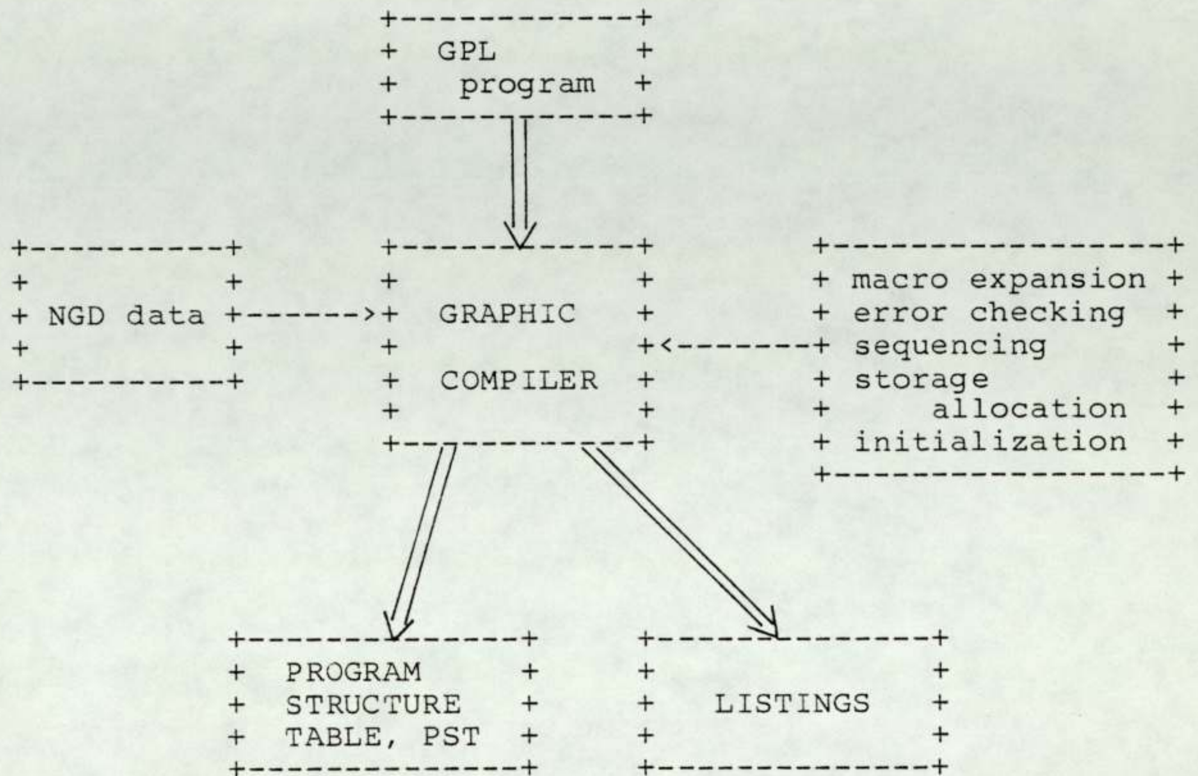


FIGURE 6.1 SIMPLIFIED MODEL OF GRAPHIC COMPILER

6.2 TRANSFORMATION OF GRAPHIC TYPE TO RUN TIME TYPE

After the graphical synthesis of the picture scheme, the transformation of the graphics data (in terms of the blocks diagram) to the run time data (for more efficient execution and storage space) is to be carried out. Of the run time data, there are two groups, one of the type records and the other the block records.

This section will deal mainly with the type specific record information transformation. The type specific information are those relating to the particular type of the functional block provided and not with the graphical outline of the block. All the run time type records are formed using the information previously provided in the graphic data. The run time type record provides a compact form of all the essential information. (The actual elements of the text record can be found in section (4.3.4) (run time type record)).

There are two ways of dealing with the type-specific transformation, namely by transforming when the graphic compiler is called, implying that the transformation will be carried out with each compilation ; or by providing already transformed run time type record, leaving only newly provided functional block type to be transformed. The difference is only in the execution time required for the compilation.

6.3 EXPANSION OF MACRO BLOCK AND SUBPICTURE

The use of macro block or subpicture in a picture leads to compactness of the graphical representation and allows blocks that are related logically or computationally to be grouped as one entity. In most cases, the operator is only interested in the relationship between the inputs and outputs, and not the "internal structure computation", of the composite blocks.

During run time, the macro block and subpicture must be expanded, that is their internal structure are represented in terms of simple blocks. This results in only one single level of simple blocks. For further details on treatment of the subpicture and macro block, refer to the section (7.1) (run time treatment of composite blocks).

6.4 ERROR CHECKING BY GRAPHIC COMPILER

6.4.1 MISSING CONNECTIONS

For the graphic compiler to function properly, certain ESSENTIAL connections in the picture must be present. If the operator may by mistake or otherwise has left out the connections and it is the function of the connection error checking module to detect these connections and inform the operator of the result. Essential connections include input (normal variable, logical or constant in nature) that may have been left "undefined". A simple but trivial example is that of one input of the multiplier function block is connected. With the other input left unconnected (i.e. undefined), the output of this block is obviously ambiguous.

6.4.2 ILLEGAL DATA TYPE CONNECTIONS

The inputs and outputs of the functional blocks are allocated one of the following data types,

(1) system normal variable

(2) logical variable

(3) constant.

A system variable is one whose value is represented in floating point number whose range is determined by the implementation in the run time processor. A logical variable is represented as an integer with value one or zero (1 or 0) i.e. it is binary in nature. The constant data type applies particularly to the input, implying that a constant input value is expected at that input. It is also in floating point number representation and the constant values are held in a constant pool data table.

Obviously it is improper or "illegal" to try to connect an output of the type "system variable" to an input of type "logical variable". Constant input can also be checked if the input value is actually a constant, since constant values are stored in the constant pool data table. This data table is a read only data base during the run time processing of the blocks. This data type-checking is similar to the type-checking function of the new programming languages (e.g. Pascal and Ada).

6.4.3 ALGEBRAIC LOOPS OF NON-RETROSPECTIVE BLOCKS

Closed loops of non-retrospective blocks are not to be allowed in the picture, to prevent "algebraic loops". A algebraic loop is a situation such that the instantaneous value of any output is fed back to the inputs of the blocks in a closed loop. With reference to the figure consisting of a summer, a multiplier and a non-linear function generator, their interconnection cause the instantaneous output of the non-linear block to be fed back to the adder. This results in a algebraic loop, since a closed loop is formed.

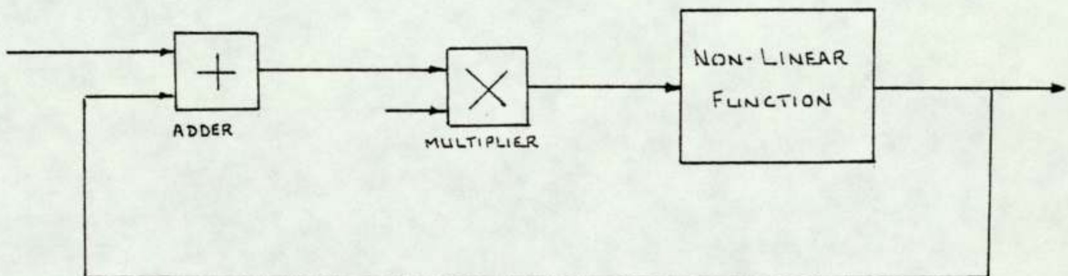


FIGURE 6.2 EXAMPLE OF ALGEBRAIC LOOP

The presence of any algebraic loops in a picture may not be obvious, especially if the picture is complex and/or using various macro blocks and subpictures. Further details on the algebraic loops can be found in section (7.2) (closed loop of blocks). These algebraic loops, if any, will have to be detected and if found to be reported to the operator. This will be expanded upon in the loop detection section (7.3).

6.5 SEQUENCING THE BLOCKS FOR EXECUTION

After the expansion of the macro blocks and the subpictures, and the completion of the error checking routines, a valid interconnected set of simple blocks results. It is necessary to sequence or sort the blocks to determine the processing order of these blocks during the run time execution. The blocks are either retrospective or non-retrospective in nature. Non-retrospective blocks require their present input values for the computation of their outputs while the retrospective blocks do not. Sequencing is only carried out among the non-retrospective blocks since all the retrospective blocks, by definition, are independent in their processing order.

Sequencing is necessary to ensure that the computation conforms to the data flow requirements of the block diagram. For example if a non-retrospective block has an input value derived from an output of another block, the other block must be executed first so that the input value is updated before it is used [ROSKO 1972]. The general discussion on sequencing (including the processing order) can be found in section (8.2).

6.6 ALLOCATION OF DATA TABLES FOR BLOCKS

Each block can now be allocated memory storage for the output, input and internal variables (if any) in the RVT (run time value table). These will be grouped together to give a data file for each block. Such data file gives a clear representation of a functional block in the data table, allowing easier recognition of the block than otherwise. The choice of the allocation of variables of block is discussed in the section (9.1).

The general format of data file is clearly shown in the RVT (run time value table) in figure (6.3), i.e. the current outputs, current inputs, past inputs and internal variables in that order. The relation between the run time simple block (RSB) and its data file is also illustrated. The RVTPTR in the RSB record points to the first entry in the data file (RVT). This first entry is usually the first output variable. Thus the next output variable is located at RVTPTR+1 and can be refered to by RVT [RVTPTR+1]. The first input variable is located at

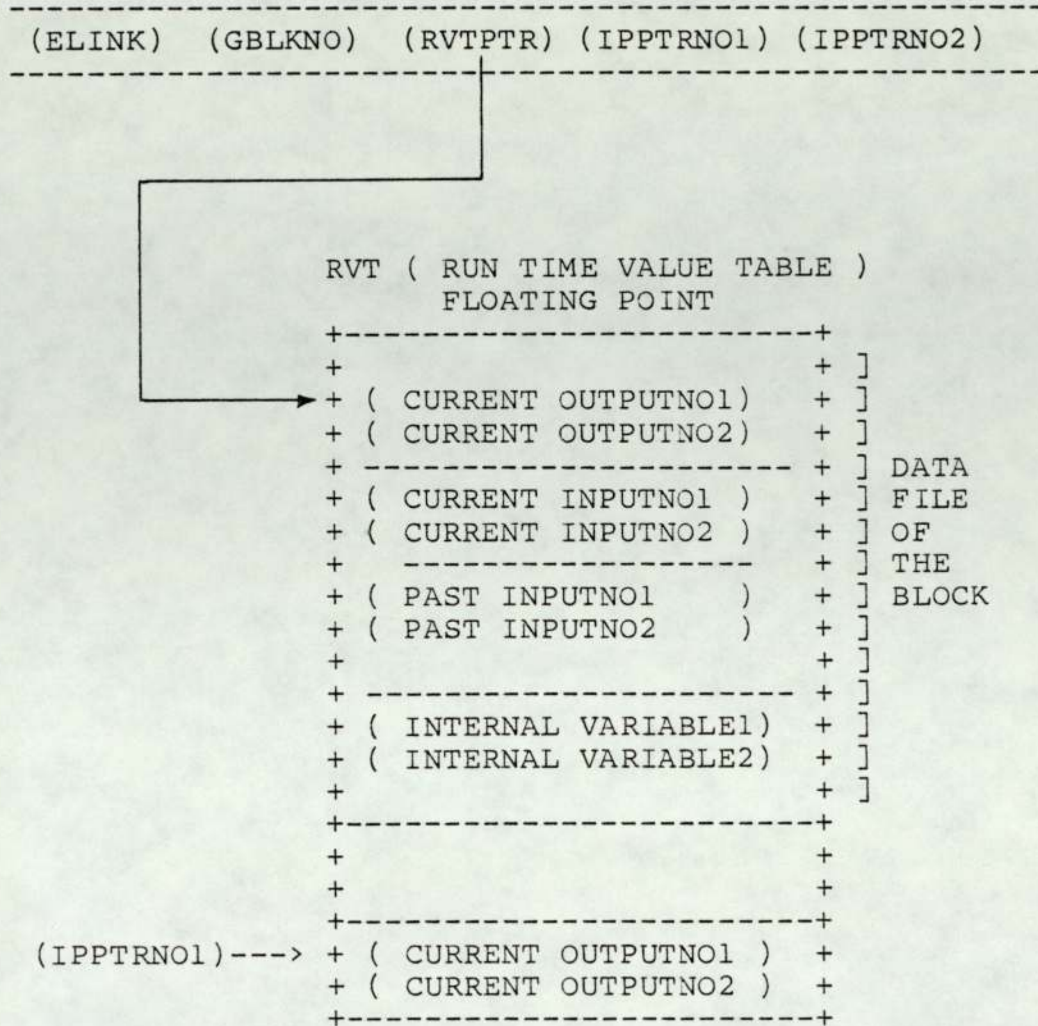
RVTPTR + NOP

where NOP is the number of output of the block. The location of the Nth input is at

RVTPTR + NOP + N - 1

Knowing the NIP (number of input), the start of the internal variables can be found and easily accessed. The IPPTR entry in the run time simple block record is used to indicate the

connection of an output to the input. IPPTRNO1 points to the output location (in RVT), that output being connected to the first input terminal. So the first input variable value is given by RVT [IPPTRNO1], similarly the rest of the inputs can be found. Direct addressing of the output to the input is used to ensure more efficient and faster processing of the blocks.



<u>LEGEND</u>

(ELINK)	Link to next block to process
(GBLKNO)	Global block number, identification purpose
(RVTPTR)	RVT pointer, points to first entry in data file
(IPPTRNO1)	Pointer to location in RVT where the output is stored, this output being connected to this input.

<u>FIGURE 6.3 RELATION BETWEEN DATA FILE AND BLOCK RECORD</u>

6.7 INITIALIZATION OF THE RUN TIME DATA TABLE

The data files of the blocks can now be initialized from the operator-provided values (with more values, computed from given values). The general philosophy involved in initialization is explained under section (9.2) (initialization of functional block). The initialization also allows the checking of the compatibility of the provided values to the terminals. The data table and the run time block data structure are now ready for execution.

6.8 LISTINGS AND MESSAGES

This function of the compiler provides the error reporting and general reporting facilities. Error reporting includes any error or warning messages arising from the previously mentioned functions of the compiler. This is consistent with the listings, error and warning reporting of conventional high level programming language compilers. The reporting facilities of the graphical compiler can be divided into the followings :

(1) errors concerning the usage of macro blocks. These concentrate on the completeness and legality of the macro block and subpicture. By completeness, it is meant that the macro block is complete in its internal structure as opposed to a un-completed attempt at the construction of a macro block. Usage of such an invalid marco block or subpicture (detectable since they are

flagged) is forbidden. Legality of a subpicture includes the usage of a subpicture only once in a picture.

(2) connection errors, include error listing of any missing essential connections, illegal data type connections and "algebraic" loops. Reports could include block and terminal references where the connection error occurs.

(3) initialization report. Messages about operator-provided values, any incompatibility between such values, and any initialization difficulties (including insufficient initial values to provide for all the initialization requirements).

General reporting would be on the listings of the blocks used in the picture and their interconnections in numerical form as opposed to the graphical form. This provides a separate form of documentation. Information about number of blocks used, their types, and total amount of memory storage for the graphic and run time representation can be provided.

6.8.1 EXAMPLES OF LISTINGS AND WARNINGS

Error reporting and warning listing can be implemented by providing a source of all the messages in a listing library. When an error is encountered, the appropriate message is selected and then listed. Some examples of the messages are provided below :

(1) Subpicture.

SUBPICTURE USED MORE THAN ONCE

(BLKID) (BLKID)

(2) Connection errors.

INPUT UNDEFINED (BLKID)(TERMID)

ILLEGAL CONNECTION

(BLKID)(TERMID) (BLKID)(TERMID)

ALGEBRAIC LOOP (BLKID) (BLKID) (BLKID)

(3) Initialization.

INCOMPATIBILE INITIAL VALUES TYPE

(BLKID) (TERMID)

INCOMPLETE INITIALIZATION

where (BLKID) is the identification block number and (TERMID) the terminal number of the block.

CHAPTER 7

COMPILATION ACTIVITIES - PICTURE VALIDATION

7.1 RUN TIME TREATMENT OF COMPOSITE BLOCKS

There are two ways of treating a composite block (i.e. a collection of blocks to be treated as a single entity). Composite blocks are of two types - macro and subpicture, the main difference being the ease to which to modify the internal composition in the subpicture type but restricted in the macro block type.

They can be looked upon as analogous to <u>subroutines</u> (or procedures) in normal high level programming languages. There will be no identity problems as shown in the figure (7.1A). The actual composite block is entered into the run time block records and a DUMMY structure (similar to the definition of a subroutine) is required to define its internal composition. The dummy structure is to be used during the processing of the composite block. Variables storage in the run time value table, RVT, must be allocated for each and every block inside the composite block, this storage being replicated for every occurrence of the composite block. Figure (7.1A) shows the effect on the run time data structure. By treating composite blocks as subroutines, little space in the run time block record is required to indicate each occurrence but, in common with subroutine calls, more parameter passing is required, resulting in slower execution. Sequencing of the blocks in

their appropriate processing order now become more difficult, when composite blocks of a mixture of retrospective and non-retrospective blocks are used.

If the composite blocks are treated as macros, they must be expanded down to the lowest level possible, thus resulting in only one level of simple blocks. Similar to the expansion of macro code statements, this requires more data storage (compared with the subroutine approach) since the structure of the macro is repeated for every occurrence of the composite block. Figure (7.1B) shows the run time data structure using the macro approach. The advantage of this approach lies in the simplicity of the execution of the blocks, no dummy structure is required and the differentiation of composite block and simple block is not required. Sequencing is simplified, applying only to one level of all simple blocks. Unfortunately, the identity of the composite block is destroyed during the macro replacement. Another data record (the RCB, run time composite block, record) is used to maintain the composite block identity.

In this project, the macro approach of the treatment of the composite blocks is adopted. So one of the functions of the graphic compiler is to deal with the macro replacement (expansion) of all the composite blocks down to the level of simple blocks.

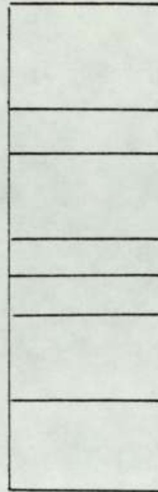
BLOCK RECORDS
(simplified)

BLK NO.	TYPE
10	A
11	C
2	B
27	D
5	C

composite

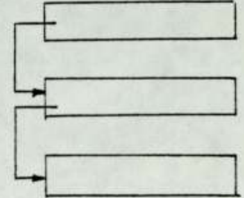
composite

RVT
value table



blk 10
blk 11
blk 2
blk 27
blk 5

block
record



dummy
structure
composite
type 'C'

FIGURE 7.1A SUBROUTINE AND EFFECT ON RUN TIME DATA

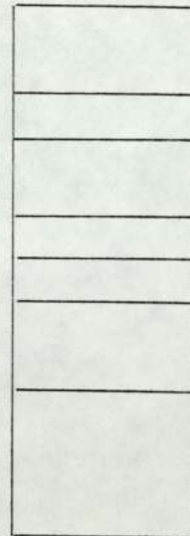
BLOCK RECORD
(simplified)

BLK NO.	TYPE
10	A
n1	t1
n2	t2
.	.
2	B
27	D
m1	t1
m2	t2
.	.

composite
block 11

composite
block 5

RVT
value table



blk 10
blk 11
blk 2
blk 27
blk 5

FIGURE 7.1B MACRO AND EFFECT ON RUN TIME DATA

FIGURE 7.1 RUN TIME TREATMENT OF COMPOSITE BLOCK

7.2 CLOSED LOOPS OF BLOCKS

In the graphical picture scheme, closed loops of interconnected blocks may be configured in order to realise the required control function. Such loops fall into two categories :

(A) loops with non-retrospective blocks only

(B) loops with at least one retrospective block.

An example of the first category is found in the obtaining the square root function as shown in the figure (7.2). This is a common practice in the analog computing [KORN, KORN 1972].

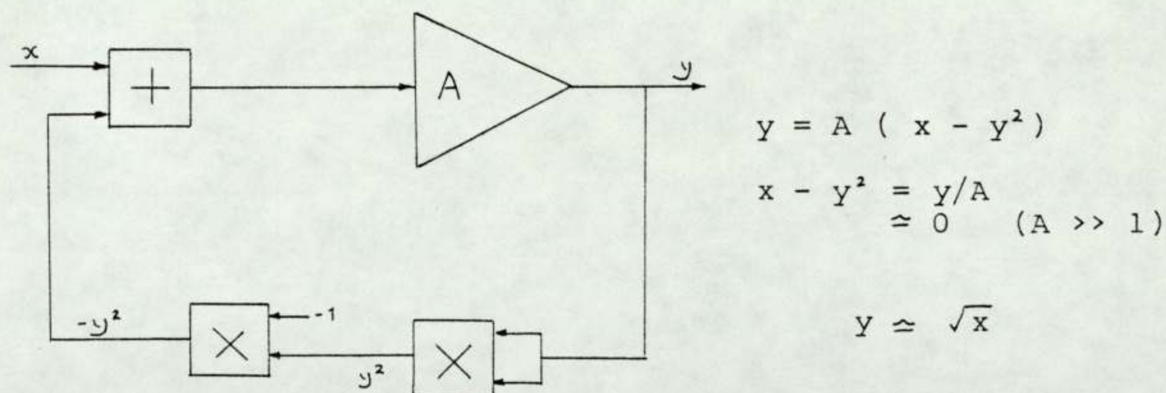


FIGURE 7.2 SQUARE ROOT FUNCTION BLOCK DIAGRAM

In this case the blocks forming the closed loop are all non-retrospective, i.e. their outputs are instantaneous functions of their inputs. An ALGEBRAIC LOOP is formed equivalent to an implicit expression of the form

$$y = F (x, y)$$

These loops present no problems to analog computing since the constituent blocks are processed simultaneously. The sequential operation of digital computing requires one block to be processed at a time and this leads to the requirement of an iterative solution. Solutions of implicitly expressed function via iterative process may be acceptable in simulation programs, but they are unacceptable in process control applications for the following reasons :

(a) the number of iterations varies, depending on the specific data involved i.e. the processing time is undeterminate.

(b) the convergence of the solution cannot be guaranteed.

The problem can be avoided by creating a new functional block which defines the required function explicitly. Algebraic loops are therefore not to allowed in the pictorial scheme, and it is a part of the function of the graphic compiler to test for the existence of such loops and to terminate processing with an error message if a loop is found.

The second situation exists, when at least one block in the closed loop is retrospective in nature i.e. its outputs are not dependent on the current value of its inputs. Such a block effectively "breaks" up the loop, allowing the blocks to be processed in a sequential manner [SPECKHART, GREEN 1976].

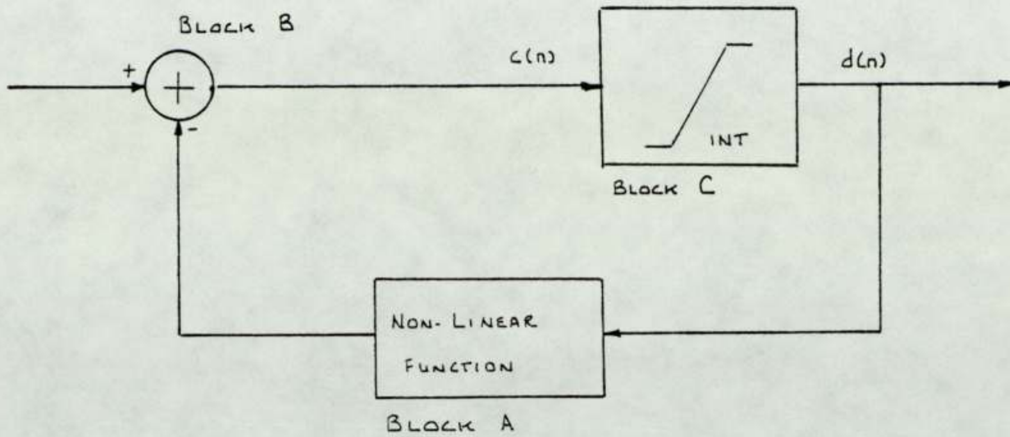


FIGURE 7.3 NON-LINEAR FILTER BLOCK DIAGRAM

The integrator block C is implemented as a retrospective algorithm (see section (2.8.1)) so that the output d(n) does not depend upon the current input c(n). The block C has effectively "broken" the closed loop. Hence block C may be processed before blocks A and B. Thereafter the processing order A B is found by tracing the signal flow round the loop starting at block C. The general technique is therefore to process all the retrospective blocks first, in any order ; then the remaining non-retrospective blocks in an order determined from the connections in the block diagram.

7.3 ALGEBRAIC LOOP DETECTION

7.3.1 INTRODUCTION

Since algebraic loops are not allowed in the picture, they must be detected by the graphic compiler. For the purpose of loop detection, a flow graph can be obtained by replacing each block by a node. This allows the application of the topological theory to this problem.

If all the non-retrospective blocks (nodes) are placed in a set, S, then topological sorting can be used [KNUTH 1978]. The nodes in set S contain information as to their interconnections. This results in a partial sorted set since nodes can only be processed after the nodes connected to their inputs are already processed. The basic principle is to pick a first node not preceded by any other nodes. (The first node has no other nodes connected to its input or one with all its input defined.) There must be at least one such node otherwise a loop exists (since this only occurs when all the nodes are in a closed loop). This node is marked and its connection information deleted. The procedure is repeated until the set S is exhausted or when a loop is detected. The topological sorting method detects any existing loops but does not identify them.

To enable identification of the nodes forming a algebraic loop, a method known as the "depth-first search" is used with some modifications [AHO, ULLMAN 1977]. The search is initiated by finding the INITIAL NODE (one with all input defined, see figure (7.4)). Using their connection relation to search for successive nodes (i.e. nodes connected to its ouput), the "tree" is traversed as far as possible. With each new node visted, the existence of this node in the path already "travelled" is checked for. A loop exists if this new node is already in the path and with the path intact, all the nodes forming the loop can be identified. Other possible paths may be formed via traversing to the preceding node to see if another

alternative route is possible. When all the nodes are "visited", all possible paths are exhausted and accounted for.

Figure (7.4) shows the loop detection method in operation. Figure (7.4A) shows the flow graph of the blocks, where node A is the initial node (all inputs defined). Two possible traversing sequences are shown in figure (7.4B) as examples of the operation of "visiting" the nodes.

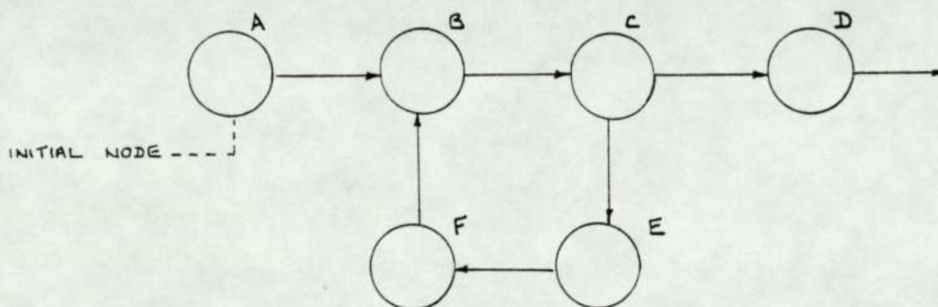


FIGURE 7.4A FLOW GRAPH

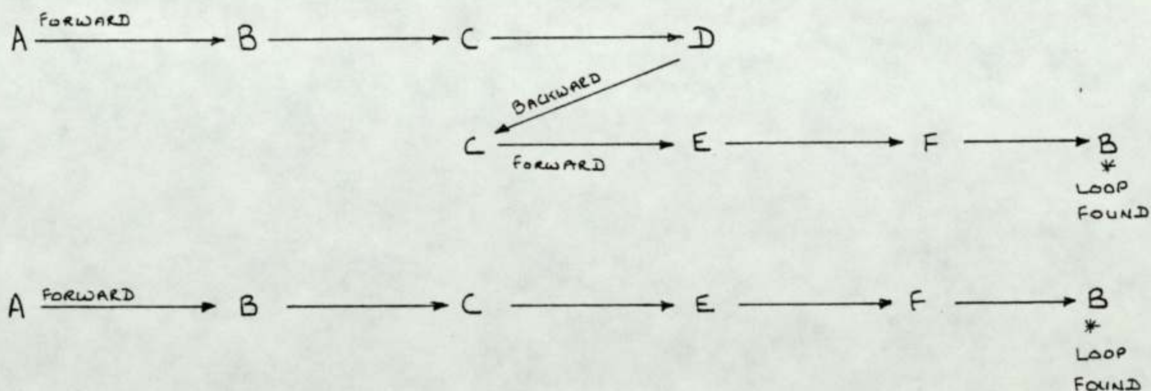


FIGURE 7.4B EXAMPLES OF VISITING THE NODES

FIGURE 7.4 LOOP DETECTION SCHEME

7.3.2 IMPLEMENTATION OF THE LOOP DETECTION SCHEME

Loop detection applies only to all the non-retrospective blocks present in the picture. Of the relationship between the blocks, these can be classified under :-

(A) "Isolated" block i.e. the block is not connected to any other blocks at the input or output ends. Isolated blocks can be ignored since closed loop will never exist for them.

(B) Block having only input connection i.e. the output is not connected to other blocks.

(C) Block having both inp_t and output connections.

The general implementation is as follows : the blocks present in the picture are maintained in a linked list BLKLIST and their interconnection information in another linked list LINKLIST. The structure of both the lists are shown as follows :

```
BLKLIST
+-----------------------------------+
+ (BLINK)   (BLKNO)   (BLKVISITED) +
+-----------------------------------+


LINKLIST
+---------------------------------------+
+ (LLINK) (FROMBLK) (TOBLK) (PATHLINK) +
+---------------------------------------+
```

LEGEND

```
(BLINK     )   link to next block record
(LLINK     )   link to next connection entry record
(BLKNO     )   identification block number
(BLKVISITED)   flag to set when block is vistied
(FROMBLK   )   block where output is connected to (TOBLK)
(TOBLK     )   block where input is connected to (FROMBLK)
(PATHLINK  )   link to records of path "travelled"
```

A block is now selected from the BLKLIST, the first on that list that is not marked visited. If this is an isolated block then, this block is marked visited i.e. the blkvisited flag is set. If this is type (B), then its connection can be deleted from the list LINKLIST and the block marked visited. This is possible since a block with no output connection cannot be part of a closed loop.

For a block of type (C), with both input and output connections, the LINKLIST is used to proceed "forward" to the next block (the next block is that which connected to the output of the block). Each forward block is tested to check if it is already present in the current path (indicated by the PATHLINK entry). If it is not present, then the connection is inserted into the path, via updating the PATHLINK. This block is then marked as visited, the

connection record deleted from LINKLIST, and the procedure repeated going to the next forward block. A loop is detected if the same block appears more than once in a current path and the loop detection algorithm may be terminated. If no more forward block can be found, then "backtracking" will accounts for all other possible paths. Backtracking involves the locating of the last but one block in the current path (blockback), deleting the last block from the path and using the block (blockback) as the focal point to search for another route. This is to ensure that blocks with output connected to more than one input terminals are thoroughly searched for alternative routes of the signal flow. So the path is created going forwarding and then deleted when backtracking if no loop is detected. The complete loop detection scheme is ended when all the block in BLKLIST are marked as visited or when a loop is detected and reported to the operator.

CHAPTER 8

COMPILATION ACTIVITIES - SEQUENCING

8.1 PROCESSING ORDER AMONGST CONTROL SCHEME BLOCKS

This section deals with the order of execution amongst the various block types provided for control algorithms in DDC. The processing sequence during execution is indicated in the figure (8.1). At the start of the processing (computation) cycle, all the input interface blocks are processed first. Ordering between the input interface blocks is arbitrary and the processing order is determined by the sequence the interfaces are linked in the list. Execution of all the input interface blocks first ensure that the correct, up-to-date input values are presented to the rest of the picture scheme.

Now all the outputs of the retrospective blocks can be computed as they are totally independent of each other. Ordering of the blocks in this group is of no significance. The order used is the order in which the blocks are linked in the list.

The non-retrospective blocks can now be processed in the order determined by the sequencing algorithm via tracing the signal flow. At this point all the blocks except for the output interfaces have been evaluated for this cycle. Now all the input queues may be updated. These input queues belong to blocks (retrospective or non-retrospective) that

require their past input values for computation.

To finish off the processing cycle, all the output interfaces can be processed (in the order that they are collected) to present the results of this computation cycle to the process environment.

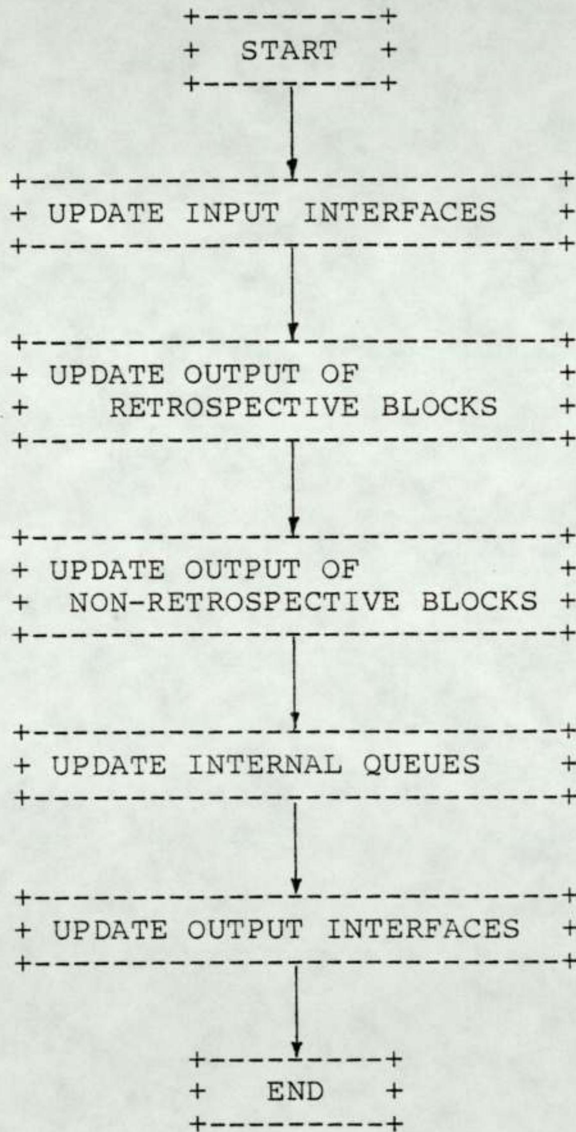


FIGURE 8.1 PROCESSING ORDER AMONGST CONTROL SCHEME BLOCKS

8.2 SEQUENCING FUNCTIONAL BLOCK

8.2.1 INTRODUCTION

Since the retrospective blocks as implemented do not use current input values in computing the current output, there is no need to arrange them in any particular sequence. Input interface and output interface blocks too need no sequencing at all. This leave only the non-retrospective blocks to be sequenced into a proper, correct processing order to ensure correct computation results.

After the expansion of the macro blocks and the subpictures, a single level of only simple blocks with all the valid interconnections results. Firstly all the blocks are collected into their respective classifications, namely the input and output interfaces, retrospective and non-retrospective. This collection of the retrospective blocks will determine the order in which the individual block is processed, the first one being on the top of the collection. Input interface and output interface blocks are dealt in the same way.

With other blocks "removed", the remaining non-retrospective blocks are usually in small groups of interconnected blocks. Replacing the block by node, the topological representation is shown typically in figure (8.2). A node cannot be evaluated until all of its input values have been evaluated. It is therefore necessary to

derive the appropriate processing order for the nodes. This is always possible since closed loops are not permitted.

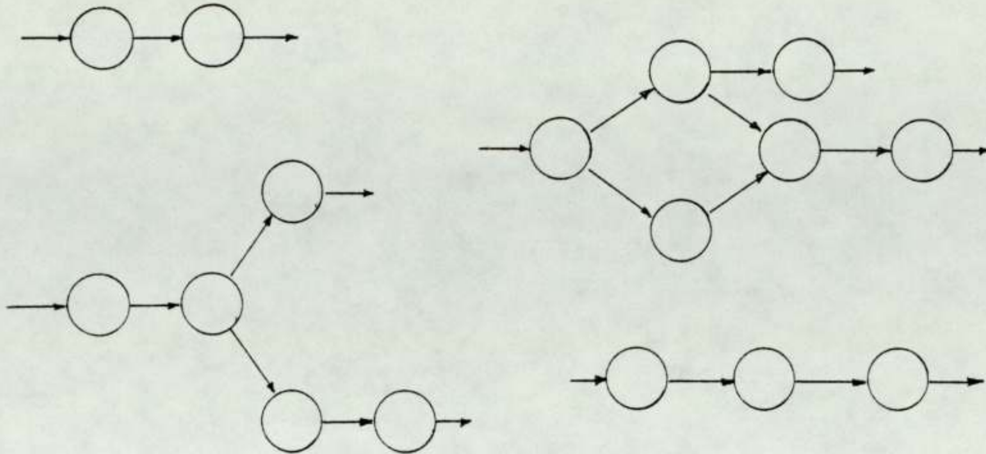


FIGURE 8.2 GROUP OF INTERCONNECTED NODES

8.2.2 SEQUENCING METHOD IMPLEMENTATION

To obtain an easy-to-prove algorithm for the sequencing, all independent, possible signal paths in the picture (only for the non-retrospective blocks) are to be found.

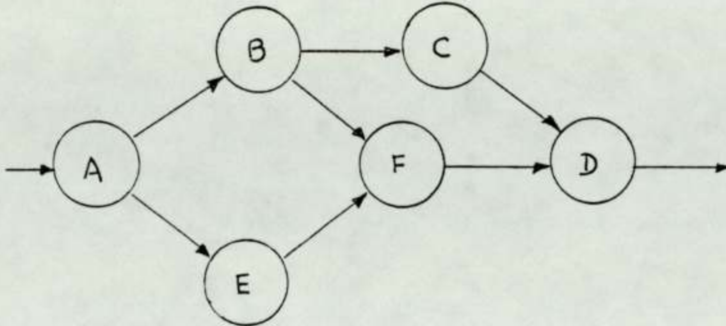
The approach to determine the independent paths is very similar to the "depth-first search" in topology [AHO, ULLMAN 1977]. The search is initiated by finding the INITIAL NODE, in this case the node with all its inputs already defined. This is used as the starting node to search for the next linked node (node which is connected to it) and traversing as far as allowed. This gives the first path. Nodes "visited" are marked by setting an appropriate flag. Other paths may be formed by traversing to the preceding

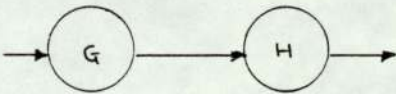
node to check if an alternative signal route is possible. Else another initial node may be found and used as the starting node. When all the nodes have been "visited" then all the possible signal paths are found.

Once all the computation (signal) paths are found, the nodes can now be sequenced fairly easily. Firstly all the starting nodes of all the paths are copied to the output set of nodes (nodes already sequenced). This is possible since the starting nodes are independent in their processing order. The next node is selected on the basis that *all its preceding nodes in all the paths which the node is present have been evaluated*. The nodes sequenced can be marked by negating the node number (the block identification number). Sequencing is completed when all the nodes in all the path are used.

The figure (8.3) shows an example of the operation of the sequencing method. All possible computation paths are found (figure 8.3B). Picking up the starting nodes gives part of the sequence as A G I . The next node B can be evaluated because node A (the preceding node) is already evaluated. Node C is then selected since both A, B are processed. But the node D cannot be evaluated since the preceding nodes in path 2) and 3) (i.e. nodes E and F) are not yet evaluated. The next node is selected, E, since it is next on the available list. Applying the method till completion gives the final sequence as

A G I B C E F D H





(1) A B C D

(2) A B F D

(3) A E F D

(4) G F

(4) I

FIGURE 8.3B SIGNAL FLOW

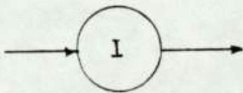


FIGURE 8.3A FLOW GRAPH

FIGURE 8.3 BLOCK SEQUENCING EXAMPLE

8.2.3 DATA STRUCTURE USED IN SEQUENCING

All the blocks in the picture are maintained in a list BLKLIST. The interconnection information is to be found in the list LINKLIST. The BLKLIST and LINKLIST are the same as those used in the LOOP DETECTION (section (7.3.2)). They

are used for the searching and finding of all the possible signal paths. The list PATHLIST maintains all the possible computation (signal) paths, via keeping all the block number of the blocks in a path together.

```
LINKLIST
      ----------------------------
        (LLINK) (FROMBLK) (TOBLK)
      ----------------------------

BLKLIST
      --------------------------------
        (BLINK) (GBLKNO) (BLKVISITED)
      --------------------------------

PATHLIST
      ----------------------------------------------
        (PLINK) (NOOFBLK) (BLKNO1) (BLKNO2) (BLKNO3)
      ----------------------------------------------
```

LEGEND

```
(PLINK  )   pointer to next path record
(NOOFBLK)   number of block within the path
(BLKNO1 )   identification block number of first block
(BLKNO2 )   identification block number of second block
```

FIGURE 8.4 DATA STRUCTURE USED IN SEQUENCING

When all the blocks are sequenced, they are linked together via the ELINK (execution link) entry, in the run time block (RSB) record, in the order in which they are to be executed during run time.

8.2.4 GENERAL COMMENTS ON SEQUENCING

The method implemented is similar to the approaches used in loop optimization and code optimization in compiler design [AHO, ULLMAN 1977]. There is an interesting relation between the flow graphs and the "gotoless" programs

which allows such programs to be checked for the logical flow of data. In modular programming, where the emphasis is mainly on structuring the programs in modules with single entry and single exit, the data flow is particularly highlighted.

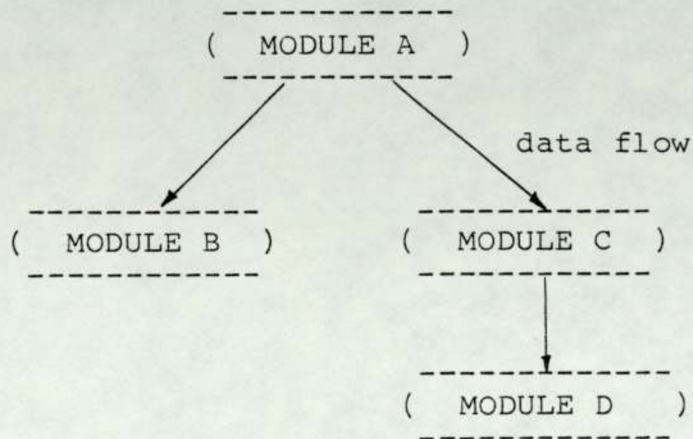


FIGURE 8.5 PROGRAM MODULES AND DATA FLOW

N. Wirth [WIRTH 1976] shows a very good example of the use of topological sorting which operate on set of nodes where partial ordering exists. The basic principle is similar to that used in the sequencing algorithm. His approach placed the main emphasis on the use of the correct data structure (in this case the linked list).

CHAPTER 9

COMPILATION ACTIVITIES - DATA MANIPULATION

9.1 ALLOCATION OF STORAGE FOR RUN TIME BLOCK

9.1.1 INTRODUCTION

There are four distinct classes of blocks, namely (i) retrospective, (ii) non-retrospective, (iii) input interface and (iv) output interface. The retrospective blocks are those whose present outputs can be computed without using the values of the present inputs (e.g. integrator) ; whereas the non-retrospective blocks have their current outputs dependent on their present inputs (e.g. multiplier and summer). The input interface and the output interface blocks can be considered as non-retrospective blocks for the purpose of allocation of memory space.

This allows us to divide all the blocks in the graphical programming language (GPL) program into the retrospective and non-retrospective groups. The fundamental structure of the program is classified by the processing (carrying out computation on) all the retrospective blocks separately from the non-retrospective blocks. Processing of the retrospective blocks can, in principle, be carried out in any arbitrary order. On the other hand, the non-retrospective blocks must be processed in an order determined by the interconnections. To fully maintain the arbitrary processing order, each retrospective block must be

totally independent of any other block as far as the computation of the output(s) is concerned. This is achieved if each block contains all the necessary data values (outputs, present and past inputs, and any internal variables) within its own data region.

An alternative proposed by Linn [LINN 1980] seeks to economise on data storage by using the principle that an input value will be obtained by accessing an output value table in accordance with the block connection pattern. However this approach implies that the retrospective blocks must be sequenced to give the correct result. Taking a simple example, where the output of a retrospective block A is connected to an input of retrospective block B. If block A is processed first, then its output is a(n+1). But since block B requires a(n) (which is over-written by a(n+1)), the processing order must be B A.

9.1.2 DATA STRUCTURE FOR RETROSPECTIVE BLOCK

In this implementation the data structure of the retrospective blocks is defined so as to allow retrospective blocks to be processed without sequencing. The main penalty of this approach is that more storage will be required by each block. Additional storage will duplicate the output value of a block to which an input is connected to. But the extra storage space is small compared with the storage for the rest of the variables of the block. (It will only take up two more storage for a block having two input terminals.)

The data file for each retrospective block has the following structure. The section denoted "current inputs" contains copies of the output values of the blocks to which the inputs are connected. This is updated at the end of each processing cycle when the new output values have been computed for all the blocks in the system.

```
+-----------------+
+    CURRENT      +
+    OUTPUTS      + )
+-----------------+ )
+    CURRENT      + )
+    INPUTS       + ) DATA ORGANISATION
+   ------------  + )
+    PAST         + ) FOR ONE BLOCK.
+    INPUTS       + )
+-----------------+ )
+    INTERNAL     + )
+     VARIABLES   +
+-----------------+
```

FIGURE 9.1 RUN TIME DATA FILE STRUCTURE

This approach has the following advantages :-

(1) the retrospective blocks can be processed in any arbitrary order, no sequencing is required. This inherently allows closed loops of retrospective blocks.

(2) all data required by the block are grouped together so that data accessing routines are simplified.

9.1.3 DATA STRUCTURE FOR NON-RETROSPECTIVE BLOCKS

Interconnected non-retrospective blocks must be processed in a sequence which ensures that no block is processed until all its input values have been updated. This can be avoided by arranging the data structure but the advantages are seen in adopting the same data structure for the variables of every block to conform to that proposed for the retrospective block. This eliminates the necessity of the compiler to differentiate between the two classes of blocks when it comes to the allocation of the data areas. Each block has its own "modular" data file to operate on, simplifying the program structure and reducing the possibility of programming errors.

9.2 INITIALIZATION OF FUNCTIONAL BLOCKS

9.2.1 INTRODUCTION

Since some functional blocks need their past values of the inputs and outputs for computation, memory storages must be allocated for these "variables". These will have to be initialized to some suitable values before the graphical programming language program (consisting of functional blocks) is executed.

Consider the case of a LEAD-LAG functional block (see figure below, using the Laplace operator s),

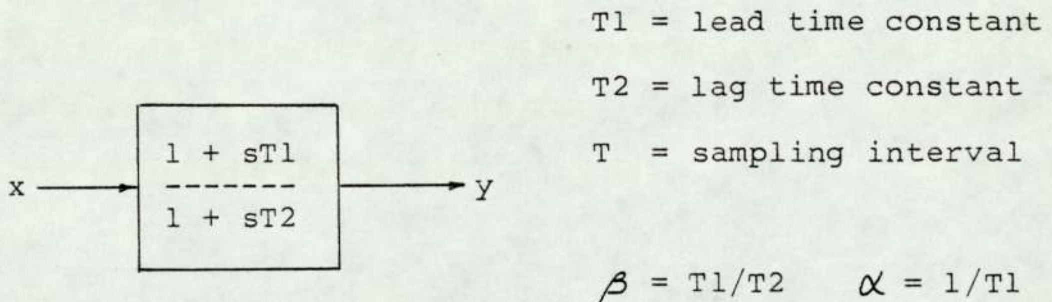


FIGURE 9.2 STRUCTURE OF LEAD/LAG BLOCK

This gives the relationship (using the z-transform method) :

$$y_n = e^{-\alpha T} y_{n-1} + (1 - \beta - e^{-\alpha T}) x_{n-1} + \beta x_n \qquad (9.1)$$

Now at n=0 i.e. initially, from equation (9.1)

$$y_0 = e^{-\alpha T} y_{-1} + (1 - \beta - e^{-\alpha T}) x_{-1} + \beta x_0$$

where y_0 and x_0 are the values of the output and the input respectively at time t = 0,

y_{-1} and x_{-1} the values of the variables at time t = -1.

Obviously, the memory storage values of the variables must be initialized to some suitable values before execution of the block program is possible.

9.2.2 INITIALIZATION CRITERIA

The following sections will discuss the different approaches of initialization of the functional blocks. There are in general three criteria for setting the initial conditions of a system [PRITSKER 1969], [WILSON, PRITSKER 1978A, 1978B] :-

(1) The system is started "empty and idle", that is all the internal variables are set to zero, and the propagation of all the effects of the internal variables is allowed to work through the system before taking any serious measurements. This criterion has the advantage of being easy in implementation.

(2) The system is started at the steady-state mode. This is the best approach but is difficult to implement since the steady-state determination is "tricky" and laborious.

(3) The system is started at the steady-state mean. This is a compromise between approaches (1) and (2), with less propagation effects than approach (1).

From the three above-mentioned approaches, approach (3) is chosen, compromising between approaches (1) and (2).

9.2.3 INPUT-OUTPUT INITIALIZATION

The most elementary approach involves the provision (via the operator or other means) of ALL the initial output and input values of blocks. It is then possible to "trim" the variables to fit the given data. This arbitrary choice of initial values may leave the system in some undetermine state, unless a careful choice is made.

Take the example of a FIRST ORDER LAG :

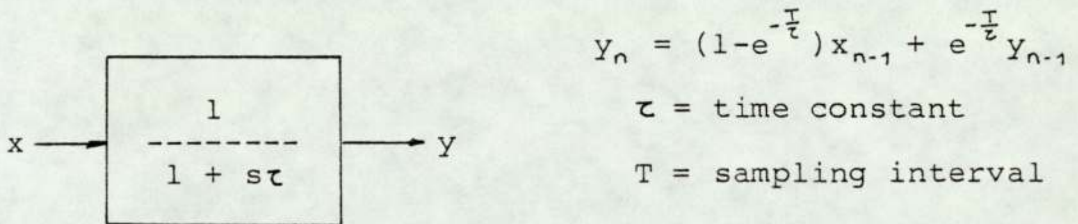


FIGURE 9.3 STRUCTURE OF FIRST ODER LAG BLOCK

Now if the initial values are provided, say

$$\text{output, } y = w \quad ; \quad \text{input, } x = u$$

then

$$x(-1) = u \quad ; \quad y(-1) = \frac{w-(1-A)u}{A}$$

where $A = \exp(-T/\tau)$

giving $\quad y(0) = w = (1-A)\; x(-1) + A\; y(-1)$

The above choice of the variables will fit the given data, since $y(0) = w$ as expected.

Judging from the above example, this method may appear easy to use. But when a sequence of blocks are connected together, then the choice of initial values are not arbitrary since the outputs may affect inputs, unless one is willing to accept the initial "settling down" period for the effects to propagate through. Given a sequence of blocks, it may be possible to initialize the blocks separately, however the system may not give the desired performance.

Since this involves provision of all the data values via the operator (hence a potential source of error), this approach is dropped in favour of the other to be described later.

9.2.4 STEADY-STATE INITIALIZATION

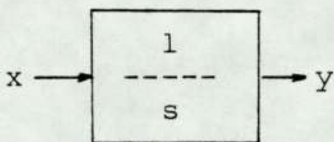
To minimise the transient on start up of the control program, the steady-state relationship between the inputs and outputs is used. In this case, the retrospective blocks are effectively being "replaced" with their steady-state relationship. For the example of the lead/lag block, at steady state, from equation (9.1)

$$\left.\begin{aligned} x(n) &= x(n-1) \\ y(n) &= y(n-1) \end{aligned}\right) \text{ STEADY STATE}$$

giving $\quad y(n) = x(n) = y(n-1) = x(n-1)$

Note that the value of the output remains the same during steady state, if the input remains unchanged. This is expected from the transfer function of the lead/lag block. It is now possible to determine the value of the input or the output, given the other. This also allows checking on the compatibility of the operator-provided initial conditions.

Retrospective blocks which deliver a constant output with a constant input have a well defined steady-state behaviour. However, there are some functional blocks that differ radically from the above behaviour. A good representative is the INTEGRATOR type functional block implemented as follows,



$$y_n = y_{n-1} + T\, x_{n-1}$$

T = sampling interval

INTEGRATOR

Note that the output will be varying with each sampling period except when the input, x, is zero in value. That is

$$y(n) = y(n-1) + T\, x(n-1)$$

$$y(n) \neq y(n-1) \quad \text{at steady state.}$$

These blocks belong to a class, the "<u>INTEGRATIVE</u>", will require more attention and care during initialization. In

the case of the INTEGRATOR, two initial values, namely output value [y(-1)] and input value [x(-1)] must be provided externally.

9.2.5 FURTHER CONSIDERATIONS

9.2.5.1 STEADY-STATE INITIALIZATION

Consider the example in figure (9.4),

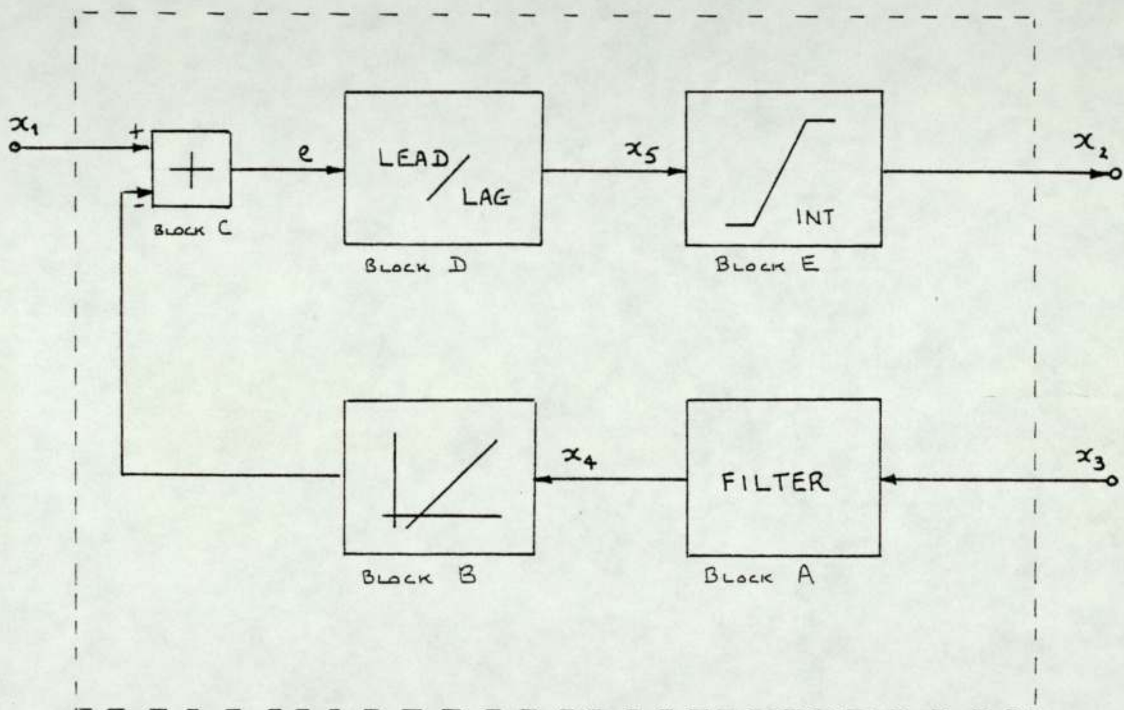
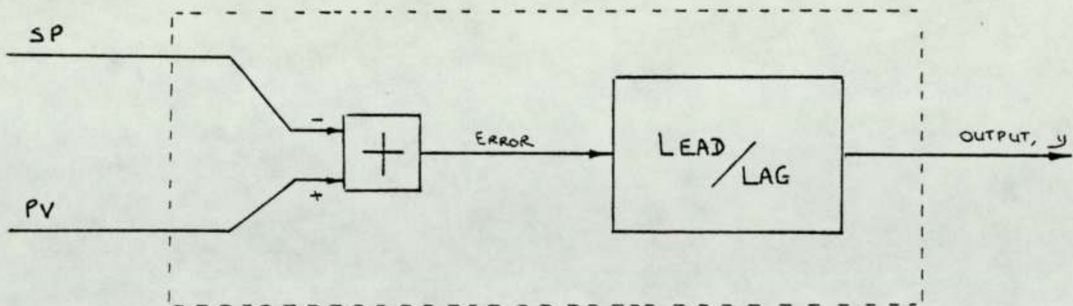


FIGURE 9.4 INITIALIZATION BLOCK DIAGRAM

In the above case, all the variables are marked x1, x2, x3, x4 and x5. If all five variables are provided with initial values, then it is only a trivial matter to select suitable values for the internal variables (using the steady state relationship).

Provision of all the variables may lead to OVER-SPECIFICATION. Consider the case below :



Only two initial values needed to be provided, say SP and PV. Since the variable error (= PV-SP) can be determined and since for the LEAD/LAG, at steady state, the output is equal to the input. Therefore the initial output is equal to the value (PV-SP).

This over-specification is a trivial matter, since the specified value may be overwritten with the calculated required value, and a warning issused to the operator. In most cases, the inputs are used as valid data when over-specification occurs.

Looking at figure (9.4) and considering only the EXTERNAL terminals need to be initialised. The external terminals are those marked with small circles. In this case, only three initial values are required, namely x1, x2 and x3. Using these values and with further manipulation, the rest of the variables (x4 and x5) can be determined, and thus the retrospective blocks initialized. This approach reduces the workload on the operator considerably.

9.2.5.2 EXTERNAL TERMINAL INITIALIZATION

Given the external initial values, the manipulation of these data with the functional blocks is important. One approach is to process all the blocks in the "correct sequence". By the correct sequence, it is meant that each block can be processed if and only if all its inputs are valid (i.e. previously defined). (By previously defined, it is meant that the values are provided via the operator or are output values of already processed blocks.) From the structure of figure (9.4), the correct processing sequence is obviously A, B, C, D, E. The retrospective blocks are replaced with their steady state relationship, making them behave like non-retrospective blocks. From block A, and given x3, x4 and e can be calculated. Variable x5 can then be found and thus all information needed for initialization are found.

The choice of approach for initialization is that of the external terminal initialization approach. That is the operator needs provide only the initial values of the EXTERNAL TERMINALS and the outputs of the integrators.

CHAPTER 10

SIMULATION

This chapter covers the simulation of the mathematical model of a process. Such a model can be analytically derived from physical theorems or obtained by process identification methods [SMITH 1972]. The chapter starts with a brief review of available simulation languages, then considers the crucial integration requirements and concludes with the implementation of some functional blocks.

10.1 REVIEW OF DIGITAL SIMULATION LANGUAGES

The early days of computer simulation were dominated by analog machines. The initial development of digital simulation languages serves two main purposes, namely (1) to provide an alternative tool to check the solutions of the analog machines and (2) as a back-up in case of the analog machine break-down [STRAUSS 1968]. Early languages correspond closely to the use of analog computer, giving rise to a family of "Digital Analog Simulators" or BOSLs (block oriented simulation languages) such as MIDAS, MIMIC and DYSAC [BRENNAN 1968]. Such early languages are relatively specialised programs and thus are rather simple and compact to implement allowing their use on small digital machines. However usually BOSLs are closed-ended with little facility for expansion [GULLAND 1973].

Meanwhile attempts had been made to develop languages that related to the structure of ordinary differential and algebraic equations (i.e. the EOSLs - equation oriented simulation languages). The earliest landmark is the development of the DSL (digital simulation language) by IBM. Interests in this approach leads to the adoption of a "standard" for simulation languages by the SCi Committee on Simulation Software [SCi SOFTWARE COMMITTEE 1967], a U.S. professional body. One important feature of EOSLs is the "macro facility" devised from advanced assemblers. This allows the repetitive use of a submodel which only needs to be defined once.

One of the main drawbacks of many simulation programs has been their lack of close man-machine interaction caused by the use of batch processing rather than interactive processing [REVETT 1973]. Recently more and more simulation languages are designed for interactive processing e.g. DARE-P [LUCAS, WAIT 1975] and BEDSOCS [EIDELSON 1980].

BOSLs use the following principle : the system is represented by a block diagram using blocks from a library of standard blocks. The model specification is entered via a terminal consisting of the block details, the parameters and interconnections between the blocks in alphanumeric form. Examples of such languages include DYSAC [HURLEY, SKILES 1963], KALDAS [DINELEY 1967] and a system by Payne [PAYNE 1974].

The process of manually translating the block diagram into the necessay numerical form for the simulation programs is an error-prone and tedious task. To bypass the manual translation, the simulation language in this implementation consists of the actual block symbols and their interconnections being communicated to the system by using graphics. Main features of the approach include (1) direct correspondence between the blocks and the physical systems and (2) the drawing on the display screen is all the documentation required to diagnose or debug the simulated model. The graphical approach is similar to the graphical programming language approach for the programming of control algorithms (section 2.3). The synthesis and most of the compilation of the pictorial program are identical. The most obvious differences are in the provision of different functional blocks and the allocation of storage locations. These differences will be discussed in section (10.9).

10.2 STATE VARIABLE REPRESENTATION

In the simulation of continuous systems, it is most convenient to represent th_ system dynamics in the *state-variable* form as a set of simultaneous first order differential and algebraic equations. These equations are of the general form :

$$\underline{Y}' = F (\underline{Y}, \underline{U}, t)$$

where $\underline{Y}$ is a vector matrix of the state variables,

$\underline{Y}'$ the derivative vector

$\underline{U}$ the independent input variables

t the continuous time variable

This is the most general form of system description embracing both linear and non-linear systems. To quote Brandin [BRANDIN 1968], "simple algebraic techniques and approximations reduce virtually all systems to a set of simultaneous ordinary differential equations of the first order". The fundamental structure of state-variable form is a set of <u>interconnected</u> <u>integrators</u>. The basic dynamic element is the integrator for which the choice of numerical integration algorithm is of crucial importance.

10.3 INTEGRATOR BLOCK IN SIMULATION

10.3.1 INTRODUCTION

The various numerical procedures for generation of solutions of the first order differential equations are very well discussed in the papers by Benyon [BENYON 1968] and Brandin [BRANDIN 1968]. The theoretical aspects and application of such procedures can be found in various books on numerical analysis such as [GEARS 1971] and [KOPAL 1955]. The classical methods for the numerical solutions of ordinary differential equations can be classified as the following :

(1) The single-step method, of which the most commonly used algorithm is that of the Runge-Kutta 4th order.

(2) The multi-step method or the predictor-corrector approach, one example being the Adams (Moulton) predictor-corrector.

Figure (10.1) and (10.2) show typical examples of both the approaches applied to a single first order differential equation.

```
Y' = F ( Y, U, t )

K1 = F ( Y[n], U[n], t[n] )
Y1 = Y[n] + K1 * H/2 ; t1 = t[n] + H/2

K2 = F ( Y1, U[n], t1 )
Y2 = Y[n] + K2 * H/2 ; t2 = t[n] + H/2

K3 = F ( Y2, U[n], t2 )
Y3 = Y[n] + K3 * H   ; t3 = t[n] + H

K4 = F ( Y3, U[n], t3 )
Y[n+1] = Y[n] + H ( K1 + 2*K2 + 2*K3 + K4 )/6

t[n+1] = t[n] + H
```

FIGURE 10.1 RUNGE KUTTA FOURTH ORDER INTEGRATION RULE

```
Y' = F ( Y, U, t )

Y'[n] = F ( Y[n], U[n], t[n] )

PREDICTED VALUE
Z[n+1] = Y[n] + (H/24)( 55Y'[n] - 59Y'[n-1] + 37Y'[n-2]
                                 - 9Y'[n-3] )

t[n+1] = t[n] + H

Z'[n+1] = F ( Z[n+1], U[n] , t[n+1] )

CORRECTED VALUE
Y[n+1] = Y[n] + (H/24)( 9Z'[n+1] + 19Y'[n] - 5Y'[n-1]
                                  + Y'[n-2] )
```

FIGURE 10.2 ADAM-MOULTON FOURTH ORDER PREDICTOR-CORRECTOR

LEGEND

H - integration step size
U - independent input variable
t - continuous time variable
Y - variable under consideration
Y' - derivative of Y

10.3.2 FEATURES OF SINGLE-STEP AND MULTI-STEP APPROACHES

The general features of the Runge Kutta algorithm are :

(A) It is self-starting. Past values of Y are not required so that the procedure can be started without initialization.

(B) It requires N evaluations of the derivatives for a Nth order method.

(C) The truncation error at each step is proportional to

$$H^{N+1}$$

where H is the integration step size and N the order of the method.

The predictor-corrector approach has the following features :

(A) It is not self starting since the previous values of Y i.e. Y[n-1], Y[n-2], Y[n-3] are required. Usually the first few parts of the solution are calculated by some other method such as a single-step approach.

(B) It may iterate the corrector until the required convergence is reached, although in practice this is rarely done.

(C) It provides an excellent estimation of the truncation error especially if the predictor and the corrector are

of the same order.

(D) Previous information must be retained and maintained so that storage requirements are increased.

The predictor-corrector approach usually requires fewer derivative evaluations per integration step than the single-step approach of the same order. However this does not necessarily imply that the predictor-corrector method is faster than the single-step approach in terms of overall computation speed. The single-step method deduces the rate of change in the variable by exploration of the values of the variable at various locations within the step. In the multi-step approach, the rate of change is deduced by extrapolation from what has been happening in the previous steps. The single-step approach also tends to have greater stability than the multi-step approach, making it possible to use a larger step length. Some variations of the multi-step approach had been found to give good stability as found in the work of Hamming [HAMMING 1959], Milne and Reynolds [MILNE, REYNOLDS 1960] and Gurk [GURK 1955].

The choice between the two numerical solution approaches is subjected to the several factors [KORN, WAIT 1978], [BRANDIN 1968] including :

(1) the nature of the systems equations, in particular the degree of the non-linearity and the range of the time constants involved.

(2) the accuracy required and the stability of the solution.

(3) the processing requirement in terms of computation time and memory space.

A further development which is particularly useful for highly non-linear systems is based on varying the step length to control the error at each stage.

10.3.3 ERROR ESTIMATION

The normal Runge-Kutta (RK) approach does not provide any estimation of the error in the solution but variations of the basic method have been derived to give error estimation. The two approaches are

(1) to use two Runge-Kutta procedures of different order and use the difference in each step to estimate the error.

(2) to use the same procedure, but with two different integration step size and compare the two results.

By the careful choice of intermediate points, some of the intermediate results are common to both the RK formulae and the computation can be minimized. Merson [MERSON 1957] has devised a modified RK method that enables the error estimation to be made easily. The Runge-Kutta-Merson (RKM) fourth order integration algorithm increases the number of

evaluations of the derivative per integration step from 4 to 5 and is shown in figure (10.3). Another development has been the Runge-Kutta-Fehlberg (RKF) variation [FEHLBERG 1969]. Work was also carried out on the RKM method by Chai and Burgin [CHAI 1974], [CHAI, BURGIN 1970]. Once the error at each integration step can be estimated, the adjustment of the integration step size is possible. This can minimize the computation time while keeping the error within bounds.

```
Y' = F ( Y, U, t )

K1 = F ( Y[n], U[n], t[n] )                      ]
Y1 = Y[n] + (H/3)K1        ; t1 = t[n] + H/3     ] 1st stage

K2 = F ( Y1, U[n], t1 )                          ]
Y2 = Y[n] + H(K1+K2)/6     ; t2 = t[n] + H/3     ] 2nd stage

K3 = F ( Y2, U[n], t2 )                          ]
Y3 = Y[n] + H(K1+3*K3)/8 ; t3 = t[n] + H/2       ] 3rd stage

K4 = F ( Y3, U[n], t3 )                                 ] 4th
Y4 = Y[n] + H(K1 - 3*K3 + 4*K4)/2 ; t4 = t[n] + H ] stage

K5 = F ( Y4, U[n], t4 )

Y[n+1] = Yn + H ( K1 + 4*K4 + K5 )/6 ; t[n+1] = t[n] + H

ERROR = H ( 2*K1 - 9*K3 + 8*K4 - K5 )/30

                                  H -- integration step size
```

FIG. 10.3 RUNGE-KUTTA-MERSON FOURTH ORDER INTEGRATION RULE

A careful study of many popular multistep and the Runge-Kutta methods had been carried out by Shampine and co-workers, and they concluded that "the fourth order RKF strategy is often a good choice as a general-purpose integration rule" [SHAMPINE 1976]. An investigation into

the performance of integration routines used in general-purpose digital simulation programs was also carried out by Martens [MARTENS 1969]. The performance was compared with respect to the overall speed, accuracy and convenience of use. He concluded that "for the general simulation of linear and non-linear systems, the variable step-size Runge-Kutta-Merson method proves to be most accurate and most efficient."

The different conclusions in the two studies indicate that no one integration rule is best for all purposes. In this project, the RKM fourth order algorithm with variable integration step size is used. This algorithm is commonly used in simulation languages such as the SLAM package [ICL 1974], BEDSOCS package [EIDELSON 1980] and DARE-P [LUCAS, WAIT 1975].

10.3.4 CONTROL OF INTEGRATION STEP SIZE

The absolute value of the integration error (ABSERR) and the relative magnitude of the error (RELERR) specified by the user can be compared with the actual error estimation. The total acceptable error TOTALERR of each integrator is calculated as

$$TOTALERR = |ABSERR| + |RELERR*INTOP|$$

where INTOP is the output value of the integrator.

The error measure used is

(A) ESTERR > TOTALERR

(B) ESTERR < TOTALERR

(C) ESTERR < TOTALERR/2

where ESTERR is the estimated error for each integrator.

If condition (A) is met for at least *one* of the integrator present the integration step is halved and the integration step is repeated. If (B) is met for *all* the integrators present the next step is taken without any change in the step length. If (C) is met for *all* the integrators present the result is accepted but the step length is doubled for the next step.

10.4 RELATIONS BETWEEN INTEGRATOR AND OTHER BLOCKS

There are two phases in implementation of the forward integration algorithm in each computation cycle. Phase (1) is the *derivative evaluation* where the other blocks are processed to evaluate the derivative input to the integrators. The integrator blocks are not involved in this phase. Phase (2) is the *integration evaluation* where only the integrator blocks are involved. This phase advances the numerical solution to the next time step.

10.5 DISTINCTION BETWEEN H AND Tc

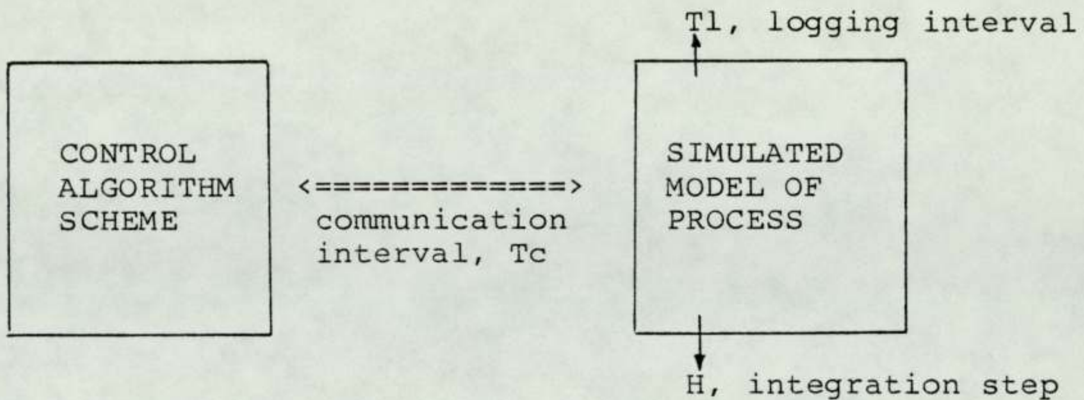


FIGURE 10.4 DISTINCTION BETWEEN H AND Tc

The communication interval Tc is the time interval when the results of the model under simulation are presented to the environment (in this case the control algorithm scheme). In this implementation, the communication interval is identical to the sampling interval, Ts used in the control scheme (section 2.8). If the simulated model itself is under investigation, then the user may wish to know the values of the output in between Tc. This interval can be denoted as the "logging interval", Tl. In this implementation the logging interval Tl is the same as Tc. The changing of Tl to a value different from Tc is not implemented.

The integration step length H is the time interval by which the integration algorithm will advance the numerical solution. In theory, the smaller the H the better is the result of the integration rule. However due to the limited precision in a small digital computer, the above does not apply due to truncation error and round-off error effects. There is an optimum choice of H for each system depending on

the integration rule used, specific computer used and the systems dynamics.

10.6 BLOCK TYPES IN SIMULATION

There are four general types of functional block to be used in simulation :

(A) the integrator. This is the basic dynamic and most important element in the simulation process. The integrator is used to advance the numerical solution of the first order differential equation from one stage in time to another. The choice of the integration algorithm is crucial and involves a compromise to obtain sufficient accuracy without excessive computing time.

(B) the non-dynamic blocks. These blocks have no memory storage or past history of the input or output values. The blocks respond only to the current input values, examples include the summer, multiplier and function generator.

(C) the special dynamic blocks. A block of this class has "memory" i.e. it utilises the past history of the input or output values for the computation of the present output. Examples include the delay and derivative function.

(D) the <u>interface</u> blocks. The input and output interface blocks are for interaction with the environment and are similar to those discussed in section (2.7).

(E) the <u>composite</u> blocks. Such a block is a collection of blocks to be treated as a single entity in graphic representation. During the compilation, the composite block will be expanded down to the simplest form in terms of blocks of the other classes. Examples include the first order lag and second order lag. In effect, the only elements appearing in the run-time system are blocks of type A, B, C and D.

10.6.1 MINIMAL BASIC SET OF BLOCKS

The minimal basic set of functional blocks required for effective simulation of a model is given below :

- integrator
- special dynamic blocks of delay and derivative function
- non-dynamic blocks of summer, multiplier, function generator and junction block.
- interfaces, input and output
- composite blocks of first order lag and second order lag.

10.6.2 COMPOSITE BLOCK IMPLEMENTATION

Composite blocks can be generally be divided into two classes : (A) those without an internal integrator component and (B) those with an integrator as one of the internal constituents. Implementation of class (A) is obvious and straightforward. The implementation of class (B) is carried out to ensure that the only dynamic element is the integrator. This is best illustrated by an example, the first order lag (see figure 10.5).

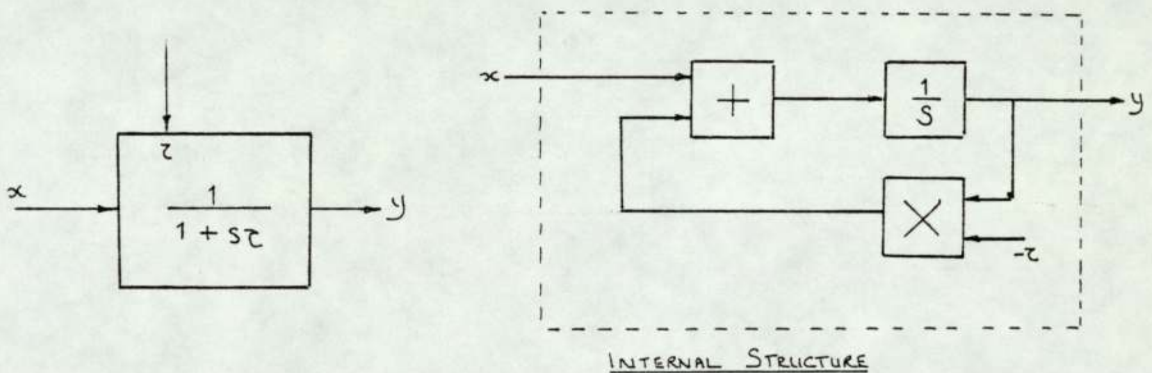
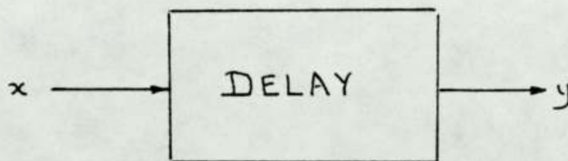


FIGURE 10.5 COMPOSITE BLOCK IMPLEMENTATION

10.7 DELAY BLOCK IN SIMULATION



The delay block function is an approximation to a continuous delay. With the Runge-Kutta-Merson RKM variable step algorithm used for the integrator function, it is not possible to provide delayed output values to correspond with the computation time used. It is also difficult to allocate

memory space for the delayed output, the number of past values varies with the change of the integration step length.

One solution will be to store the data at the end of each integration step. The output of the delay block is kept constant during the integration step interval. For the approximation to be accurate, the integration step should be kept as short as possible.

10.8 DERIVATIVE BLOCK IN SIMULATION

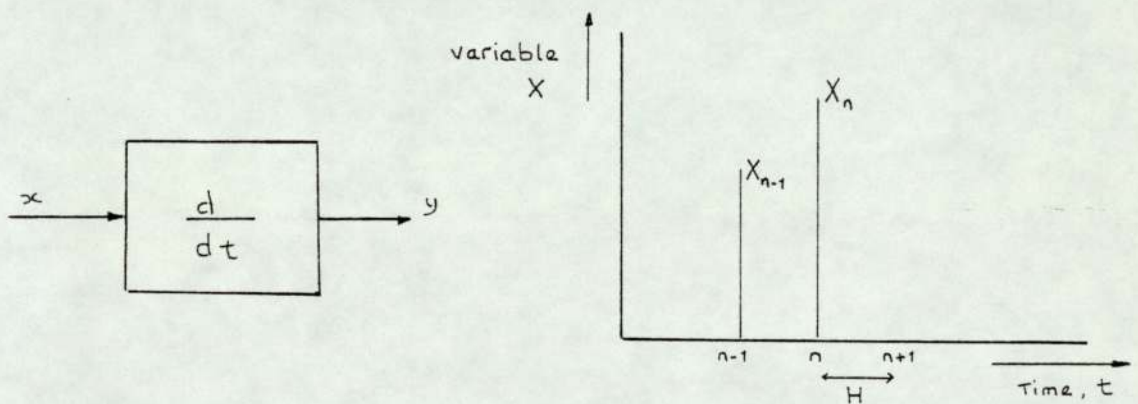


FIGURE 10.6 DERIVATIVE BLOCK & DIFFERENCE EQUATION APPROACH

Two approaches to implement the derivative block are considered, (A) difference equation [BIBBERO 1977]. The derivative of the variable X can be approximated at the time interval t[n] to t[n+1] by

$$\text{derivative} = (X[n] - X[n-1])/(t[n] - t[n-1])$$

Note that this implementation will give a block the output of which varies only at the end of each integration step.

(B) the use of the integration function

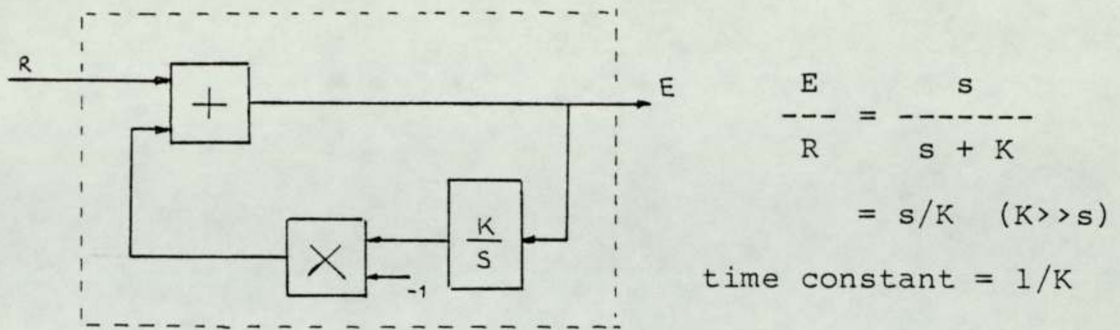


FIGURE 10.7 DERIVATIVE FUNCTION FROM INTEGRATION

In this implementation, the choice of K is important. The larger K is, the better is the approximation. However too large a K may make the time constant of the function so small that it may reduce the range of the integration step size used in the rest of the system. One interesting choice of K is 1/H where H is the integration step length to give the update of the derivative output within each integration interval as required by the RKM algorithm.

In this project, the derivative function will be approximated by the first order difference equation approach for the following reasons :

(1) the integration function approach increases the processing time during execution due to the extra integrator and can have an advance effect on the step length.

(2) the difference approach needs less memory space than the integration approach.

10.9 SIMULATION AND THE GRAPHIC COMPILER

The graphic approach for the simulation of the model of process and the effects on the graphic compiler (chapter 6) are discussed below :

(1) data tranformation of the type-specific data from graphic to run time. The mechanism is identical to that necessary for the compilation of the control program, so no modification is required.

(2) expansion of composite blocks. The basic expansion action is identical.

(3) error checking. In simulation, closed loops of non-dynamic blocks (algebraic loops) are not allowed. So the loop detection scheme need no changing.

(4) sequencing of blocks is further discussed under section (10.9.1).

(5) the allocation of storage for the run time data file is discussed in section (10.9.2).

(6) initialization. In simulation, the special dynamic blocks need to be initialized (section 10.6).

(7) the listings and messages system is identical and changes are not required.

10.9.1 SEQUENCING OF BLOCKS

For the simulation program, the block types that need no sequencing are the integrator (similar to the retrospective blocks used for control algorithms (section 2.7)) and the interfaces. The rest of the blocks must be sequenced to ensure that the computation conforms to the data-flow requirements. The sequencing method is identical to that for the control scheme and the details are given in section (8.2.2). The execution order of the block types is shown in figure (10.8)

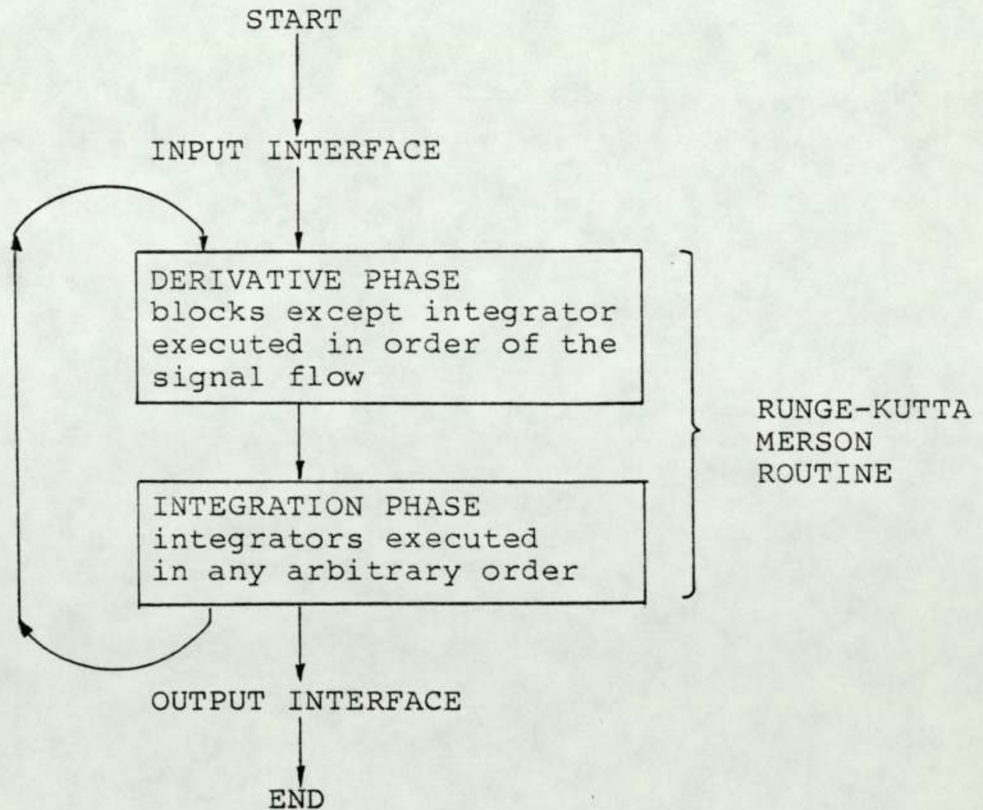
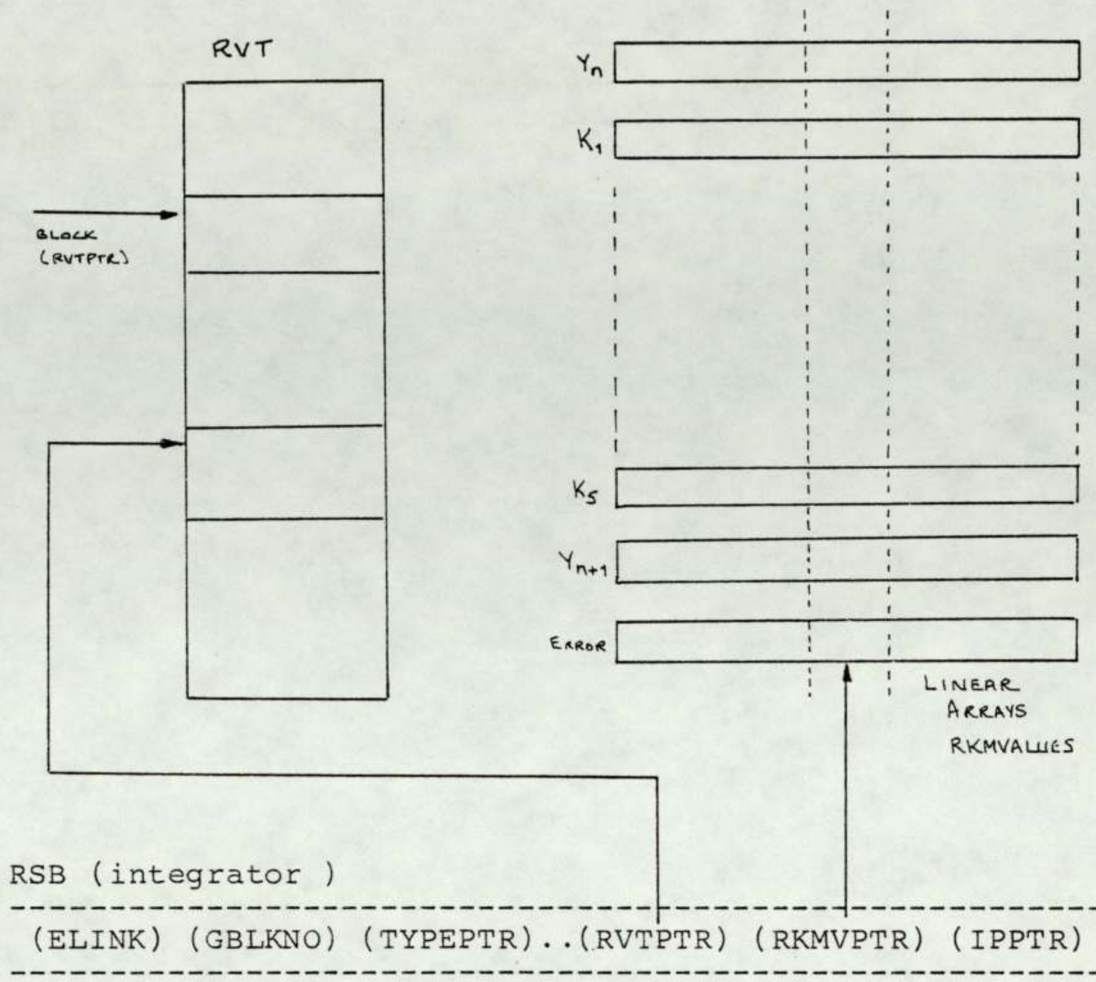


FIGURE 10.8 EXECUTION ORDER AMONGST SIMULATION BLOCK

10.9.2 STORAGE ALLOCATION FOR BLOCKS

As previously stated, the only dynamic block appearing in the program structure after the expansion of composite blocks is the integrator. Efficient computation of the integrator function is therefore important. To achieve this, all the Yn, K1, K2, K3, K4, K5, Yn+1, ERROR (used in the Runge Kutta Merson RKM integration rule) are stored in linear floating-value arrays (figure 10.9). This allows the calculation of all the K's as one set.

The other (non-integrator) blocks are just allocated data storage in the RVT, run time value table, and the data file is identical in structure to that for control algorithm (section 9.1). The linear arrays, RKMVALUES are totally separate from the storage in RVT. These extra storage necessary for the integrator affects the structure of the run time data records slightly. A new pointer (RKMVPTR) is required to indicate to the location in the RKMVALUES, where the values for the RKM routine are stored. Since the MXPTR entry in the RSB (run time simple block) record is always a null entry (section 4.3.2), this entry can be used for the RKMVPTR. Figure (10.9) shows the effects on the RSB record of an integrator.



<u>LEGEND</u>

RSB	run time simple block record.
(RKMVPTR)	pointer to location of linear arrays where values for Runge Kutta Merson routine are stored.
(RVTPTR)	pointer to RVT (run time value table), data file for the block.

The rest of the entries are identical to figure (4.13).

FIGURE 10.9 DATA ALLOCATION FOR INTEGRATOR

CHAPTER 11

TESTING OF CONTROL ALGORITHMS

11.1 INTRODUCTION

This chapter deals with the interaction of the control algorithm program with the simulated model of the process (figure 11.1). The control programs can be thoroughly tested and evaluated to ensure satisfactory performance.

The two following publications gives some indications as to the importance and difficulties involved in the testing to obtain reliable control software : [KRATZER 1979], [D'HUSLTER 1979]. Kratzer described a system where the development of process control software is divided into three phases. The first phase covers the design of control algorithms and off-line simulation. Phase 2 comprises evaluating the control program on a process computer in a real-time environment. The final phase covers refined simulation on a dual computer system where one computer simulates the plant while the other is controlling the plant through realistic interfaces.

In this implementation, a simpler approach is adopted. Both the simulation of the model and the control algorithm are to be carried out on one computer. Realistic interfaces can be obtained through the use of appropriate interface functional blocks.

The control program from the GPL (graphical programming language) system is totally self-contained since it only communicates through the input and output interfaces. These interfaces are implemented to communicate through reserved memory locations and are not interested in the actual process interface equipment used (figure 11.2 and section 2.7.1). These same interfaces are used in the simulation of the model of the process. This is essential to ensure the independence of the control program. The control program does not and need not know that it is only trying to control a model of the process. This is important to avoid changes when the control program is transferred down to the dedicated processor controller for the actual applications (section 2.6).

One of the most important parameter specification is the sampling interval, Ts, used in the control program. Ts is dependent on the actual process time constants. Variations of Ts can be carried out to check the effect on the control performance. Ts is also the communication interval between the two subsystems of process model and control scheme.

11.2 INTERACTION SUPERVISOR

Both the execution of the control program and the model of the process are to be carried out on the same host computer of the GPL system. For orderly interaction, an interaction supervisor (IS) is designed to "sit" upon the

two programs (figure 11.1). The basic task of the IS is to transfer the control of the execution hardware from the control program to the process simulation program.

The operation of IS is as follows for a computation cycle,

(1) transfer the values of the process model output to the input interfaces of the control program.

(2) start execution of the control program.

(3) transfer the values of the output of the control program to the process model input.

(4) execute the program to simulate the process model.

The interaction communciation is handled by the IS in the following manner : all the terminals of interest of the input and output interface blocks are to be "tagged" by a label (see figure 11.1). The two terminals with the same tag label are to be related by value. Figure (11.3) shows the data structures used by the IS for the inter-communication.

The interaction supervisor may be developed to any level of sophistication required. The minimal functions of IS include :

(1) the controlling of the start and ending of the whole process of interaction.

(3) provides for interactive parameter display and

modifications such as the value of sampling interval, Ts.

(4) the specification of parameters for logging and to perform the logging of the values. (Figure 11.3 shows the basic data structure required for this purpose.)

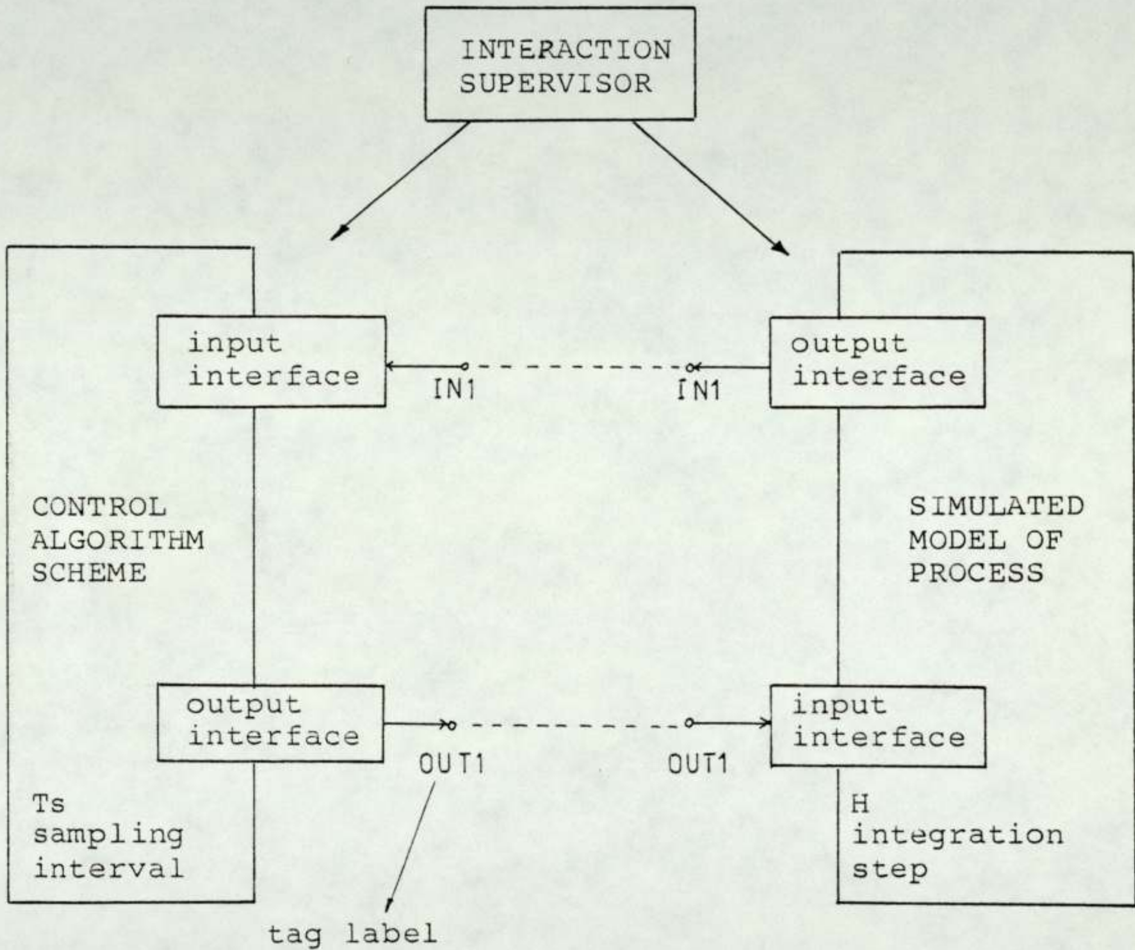


FIGURE 11.1 INTERACTION BETWEEN CONTROL SCHEME AND MODEL

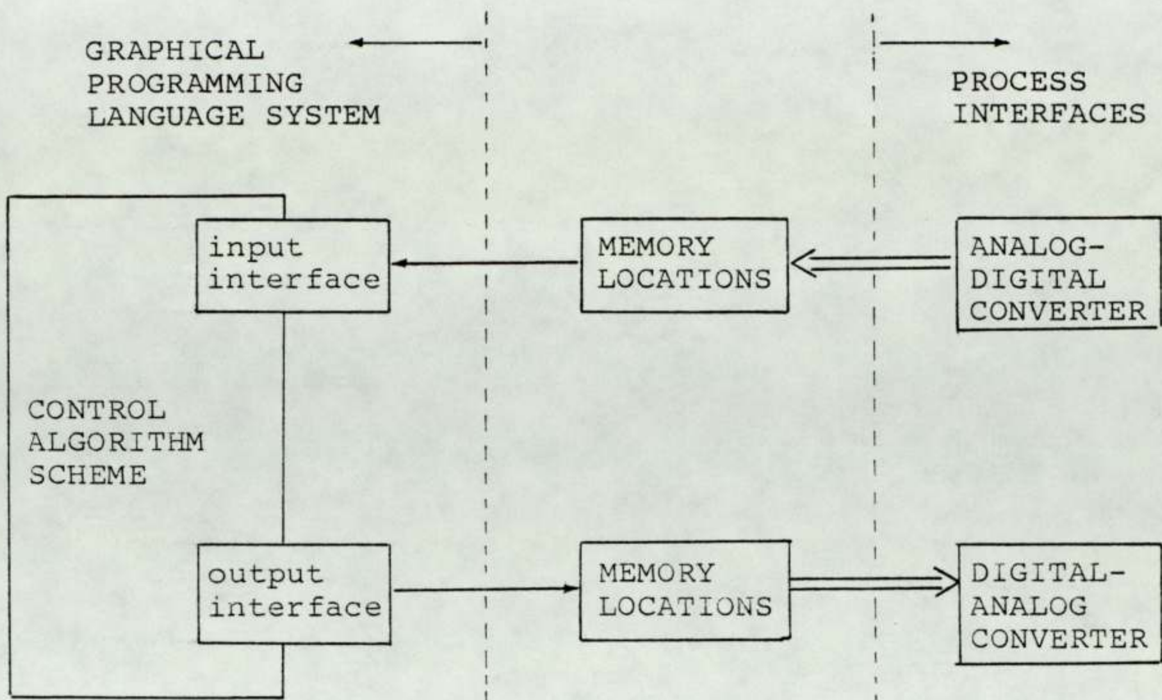


FIGURE 11.2 INTERFACING TO ACTUAL PROCESS

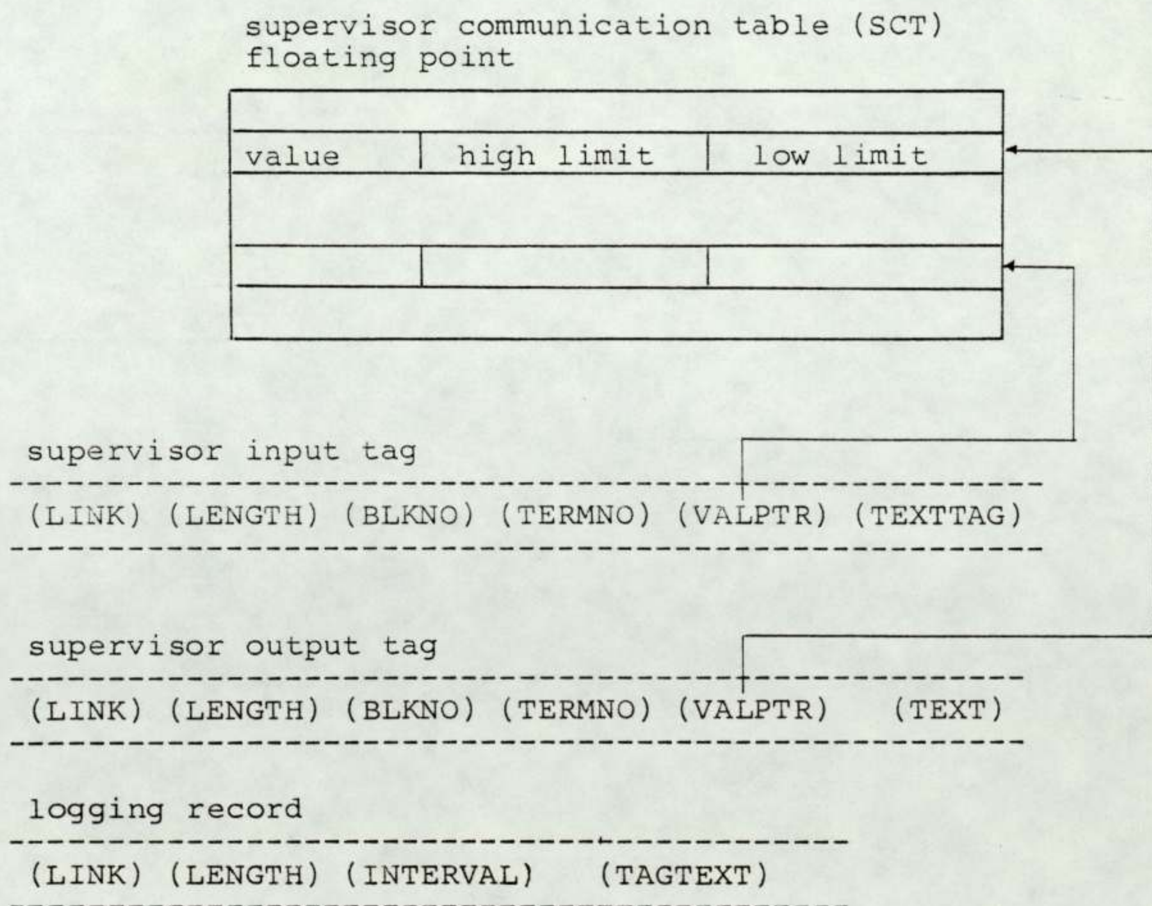


FIGURE 11.3 DATA STRUCTURE FOR INTERACTION COMMUNICATION

CHAPTER 12

CONCLUSION AND RESULTS

12.1 PROGRAMMING IMPLEMENTATION OF SYSTEM

This section gives a general description of the development and implementation of the graphical programming language system. The program for the basic requirements for graphic editor has been completed and now occupies about 24K (16 bits) words. This however is not a good indication of the memory space required by the editor since that depends upon the level of sophistication required and the use of overlays to minimise the space requirement. Besides the essential editing routines, facilities are provided to initialize the graphic data structure records via reading in data records from a file on the storage disk. The format of the file is machine-independent and presents an easy means of access to the user.

Most of the essentials of the compiler are completed, needing about 30K words of storage. Again the required sophistication level can affect the size and overlays can used. Only a limited number of the types of functional blocks (for the control algorithm and simulation) are provided at present. It would be relatively easy to expand the range of functional types. Appendix B shows the development of a graphical block with the corresponding algorithm procedure.

A basic minimal simple interaction supverisor has been built to test the working and co-operation between the two subsystems (control algorithm and model of process).

12.2 CONCLUSION

This research has focussed on a software engineering method offering a system which supports the designing and testing of process control algorithms using the facilities of computer graphics. The system offers an easy to use and fully documented programming facility based on drawings. By treating each software module block as a "black box", the top-down structured programming methodology is strongly encouraged and promoted.

The application of the block diagram structure for process control is analysed. The frequently used features required for process control have been identified and provided for in terms of the appropriate standard functional blocks. Each graphical block corresponds to pre-defined software module performing a basic function. This approach eliminates the need for knowledge of formal programming, since the "programming" (the synthesis phase) is carried out by connecting the required functional blocks together using graphics. A facility to define composite blocks as subsections of a block diagram had been included. This allows segmentation of a block diagram or the repeated use of a combination of elementary blocks as a "macro" block.

Once the machine-independent graphical representation is completed, it is compiled to give a program structure table. The machine-independent program structure table is a compact numerical representation of the block diagrams. This compilation includes the following functions : macro expansion to give a level of basic simple blocks, error-checking to ensure logical flow of data signals, sequencing to determine the necessary order of execution of the blocks and allocation of the storage area necessary for each block. A program generator phase combines code segments relating to the functional routines of the blocks with the program structure table to give the final control program code appropriate to the target processor.

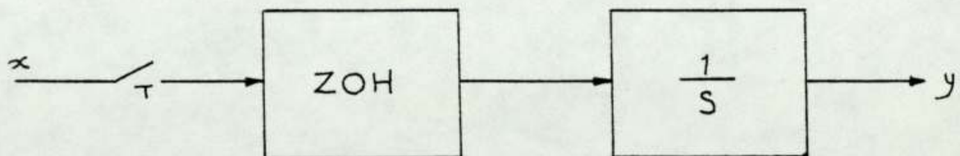
The same graphics based technique has been used to develop a process simulator. This allows the investigation and evaluation of control algorithms through interaction with the simulated model of the process. The control algorithm and process model are run together in one host computer. This evaluation is essential to ensure reliable and satisfactory control schemes before committing them to dedicated process control hardware.

APPENDIX A

Transfer function of blocks using z-transform method

For digital computation, the analog actions such as integration must be transformed into their equivalent discrete forms appropriate to a sampled-data systems. There are various approaches to the tranformation operations [RAGAZZINI 1958]. One approach is via the approximation of the sample and hold operation using a ZOH (zero order hold). The ZOH is important from a practical view since it is simple in nature and is readily implemented. To quote Smith, "In by far the majority of the process control applications, the zero-order hold is used" [SMITH 1972]. So the z-transform method and a ZOH are used to implement some of the blocks provded for the DDC, some of the blocks are discussed below :

(a) INTEGRATOR



$$\frac{y}{x}(s) = \frac{1 - e^{-sT}}{s^2}$$

$$\frac{y}{x}(z) = \frac{Tz^{-1}}{1 - z^{-1}}$$

z = backwards shift operator

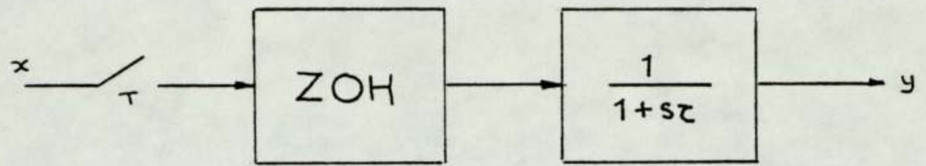
s = Laplace operator

T = sampling interval

$$\underline{y_{n+1} = y_n + Tx_n}$$

This is a retrospective implementation.

(b) FIRST ORDER LAG



$$\frac{y}{x}(s) = \frac{1-e^{-sT}}{s} \frac{1}{1+s\tau}$$

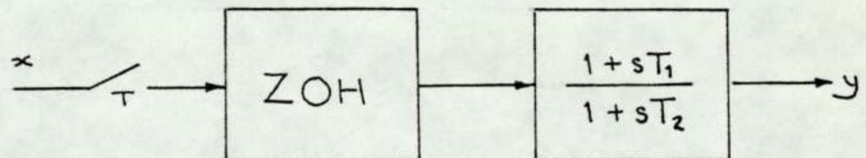
τ = time constant

$$\frac{y}{x}(z) = \frac{1-e^{-\frac{T}{\tau}}}{z-e^{-\frac{T}{\tau}}}$$

$$\underline{y_{n+1} = e^{-\frac{T}{\tau}} y_n + (1-e^{-\frac{T}{\tau}}) x_n}$$

retrospective implementation

(b) LEAD/LAG FUNCTION



$$\frac{y}{x}(s) = \frac{1-e^{-sT}}{s} \frac{1+sT_1}{1+sT_2}$$

T1 = lead time constant

T2 = lag time constant

$$\frac{y}{x}(z) = \frac{\beta z-\gamma}{z-e^{-\alpha T}}$$

$$\beta = \frac{T_1}{T_2} \quad \alpha = \frac{1}{T_2}$$

$$\gamma = (1-\beta-e^{-\alpha T})$$

$$\underline{y_{n+1} = e^{-\alpha T} y_n + \gamma x_n + \beta x_{n+1}}$$

non-retrospective implementation

APPENDIX B

This section shows the development of the graphical details of the functional block symbol. The example of the integrator illustrates the development. Figures (B.1) and (B.2) show the graphical details while figure (B.3) shows the algorithm routine procedure.

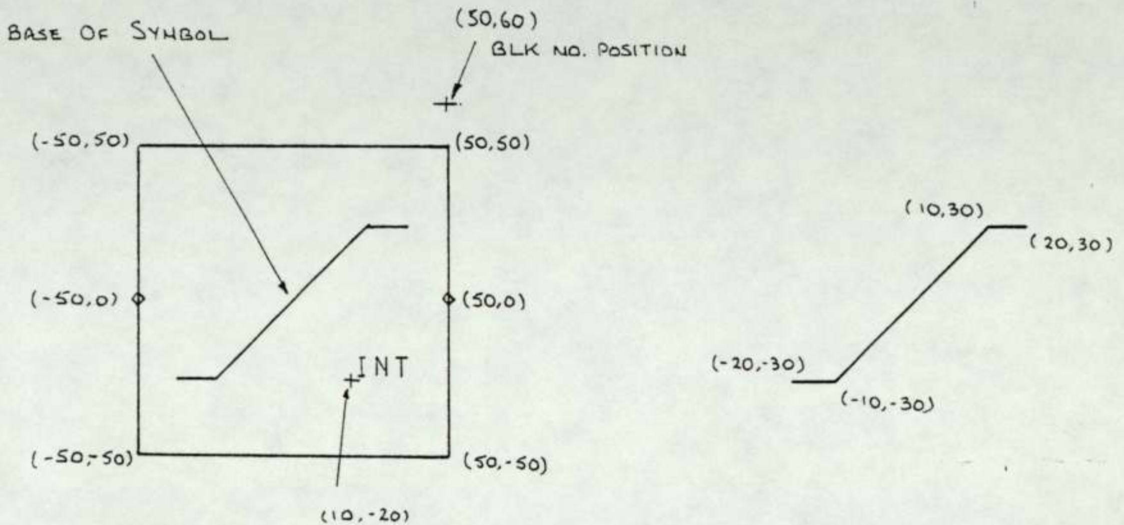


FIGURE B.1 GRAPHICAL DETAILS OF INTEGRATOR BLOCK

GRAPHIC INFORMATION

typeno = 31 classno = 0 mxptr = -1 (invalid)

nip = 1 nop = 1

TERMINAL POSITION (-50,0) (50,0)

BLOCK NUMBER POSITION (50,60)

BLOCK OUTLINE (-50+2000,-50) (-50,50) (50,50)

(50,-50) (-50,-50)

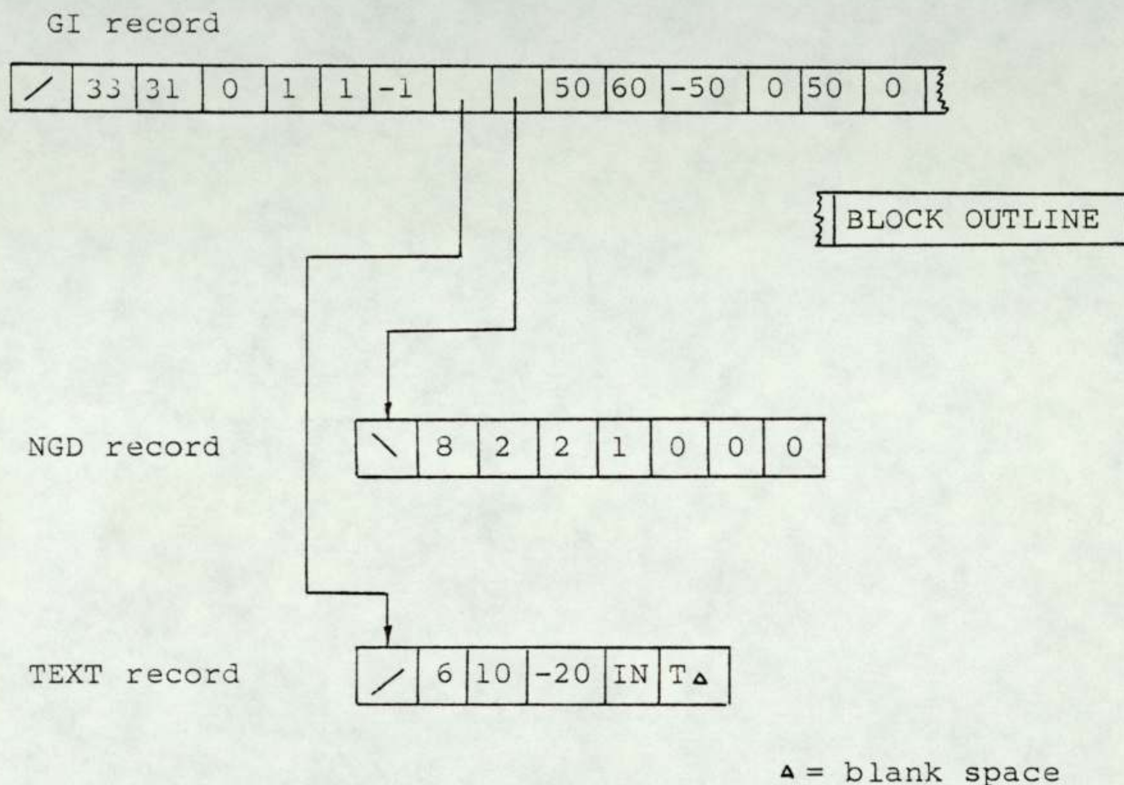
(-20+2000,-30) (-10,-30) (10,30) (20,30)

NON-GRAPHICS INFORMATION

NIV = 2 GF = 2 ROF = 1 LIF = 0 LOF = 0 CIF = 0

BLOCK TEXT

TEXT =INT   text position (10,-20)



The actual elements of each records are discussed in section (4.1).

FIGURE B.2 GRAPHIC DATA RECORDS FOR INTEGRATOR BLOCK

The CORAL66 procedure for the integrator in figure (B.3) should be read in conjunction with figure (6.3).

```
'CORAL'
'PROGRAM' INTEGRATOR
'COMMENT' CORAL66 procedure for integrator block ;
'DEFINE' VI "'VALUE''INTEGER'" ;
'EXTERNAL'('PROCEDURE' INTEGRAL   (VI) ;
           'PROCEDURE' UPINTEGRAL (VI) ;  ) ;
'EXTERNAL'('FLOATING' TSAMPLE  ;(sampling interval)
           'FLOATING''ARRAY' RVT [1:5000] ;
           'INTEGER''ARRAY' RDS [1:5000]  ;
                    (run time data structure array)
            ) ;
'DEFINE' RSBRVTPTR (PTR)      "RDS [PTR+8] " ;
'DEFINE' RSB NO BEFORE INPUT "10"            ;

'SEGMENT' INTEGRATOR
'BEGIN'
'INTEGER' BASE,INPTR,IPPTR ;

'COMMENT' Run time Value Table data file structure ;
'DEFINE' OUTPUTNOW  "RVT [BASE  ]" ; (current o/p)
'DEFINE' INPUTNOW   "RVT [BASE+1]" ; (current i/p)
'DEFINE' INPUTPAST  "RVT [BASE+2]" ; (past i/p   )
'DEFINE' OUTPUTPAST "RVT [BASE+3]" ; (past o/p   )

'COMMENT'-----------------------------------------
          INTEGRATOR --  RETROSPECTIVE
          outnow   innow   inpast  outpast
            y(n)     x(n)   x(n-1)  y(n-1)

          algorithm  y(n) = y(n-1) + T*x(n-1)
          BLKBASE - pointer to the start of the
                 run time simple block (RSB) record.
          ----------------------------------------- ;

'PROCEDURE' INTEGRAL  ('VALUE''INTEGER' BLKBASE) ;
'BEGIN'
       BASE := RSBRVTPTR (BLKBASE) ;
       OUTPUTNOW := OUTPUTPAST +
       INPUTPAST*TSAMPLE ;
'END' ;

'COMMENT' -- update the input queue --             ;
'PROCEDURE' UPINTEGRAL ('VALUE''INTEGER' BLKBASE ) ;
'BEGIN'
       BASE := RSBRVTPTR (BLKBASE)      ;
       INPUTPAST  := INPUTNOW  ; (store previous i/p)
       OUTPUTPAST := OUTPUTNOW ; (store previous o/p)
       INPTR      := BLKBASE + RSB NO BEFORE INPUT ;
       IPPTR      := RDS [ INPTR ] ; (input pointer)
       INPUTNOW   := RVT [ IPPTR ] ; (current i/p)
'END' ;

'END'
'FINISH'
```

FIGURE B.3 CORAL 66 PROCEDURE FOR INTEGRATOR

ABBREVIATIONS USED IN REFERENCES

ACM	Association of Computing Machinery
AFIPS	American Federation of Information-Processing Societies
FJCC	Fall Joint Computer Conference
HMSO	Her Majesty Stationery Office (UK)
IECI	IEEE Transactions on Industrial Electronics and Control Instrumentation
IEE	Institute of Electrical Engineers (UK)
IEEE	Institute of Electrical & Electronic Engineers (NY)
IFAC	International Federation of Automatic Control
IFIP	International Federation of Information-Processing
IMACS	International Association for Mathematics & Computer in Simulation
SJCC	Spring Joint Computer Conference
SIAM	Society for Industrial & Applied Mathematics
UMRCC	University of Manchester Regional Computer Centre (UK)
UKAC	United Kingdom Automation Council

LIST OF REFERENCES

ABRAMS M.D. (1971)
'Data structure for computer graphics'
Proc. symposium on "DATA STRUCTURE IN PROGRAMMING LANGUAGES", SIGPLAN Notice, Vol 6 No 2, Feb 1971, pp 268-286

ACM (1979)
'Status report of the graphics standard committees of ACM/SIGGRAPH'
Computer Graphics, Vol 13 No 3, 1979

AHO A.V., ULLMAN J.D. (1977)
'Principles of compiler design'
Addison-Wesley, 1977

ASTROM K.J. (1980)
'Design principles of self-tuning regulator'
Proc. of an International Symp. on "METHODS & APPLICATIONS IN ADAPTIVE CONTROL", Bochum 1980, (edited by Unbehaugen), pp 1-20, Springer-Verlag

AUSLANDER D.M., TAKAHSHI Y., TOMIZUKA M. (1978)
'Direct digital process control : practice and algorithms for microprocessor application'
Proc. of IEEE, Feb 1978, Vol 68 No 2, pp 199-208

BARNES J.C.P. (1975)
'The use of RTL/2 for multitasking'
Minicomputer Forum 1975, Conf. Proc., On-Line Conf. Ltd (UK) 1975, pp 157-166

BATES D.G. (1968)
'PROSPRO/1800'
IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol IECI-15 No 2, Dec 1968, pp 70-75

BENYON P.R. (1968)
'A review of numerical methods for digital simulation'
Simulation, Nov 1968, Vol 11 No 5, pp 219-238

BERGERON R.D., BONO. P.R., FOLEY J.D. (1978)
'Graphics programming using the CORE system'
Computing Surveys, Vol 10 No 4, Dec 1978, pp 389-442

BIBBERO R.J. (1977)
'Microprocessors in instruments and control'
John Wiley & Sons Inc., 1977
pp 61-63 cascade control
pp 155-176 PID and other algorithms

BRANDIN D.H. (1968)
'Mathematics of continuous system simulations'
Proc. AFIPS 1968 FJCC, Vol 33, pp 345-352

BRENNAN R.D., LINEBURGER R.N. (1968)
'A survey of digital simulation : digital analog simulator programs'
in "SIMULATION : THE DYNAMIC MODELLING OF IDEAS & SYSTEMS WITH GRAPHICS", (edited by J.McLEOD), PP 244-255, McGraw Hill, 1968

BRISTOL (1975)
'Bristol UCS 300 - the process controller'
(Technical Bulletin TM280A), American Chain and Cable Co. Inc. USA

BRISTOL E.H. (1977)
'Designing and programming control algorithms for DDC systems'
Control Engineering, Jan 1977, Vol 24 No 1, pp 24-28

CALCOMP (1974)
'CALCOMP Manual'
University of Manchester Regional Computer Centre (UMRCC), 1974

CARLSON E.D (1978)
'Graphic terminal requirements for the 1970's '
Computer, Aug 1978, pp 37-45

CAZDOW J.A., MARTENS H.A. (1970)
'Discrete-time and computer control system'
Prentice-Hall Inc, 1970

CHAI A.S., BURGIN G.H. (1970)
'Comment on Runge-Kutta-Merson algorithm'
Simulation, Aug 1970, Vol 15 No 2, pp 89-89

CHIA A.S. (1974)
'Modified Merson's integration algorithm which saves two evaluations at each step'
Simulation, March 1974, Vol 22 No 3, pp 90-93

COTTON I.W., GREATOREX F.S. (1967)
'Data structure and techniques for remote computer graphics'
Proc. AFIPS 1967 SJCC, Vol 26, pp 533-544

DAVIS M.R., ELLIS T.O. (1964)
'The RAND tablet : A man-machine communication device'
Proc. AFIPS 1964 FJCC, Vol 33, pp 325-331

DEPLEDGE P.G. (1981)
'Fault-tolerant computer systems'
IEE Proc. Part A, May 1981, Vol 128 No 4, pp 257-272

D'HULSTER F.M., DEKEYSER R.M., HEYSE J.E.,
VAN LAUWENBEWRGHE A.R. (1979)
'The computer as an aid for the implementation of advanced control algorithms on physical processes'
Proc. of IFAC symposium on "COMPUTER AIDED DESIGN OF CONTROL SYSTEMS", Zurich Aug 1979, pp 31-36, Pergamon Press

DINELEY J.L., PREECE C. (_967)
'A manual of KALDAS programming'
Oriel Press Ltd (UK) 1967

DODD G.G. (1969)
'Elements of data management systems'
Computing Surveys, Vol 1 No 2, July 1969, pp 117-133

DUYFJES G., DE JONG P.J., VERBRUGGEN H.B. (1977)
'Questionnaire on applications of modern control theory in process industry - results and comments'
Proc. of 5th IFAC/IFIP conf. on "DIGITAL COMPUTER APPLICATIONS TO PROCESS CONTROL", Hague 1977, pp 833-841, North-Holland Publishing Co.

EDWARDS F., LEE F.P. (1972)
'Man and computer in process control'
pp 4-5, Institute of Chemical Engineers (UK), 1972

EIDÉLSON A.F., ROBINSON I.J. (1980)
'Implementation of BEDSOCS : an interactive simulation language'
Computer Journal, British Computer Society, Feb 1980, pp 233-240

EVANS D., VAN DAM A. (1968)
'Data structure programming system'
Proc. IFIP Congress 1968, pp 67-72, Spartan Books Ltd.

FEHLBERG E. (1969)
'Low order classical Runge-Kutta formulas with step size control and their application to some heat transfer problems'
NASA REPORT TR R-315, G.C.Marchall Flight Center, Huntsville, Alabama, April 15 1969

FOLEY J.D., WALLACE V.L. (1974)
'The art of natural man-machine conversation'
Proc. IEEE, Vol 62 No 4, April 1974, pp 462-471

FRANKS A.J. (1968)
'B-LINE, Bell line drawing language'
Proc. AFIPS 1968 FJCC, Vol 33, pp 179-191

FREVERT L. (1975)
'Realtime language PEARL - concepts of language design and implementation'
Minicomputer Forum 1975, Conf. Proc., On-Line Conf. Ltd (UK), pp 183-191

GASPAR T.G., DOBROMTOFF V.V., BURGESS D.R. (1968)
'New process language uses English terms'
Control Engineering, Oct 1968, pp 118-121

GEARS C.W. (1971)
'Numerical initial value problems in ordinary differential Equations'
Prentice-Hall, Englewood Cliffs, N.J., 1971

GINO (1976)
'GINO-F user manual'
Computer Aided Design Centre, Cambridge, UK, 1976

GOOD M. (1981)
'ELUDE & the folklore of user interface design'
Proc. of ACM SIGPLAN SIGOA symposium on "TEXT MANIPULATION", Oregon, June 1981, SIGPLAN Notices, Vol 16 No 6, pp 34-43

GRAY J.C. (1967)
'Compound data structure for computer aided design : A survey'
Proc. ACM National Meeting, 1967, pp 355-365

GULLAND W.G. (1973)
'Continuous system simulation - a review paper'
Proc. of conf. on "COMPUTER AIDED CONTROL SYSTEM DESIGN", IEE, April 1973, pp 186-192

GURK H.M. (1955)
'The use of stability charts in the synthesis of numerical quadrature formulae'
Quarterly of Applied Mathematics, Vol 13 No 1, April 1955, pp 73-78

HALLIWELL J.D., EDWARDS T.A. (1977)
'A course in standard CORAL 66'
NCC Publication, National Computing Centre Ltd (UK)

HAMMING R.W. (1959)
'Stable predictor-corrector methods for ordinary differential equations'
Journal of the Association for Computing Machinery, Vol 6 No 1, Jan 1959, pp 37-47

HEALEY M. (1975)
'A survey of minicomputer applications in industrial control'
Minicomputer Forum 1975, Conf. Proc., On-Line Conf. Ltd (UK), pp 493-503

HEILMAN R.L., MARCHANT J.M. (1978)
'TIGS - an overview of the terminal independent graphics system'
Proc. of SIGGRAPH 1978, 5th Annual Conf. on "COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES", Altanta, Aug 1978, ACM, pp 93-107

HEROT C.F., WEINZAPFEL G. (1978)
'One point touch input of vector information for computer displays'
Proc. of SIGGRAPH 1978, 5th Annual Conf. on "COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES", Aug 1978, ACM, pp 210-216

HOBBS L.C. (1981)
'Computer graphics display hardware'
IEEE Computer Graphics and Applications, Vol 1 No 1, Jan 1981, p 25-39

HOROWITZ E., SAHNI S. (1976)
'Fundamentals of data structures'
Pitman Publishing Ltd, 1976, pp 106-168

HURLEY J.R., SKILES J.J. (1963)
'DYSAC - a digitally simulated analog computer'
Proc. of AFIPS 1963 SJCC, Vol 23, pp 69-82

ICHBIAH J.D., HELIARD J.C., ROUBINE O., BARNES J.G.P., KRIEG-BRUECHNER B., WICHMANN B.A. (1979)
'Preliminary ADA reference manual'
SIGPLAN Notice, Vol 14 No 6, June 1979

ICL (1974)
'SLAM - a simulation language for analogue modeling'
ICL 1900 series, ICL 1974

IECI (1968)
IEEE Transactions on Industrial Electronics and Control Instrumentation
Vol IECI-15 No 2, Dec 1968

IECI (1969)
IEEE Transactions on Industrial Electronics and Control Instrumentation
Vol IECI-16 No 3, Dec 1969

IPW/EWICS (1981)
'Draft Standard on Industrial Real Time FORTRAN'
Technical Committee of International Purdue Workshop on Industrial Computer Systems (IPW) and European Workshop on Industrial Computer Systems (EWICS), SIGPLAN Notice, Vol 16 No 7, July 1981, pp 45-60

JACKSON A.S. (1960)
'Analogue computation'
McGraw Hill, 1960, pp 255-257

JONES B. (1976)
'An extended ALGOL-60 for shaded computer graphics'
Proc. ACM symposium on "GRAPHICS LANGUAGES, COMPUTER GRAPHICS", Vol 10 No 1, 1976, pp 10-17

KELLY V.H., WAKEFIELD A.J. (1967)
'APEX - a new approach to programming for on-line control'
Proc. 2nd UKAC control convention on "ADVANCES IN COMPUTER CONTROL", April 1967, IEE publication No 29

KEY K.A. (1965)
'Analogue computer for beginners'
Chapman and Hall, London, 1965, pp 155-158

KNOWLTON K.C. (1969)
'A programmer's description of L6'
Comm. ACM, Vol 9 No 8, Aug 1969

KNUTH D.E. (1978)
'The art of computer programming Vol 1 : Fundamental Algorithms'
Addison-Wesley, 1978, pp 258-268

KOPAL Z. (1955)
'Numerical analysis'
Chapman and Hall, London, 1955.

KORN G.A., KORN T.M. (1972)
'Electronic analog & digital computer'
McGraw Hill, 1972

KORN G.A., WAIT J.V. (1978)
'Digital continuous system simulation'
Prentice-Hall, 1978
pp 79-99 DARE-P simulation language
pp 169-184 integration routines

KORN G.A. (1979)
'Real-time applications of computer-aided design'
Proc. of IFAC symposium on "COMPUTER AIDED DESIGN OF CONTROL SYSTEMS", Zurich 1979, (edited by CUENOD), Pergamon Press, pp 649-668

KRATZER G. (1979)
'Design and implementation of process control software under realistic environment conditions'
Proc. of 2nd IFAC/IFIP symposium on "SOFTWARE FOR COMPUTER CONTROL", Prague June 1979, (edited by NOVAK), pp 149-153, Pergamon Press

KULSRUD H.E. (1968)
'A general-purpose graphics language'
Comm. ACM, Vol 11 No 4, April 1968, pp 247-254

LANG C.A., GRAY J.C. (1968)
'ASP - a ring implemented associative structure package'
Comm. ACM, Vol 11 No 8, Aug 1968, pp 550-555

LEE W.T., BOARDMAN R.M., HIGHAM J.D. (1967)
'Block diagrammatic programming in computer control'
Proc. 2nd UKAC computer convention on "ADVANCES IN COMPUTER CONTROL", April 1967, IEE publication No 29

LUCAS J.J, WAIT J.V. (1975)
'DARE-P - a portable CSSL-type simulation language'
Simulation, Jan 1975, Vol 21 No 1, pp 17-27

LUCIDO A.P. (1978)
'An overview of directed beam graphics display hardware'
Computer, Nov 1978, pp 29-36

LINN C.Y., BARKER H.A. (1979)
'A process control system based on a graphical language'
Proc. of 3th International Conf on "TRENDS IN ON-LINE COMPUTER CONTROL SYSTEMS", March 1979, IEE 1979, pp 45-48

LINN C.Y. (1980)
'Digital control system programming from process control diagram'
PhD Thesis, Department of Electrical Engineering, University of Aston in Birmingham, July 1980.

MACHOVER C., NEIGHBORS M., STUART C. (1977)
'Graphics Display'
IEEE Spectrum Vol 14 No 8 , Aug 1977, pp 24-32
IEEE Spectrum Vol 14 No 10, Oct 1977, pp 22-27

MACHOVER C. (1977A)
'A brief personal history of computer graphics'
Computer, Nov 1978, pp 38-45

MARTENS H.R. (1969)
'A comparative study of digital integration methods'
Simulation, Feb 1969, Vol 12 No 2, pp 87-94

McMANIGAL D.F., STEVENSON D.A. (1980)
'Architecture of IBM 3277 graphics attachment'
IBM System Journal, Vol 19 No 3, 1980, pp 331-344

MEADS J.A. (1972)
'A terminal control language'
in "Graphics Languages" (edited by NAICE, ROSENFELD)
North-Holland, 1972

MERSON R.H. (1957)
'An operational method for the study of integration processes'
Proc. of Symposium on "DATA PROCESSING", Weapons Research Estab., Salisbury, South Australia, 1957

MILNE W.E., REYNOLDS R.P. (1960)
'Stability of a numerical solution of differential equations - part II'
Journal of the Association of Computing Machinery, Vol 7 No 1, Jan 1960, pp 46-56

MUSSTOPF G. (1979A)
'Microprocessor hardware and software'
Proc. of 2nd IFAC/IFIP symposium on "SOFTWARE FOR COMPUTER CONTROL", Prague June 1979, pp 23-50, Pergamon Press

MUSSTOPF G., ORLOWSKI H., TAMM B. (1979B)
'Program generators for process control applications'
Proc. of 2nd IFAC/IFIP symposium on "SOFTWARE FOR COMPUTER CONTROL", Prague June 1979, pp 11-22, Pergamon Press

NEWMAN W.M., SPROULL R.F. (1979)
'Principles of interactive computer graphics (2nd ed.)'
McGraw Hill, 1979
pp 147-158 graphical input devices
pp 159-182 picking selection methods
pp 443-478 userface interface design

NEWMAN W.M, VAN DAM A. (1978)
'Recent efforts towards graphics standardisation'
Computing Surveys, Vol 10 No 4, Dec 1978, pp 365-380

NOBLE J.S. (1977)
'The evolution of process control software'
Proc. symposium on "DEDICATED DIGITAL CONTROL", University of Aston in Birmingham, 1977, Institute of Measurement and Control (UK)

OHLSON M. (1978)
'System design consideration for graphics input devices'
Computer, Nov 1978, pp 9-18

PAYNE C.A.J. (1974)
'Programming by block diagrams - a computer language to suit the process engineer'
Canadian Control & Instrumentation, Dec 1974, Vol 13 No 12, pp 25-30

PIKE H.E. (1970)
'Process Control Software'
Proc. of IEEE, Vol 58 No 1, Jan 1970, pp 87-97

PIKE H.E. (1972)
'Future trends in software development for real-time industrial automation'
Proc. AFIPS 1972 SJCC, Vol 40, pp 915-923

PRESIS R.B. (1978)
'Storage CRT display terminals : evolution & trends'
Computer, Nov 1978, pp 20-26

PRITSKER A.A.B., PEGDEN C.D. (1979)
'Introduction to simulation and SLAM'
John Wiley & Sons, 1979

RAGAZZINI J.R., FRANKLIN G.F. (1958)
'Sampled-data control system'
McGraw Hill, 1958, pp 117-144

REVETT M.C. (1973)
'Control system design using ADSOL, an on-line digital simulation program'
Proc. of conf. on "COMPUTER AIDED CONTROL SYSTEM DESIGN", IEE, April 1973, pp 193-197

ROSKO J.S. (1972)
'Digital simulation of physical system'
Addison-Wesley, 1972
pp 372 sorting & sequencing
pp 402-422 integration approximation

ROSS D.T., RODRIGUEZ J.E. (1963)
'Theoretical foundation for computer aided design system'
Proc. AFIPS 1963 SJCC, Vol 23, pp 305-322

ROVNER P.D., FLEDMAN J.A. (1968)
'The LEAP language & data structure'
Proc. IFIP Congress 1968, Vol 1, pp 579-585, Spartan Books Ltd.

RZEHAK H. (1978)
'Redundancy in hardware and software of process control'
Proc. of IMACS symposium on "SIMULATION OF CONTROL SYSTEMS", (edited by TROCH I.), North-Holland Publishing Co., pp 7-15

SCHARK G. (1976)
'Design, implementation and experiences with a higher-level graphics language for interactive computer aided design purpose'
Proc. ACM symposium on "GRAPHICS LANGUAGES, COMPUTER GRAPHICS", Vol 10 No 1, 1976, pp 18-23

SCIAM (1977)
Scientific American - special issue on microelectronics.
Sept. 1977, Vol 237 No 3.

SCHOEFFLER J.D. , TEMPLE R.H. (1970)
'A real-time langauge for industrial process control'
Proc. IEEE, Jan 1970, Vol 58 No 1, pp 98-106

SCHOEFFLER J.D. (1972)
'The development of process control software'
Proc. AFIPS 1972 SJCC, Vol 40, pp 907-914

SCi SOFTWARE COMMITTEE (1967)
'The SCi continuous system simulation language (CSSL)'
Simulation, Dec 1967, pp 281-303

SHAMPINE L.F., WAIT H.A., DAVENPORT S.H. (1976)
'Solving non-stiff ordinary differential equations - the State of the Art'
SIAM Review, Vol 18 No 3, July 1976, pp 376-411

SHAPIRO K.G. (1978)
'Data structure for picture processing'
Proc. of SIGGRAPH 1978 5th Annual Conf. on "COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES", Altanta, Aug 1978, pp 140-146

SIMPLEPLOT (1978)
'SIMPLEPLOT manual'
UMRCC, 1978

SMITH C.L. (1972)
'Digital computer process control'
Intext Educational Publishers, 1972
pp 91 quote on ZOH
pp 166-179 PID
pp 184-200 process identification methods

SMITH D.N. (1971)
'GPL/I : A PL/I extension for computer graphics'
Proc. AFIPS 1971 SJCC, Vol 38, pp 511-528

SPECKHART F.H., GREEN W.L. (1976)
'A guide to using CSMP'
Prentice-Hall Inc., 1976, pp 168-172

STRAUSS J.C. (1968)
'Digital simulation of continuous dynamic systems : An overview'
Proc. of AFIPS 1968 FJCC, Vol 33, pp 339-344

STEUSLOFF H.U. (1979)
'Programming distributed computer systems with higher level languages'
Proc. IFAC/IFIP workshop on "DISTRIBUTED COMPUTER CONTROL SYSTEMS", Tampa, Oct 1979, pp 39-50, Pergamon Press

SUTHERLAND I.E. (1963)
'SKETCHPAD - A man-machine graphical communication device'
Proc. AFIPS 1963 SJCC, Vol 23, pp 329-346

TANIMOTO (1976)
'An iconic/symbolic data structuring scheme'
in "Pattern recognition and artificial intelligence", Academic Press, 1976, pp 453-471

TEKTRONIX (1976A)
'Tektronix 4051 graphic system operator's manual'
Tektronix Inc., 1976

TEKTRONIX (1976B)
'Tektronix 4051 option 1 data communication interface instruction manual'
Tektronix Inc., 1976

TEXAS INSTRUMENTS (1978)
'Model 990 computer reference manual, Volume 1 Concepts and Facilities'
Texas Instruments, 1978

THALMANN N.M., THALMANN D. (1981)
'A graphical PASCAL extension based on graphical types'
Software : Practice & Experience, Vol 11 No 1, Jan 1981, pp 53-62

TOCZYLOWSKI E. (1978)
'Large scale steady state process simulation in design of supervisory control'
Proc. of IMACS symposium on "SIMULATION OF CONTROL SYSTEMS", (edited by TROCH I.), Sept 1978, North Holland Pub. Co., pp 55-61

VAN DAM A. (1971)
'Data and storage structure for interactive graphics'
Proc. symposium on "DATA STRUCTURE IN PROGRAMMING LANGUAGES", SIGPLAN Notice, Vol 6 No 2, Feb 1971, pp 237-267

VAN DAM A., EVANS D. (1967)
'A compact data structure for storing, retrieving and manipulating line drawings'
Proc. AFIPS 1967 SJCC, Vol 30, pp 601-610

VARGA A., SIMA V., POPTESCU Th., VASILIU C. (1979)
'Process control algorithms for microprocessors'
Proc. of 2nd IFAC/IFIP symposium on "SOFTWARE FOR COMPUTER CONTROL", Prague 1979, pp 161-164, Pergamon Press

WELLER D.L., CARLSON E.D., GIDDINGS G.M., PALERMO F.P., WILLIAMS R., ZILLES S.N. (1980)
'Software architecture for graphical interaction'
IBM System Journal, Vol 19 No 3, 1980, pp 314-330

WILKE J.D.F. (1979)
'A microprocessor philosophy for process control systems'
Proc. of 3rd International Conf. on "TRENDS IN ON-LINE COMPUTER CONTROL SYSTEMS", March 1979, IEE 1979, pp 115-120

WILLIAMS R. (1971)
'A survey of data structure for computer graphics systems'
Computer Survey, Vol 3 No 1, March 1971, pp 1-21

WILSON J.R., PRITSKER A.A.B. (1978A)
'A survey of research on the simulation start-up problem'
Simulation, Vol 31 No 2, Aug 1978, pp 55-58

WILSON J.R., PRITSKER A.A.B. (1978B)
'Evaluation of startup policies in simulation experiment'
Simulation, Vol 31 No 3, Sept 1978, pp 79-89

WIRTH N. (1976)
'Algorithms + data structure = programs'
Prentice Hall, 1976
pp xiii quote for data & algorithm
pp 163-182 linked list data structure
pp 182-189 topological sorting

WOODWARD P.M., WETHERALL P.R., GORMAN B. (1974)
'Official Definition of CORAL 66 (3rd Ed)'
HMSO (UK) 1974

YOURDON E. (1975)
'Techniques of program structure and design'
Prentice-Hall Inc., 1975, pp 36-77