University of South Wales

2053147

A Real-time Control and Modelling System based upon Logic Simulation

Martin Alfred McCabe. B.Tech., C.Eng., M.I.E.E.

A dissertation submitted to the Council for National Academic Awards for the degree of Master of Philosophy.

The Polytechnic of Wales
Department of Electrical and Electronic Engineering.

July 1987

Declaration

This dissertation has not been, nor is being currently submitted for the award of any other degree or similar qualification.

M. A. McCabe

Acknowledgements

The author wishes to thank the Polytechnic of Wales and Dr Richard Murray-Shelley for their support for this project.

In particular, he would like to thank his wife for her strenuous efforts in the typing and correcting of this thesis as well as her support throughout the project.

Table of Contents

Synopsis

1 Introduction 1

2 Background 3

3 Suitable languages 5

4 Control language requirements 7

5 Control language Philosophy 9

6 Language structure 12

6.1 Functions 12

6.2 Junctions 13

6.3 Structural example 14

7 Functions and the function list 16

7.1 Input/output definition in the list 17

7.2 Function types 18

8 Building Program 20

8.1 Development environment 21

8.2 Storage requirements 24

8.3 Function entry 24

8.4 Function selection 25

8.5 Function numbering 26

8.6 Drawing connecting wires 26

8.7 Selections after connecting 27

8.8 Program Documentation 28

9 Testing Facilities 30

9.1 Junction Structure 31

10 Monitoring Junctions 33

10.1 On screen junctions 34

10.2 Off screen junctions 34

11 Displaying, changing and blocking junctions and monitors 36

11.1 Displaying a junctions contents 36

11.2 Changing the actual value 36

11.3 Blocking and unblocking 37

11.4 Test example 37

12 Function Interpreter 39

12.1 Sub-lists 41

12.1.1 Sub-list-1 (last page) 42

12.1.2 Sub-list-2 (last page - 1) 43

13 Multi-plant control 44

13.1 Using offset functions 45

13.2 Designing with Offsets 46

14 Circuit design 49

14.1 Oscillators and Timers 49

14.2 Alarm Detector 50

14.3 Set-Point Calculator 52

14.4 Inputs and Outputs 52

14.5 Distributed Process Control 53

14.6 Remote development 54

14.7 Command Breakdown 57

15 Hardware 59

16 Real-time executive 61

16.1 Existing executives 62

16.2 Executive operation. 63

16.3 Executive implementation. 64

16.4 Task Control 65

16.5 Executive Task Time Control Example 66

16.6 Integrating DOS 67

16.7 Problems with DOS integration 69

17 Other Applications 70

17.1 Control Simulation 70

17.2 Logic Simulation 71

18 Function Characteristics 72

18.1 Digital Logic Functions 73

18.2 Miscellaneous Digital Functions 74

18.3 Analogue Functions 77

18.4 Maths Functions 85

18.5 List Control Functions 88

18.6 Input/Output Functions 90

19 User Interface 92

19.1 Plot Program language selection 94

20 Performance Evaluation 96

References

Appendix A - Development Facilities

Appendix B - Function Diagrams

Appendix C - Program Examples

Appendix D - Builder Screen Displays

Appendix E - Published Papers

Synopsis

This thesis introduces the difficulties encountered when designing software for real-time process control applications. The currently available design tools are discussed and a dual language system is proposed as a solution.

A novel graphics based programming language and development environment is described which attempts to overcome the many restrictions of conventional high level languages. This language is significantly more reliable and maintainable than text based languages; this opens up the possibility for sophisticated process control programming being undertaken by electronic or process engineers as well as experienced programmers.

The philosophy and structure of the graphics based language is discussed along with a description of the function building facilities, the programming environment and the extensive tracing and simulation facilities.

Several problems which complicate the task of real-time software development are introduced such as concurrency, multi-plant and distributed control. The language system offers a unique solution to such difficulties which greatly simplifies the programmers task.

Details of the language functions are given along with examples of its use and integration with user interfaces.

1 Introduction

The real-time microcomputer control of an industrial process is a particularly difficult task to undertake. The requirements for rapid response to external signals and the often complex graphical operator interfaces demanded by users results in a severe strain on computer processing power and programming effort.

The programming task often divides into two parts. One part must handle the urgent task of monitoring many input signals and generating control outputs, the other less urgent part must interface between the parameters describing and controlling the process, and an operator or manager who requires this information in its most convenient form. The main thrust of this research is to separate these two tasks and provide a novel solution to the real-time control problems in order to produce flexible and reliable control software with the minimum of programming experience and effort.

The solution to be presented provides the following facilities:-

1. Segregation and isolation of the high priority real-time control software from the operator interface without sacrificing inter-task data exchange.

2. A control oriented language with inherent "parallel processing" and resistance to programming errors.
3. A graphics based design and debug environment to support the control language.
4. Full on-line facilities for program changes and monitoring.

2 Background

The current situation with respect to microcomputer software design for industrial process control is to select a high level language for the bulk of the project and use assembly language when speed is important. The high level language is usually a general purpose one such as "C" or Pascal, which is capable of almost any programming task but not very well suited to any particular one relevant to a control system. Other important ingredients include an operating environment which is usually a general purpose real-time executive, and debugging facilities which usually require an expensive emulator. These development facilities become progressively more ineffective as the control task becomes larger and are very cumbersome once the program is operating in the field.

When control programs require development, the average industrial control engineer must either make use of a professional programmer to translate the control strategy into a conventional text-based language such as "C", or make use of a relatively limited proprietary language available from companies such as Burroughs. This second option offers many of the solutions to the development problems but suffers from several drawbacks. The most obvious drawback is that special computers must be used for program development and for the target computer; other

drawbacks include the relatively powerful but inflexible language facilities and testing facilities available and the relatively high cost of the equipment which can usually only be purchased from one supplier.

3 Suitable languages

Many programming languages have been developed for both general purpose use and special purpose use. The first step in solving the special problems of the process control engineer was to investigate the suitability of existing languages.

Most control programs can be divided into two interactive parts; one section handles the man/machine interface whilst the other section handles the second to second plant control. It was evident from examining several languages that some were very suitable for the man/machine interface. These include Pascal, Modula 2 and most general purpose languages but in particular, modern dialects of Basic. When selecting a language for the plant control, the choice is severely limited since few languages include such facilities as 3-term control and filters as standard and those that do are often dedicated to expensive proprietary computers which lack the flexibility to interface to other modern software.

It became very clear early in the project that it must be accepted that no one single language had all of the facilities required by the industrial process engineer. It also became clear that the development of control programs was the most difficult task since man/machine interfaces could be relatively easily designed using modern high level

languages.

Before considering the special requirements of the real-time control section, some explanation is necessary for the suggestion that the Basic language would be the most suitable choice for generating a user interface. The modern Basic language has progressed a long way from its origins as a simple teaching language. Recent dialects enable much more structuring to be applied to the design and are often supplied with the most comprehensive facilities for graphics displays and user input. One very important benefit is the ability to run Basic as an interpreted or compiled language which offers the convenience of program development using an interpreter but the speed, once developed, of a compiled program.

Last but not least is the apparent user friendliness presented by a Basic interpreter which often makes it the first choice language for non-professional programmers. This important feature is significant because it is the aim of the project to produce an environment which allows program development and modification by process control engineers who are not necessarily full-time professional programmers.

4 Control language requirements

Once the software required is divided into two parts, one of which can be implemented using existing interpreters and compilers, it becomes necessary to define the requirements for real-time control program development. These requirements are outlined below:-

1. A high level for Process control.
2. Impossible to "hang-up" even when incorrectly programmed.
3. Multi-tasking with dynamic task priorities.
4. Modular development facilities.
5. Built-in functions orientated towards the requirements of process control.
6. Facility for on-line program changes.
7. Extensive on-line monitoring of plant inputs and outputs as well as internal program variables.
8. Simulation of plant inputs.
9. Protection of critical outputs to the plant during on-line fault finding.
10. Simple interfacing to and from the user interface.
11. Simple interfacing to and from remote outstations.
12. Fast execution speed.

Clearly from the above list, no one existing language will satisfy all of the requirements. The most difficult

requirement which encompasses many items in the above list is to provide simultaneous program execution, program changes and program monitoring. Quite apart from actually providing the facilities, the most important problem is how to present these powerful facilities to a process control engineer with as little complexity as possible.

The process control language requirements previously defined are comprehensive and potentially very difficult to manage. Conventional languages are largely based on text which does not reflect program flow very well and does not include easy monitoring of inputs and outputs. A text based language can be very difficult to follow unless it is very well structured, especially when operating in a multi-tasking environment where task interaction is not always obvious.

5 Control language Philosophy

It was noted very early on that there was some considerable similarity between the requirements for a real-time language and the facilities provided for conventional electronic hardware development. Electronic circuits are inherently multi-tasking since they continuously operate. It is possible to monitor logic states and voltages at any point using voltmeters without stopping the circuit. The use of specialised integrated circuits allows modular development and the concept of a circuit diagram permits the signal flow to be easily seen.

If software could be designed to simulate the desirable characteristics of hardware, the problem of multi-tasking would be effectively solved. The simulation could be carried further to permit software to be developed by "drawing" circuits onto a VDU screen and providing simulated logic probes and digital meter displays for any junction within the circuit. The analogy could be extended to include special purpose "Integrated Circuits" simulated in software for such functions as logic, arithmetic and control algorithms. Because these functions would be linked by lines to other functions the signal flow within the software would be obvious in the same way that conventional electronic circuit diagrams show the paths for current flow.

Although simulating an electronic circuit in software would provide many of the advantages of electronic design, the problems of electronic design of course need not be simulated; these include power supplies, noise, signal loading problems and inserting new components into operating circuits.

Because the concept of software design using circuit design techniques is so different from other software tools, a complete development environment had to be produced. Before defining the capabilities of such an environment, however, the actual "circuit" capabilities had to be defined. This should include the type of functions available, how they should be connected and how they should operate to give the illusion of hardware circuits.

Two fundamental elements of this new language could be identified; the function (equivalent to an Integrated Circuit) and the junction (equivalent to a circuit connection). The function had to have a variable number of inputs and outputs which are connected to the inputs and outputs of other functions to produce a circuit. All connections between functions must be made by joining each input or output to junctions only; this enables all inputs and outputs to be monitored during execution. Functions necessary for process control must be able to handle two types of value; a digital signal and an analogue signal. Two types of junction are therefore provided; the digital type can hold either a 0 or 1 and the analogue type can contain a 16 bit signed integer.

The analogue value limitation of +32767 to -32768 was made quite intentionally since this seemed to offer the optimum choice between simplicity, execution speed and the actual requirements in process control situations. Since most analogue inputs from industrial sensors do not exceed a 12 bit integer, there would be an unnecessary overhead if the precision were to be greater than that finally chosen. In view of the requirement for precision during a mathematical scaling process, provision has been made for some inputs and outputs from mathematical functions to handle signed 32 bit integers.

6 Language structure

6.1 Functions

Lists of functions are created in memory by a building program. The function is stored in the list as a function code to specify the type of function followed by a list of junction numbers, one for each function input and output. The list of functions is divided into sections called pages; this is a convenient way of dividing a large design into small modules which can be completely displayed on one screen. There is no practical limit to the number of pages and the only limitation to the number of functions in a page is the space available on the VDU screen during the design of a page.

A built in library of over 40 functions ranging from 2 input AND gates to 3-term controllers is available. To use a function, it must be drawn onto the VDU screen as part of a circuit "page" and its inputs and outputs connected to numbered junctions. Information is transferred from one function to another by joining the output of one function to the same junction as the input of the next function. This is of course the same as conventional hardware logic design but without any soldering.

Once a function has been entered, it is included into the function list which is being continuously interpreted.

The interpreter simply works down this list of functions on a regular time basis (say once every 0.1 seconds) and would normally interpret every function in that list. Each function appears to the interpreter as a code to identify the function type (OR gate, multiplier etc.) followed by a list of input/output junction addresses in RAM. Data is taken from the junctions connected to the function inputs, processed by the specified function program, and the results written into the junction connected to the function output.

The function list is a linear sequence of function type and input/output information. This is divided into sections or pages for the convenience of displaying on a VDU. The list is executed regularly (by the interpreter) in a fixed order from the start to the end. The execution order is important to ensure that input changes work their way to the outputs as rapidly as possible and preferably in one pass of the interpreter through the list. The function number determines its position within the execution order for that page.

6.2 Junctions

A junction may be either an analogue or digital type depending upon the type of data being held. A digital junction can contain either a 0 or a 1 and would be used for example, to connect two AND gates together. An

analogue junction can contain an integer between -32768 and +32767. Each junction is given a reference number which is used to determine its actual location in memory.

One of the most difficult problems associated with process control software development is how to test the program when the software and/or the hardware is incomplete and few or no hardware inputs/outputs exist. The consequences of having little control over inputs and outputs is also very apparent when commissioning or modifying hardware and software on-site and/or on-line.

This new language system attempts to reduce these debugging problems by building in a blocking ability into every analogue and digital junction. Each junction contains an actual value, a blocked value and a flag to indicate which value is to be taken as an input to a function. All function outputs only affect the actual value stored in a junction. Once the building program blocks a junction during program development, any function using this junction for its input data will take the blocked value rather than the actual value. The blocking ability not only allows inputs to be simulated but also allows intermediate connections to be temporarily broken without disturbing functions using the connection as an input.

6.3 Structural example

The builder constructs a list of function descriptions in

memory which are executed by the interpreter program. The format of a function within the list is very simple and may be illustrated by considering a 2 input AND gate. This occupies 7 bytes as follows:

- 1 byte function code for an AND gate,
- 2 byte "input A" junction address,
- 2 byte "input B" junction address and a
- 2 byte "output" junction address.

The junction address is in fact the relative address of the RAM location used for input or output data. Included in these digital junction addresses is a reserved bit to indicate when the data should be inverted before use or inverted before output. This inversion capability enables a basic AND gate to be used as a NAND or in fact an OR gate.

The function list is a continuous linear sequence of such function descriptions. The interpreter simply works down this list, reads the next function code and jumps to a function handling routine determined by that code. The function handling routine will execute a predetermined function (in the above case, a digital AND function) using the data locations specified in the function list. After executing this function, control is passed back to the interpreter which proceeds to the next function description in the list.

7 Functions and the function list

The function list is a single continuous sequence of function definitions. The list is immediately preceeded in memory by a page start address table which gives a 2-byte relative offset from the start of the function list for each page in the list.

p = number of pages in the system.

s = start address of "page start address table".

```
address s              |          0                |
address s+2            |    page 2 start offset    |
address s+4            |    page 3 start offset    |
address s+6            |    page 4 start offset    |
                       |          :                |
                       |          :                |
                       |          :                |
address s+(p*2-2)      |    page p-1 start offset  |
address s+(p*2)        |    page p start offset    |
address s+(p*2+2)      |    start of list offset   |
                       |          :                |  start of list
                       |          :                |
                       |          0                |  end of list
```

Each function definition within the list consists of (2*n+1) bytes where n is the number of input/output connections to the function.

The first byte is always a code indicating the function type; each input/output connection is defined in 2 bytes. The input/output definition is actually the relative address of the junction used to obtain input information or to store a result.

7.1 Input/output definition in the list

There are 2 types of input/output definition:

Digital input/output

Bits 0-14 contains the offset address of the junction relative to the start of the digital junctions. Bit 15 is set to 1 if the data is to be inverted.

Analogue input/output

Bits 0-15 contains the offset address of the junction relative to the start of the analogue junctions. This will be in multiples of 5.

The function routine has knowledge of whether the definition is for an analogue or digital junction. The order of input/output definitions within a function definition is chosen for the convenience of the function program to enable it to execute as fast as possible.

The functions are in general unaware of page boundaries. The list execution tasks will continue until the end of the list which is indicated by a function code of zero.

The only exception to the above is the Constant function. The 2 bytes following the constant function code is the constant value and not an input/output definition.

7.2 Function types

The digital functions provided include:

- simple logic; 2/4 input AND, OR, EXOR, buffers etc.,
- memory functions; flip-flops and latches,
- integrator.

Analogue functions are described in the same way except that the provision for data inversion with an analogue junction is not included. All analogue functions operate with integers to 2-byte precision. It is rare that this limitation becomes a problem in process-control with sensible number manipulation.

The type of analogue functions provided include:

- arithmetic; multiply, divide, square root etc.,
- comparisons; equality, greater than etc.,
- counters/timers,
- data tables,
- multiplexers, demultiplexers,
- input/output; digital and analogue,
- special purpose; 3-term controller, filter etc.

Functions may consist of a mixture of digital and analogue inputs and outputs. Compare functions for example use 2 analogue junctions for input data and use a digital junction for the result.

Although the present list of functions will meet the

demands of many applications, it is possible to extend the system if other special purpose functions are necessary.

8 Building Program

The building program enables the engineer to build up a function list which being interpreted in real-time. The design is drawn onto a VDU screen using block graphics to represent the functions, junctions and connecting wires. A keyboard and joystick/mouse are used to select the desired function and to "wire" it up to junctions on the screen. The screen positions of the functions, junctions, connecting wires or indeed any text comments about the operation of the circuit are determined by the engineer. The only information required by the interpreter is the function code and the junction numbers to which each input/output line connects. The rest of the information drawn onto the screen is to permit easy understanding of the circuit and to provide a commented and readable printout for documentation.

Effectively then, a logic circuit is drawn onto the screen in a similar manner to hardware design using a CAD computer.

The action of drawing a circuit onto the screen enters the function into the function list being interpreted. A complete sub-circuit can be constructed and saved as a page onto disk.

Once a page of logic has been constructed, it may be stored on floppy disk to enable another to be constructed

or loaded from disk. The maximum number of pages that can be included in a system is largely determined by disk capacity, but a maximum of between 40 and 60 permits quite large controllers to be designed. This page information is not required by the interpreter and so disks need not be on-line once an application has been fully developed.

Being able to recall and edit pages of logic from the disk enables a designer to build a library of commonly used circuits. For example, a set-point ramp generator may be developed, tested and saved. Any application requiring this facility will simply necessitate loading this page into the building program, re-numbering the junctions and saving as part of the new application.

8.1 Development environment

The program development environment was designed to be simple with as much in-context assistance given as possible. The complete development environment operates concurrently with the execution of the function list but as an alternative to the user interface program.

Since the building program is essentially a graphics based CAD system for non-professional programmers it has to be foolproof in operation with "help" facilities and prompts. Some form of pointing device was necessary as an alternative to moving the cursor on the screen using the keyboard and therefore a joystick was initially selected.

Interfacing with a mouse has subsequently been developed; a graphics tablet would be possible if desired.

During the initial design phase of the building program it became necessary to choose the form that the graphics display should take. There were two choices:-

- Full bit mapped graphics display.
- Character based graphics display.

Most general purpose CAD computers use a bit mapped graphics display to enable any shape to be displayed. This has severe drawbacks in that display update is very computer intensive and saving such displays requires large amounts of disk and memory space. These problems are much more significant if colour is considered desirable.

Because of the above-mentioned problems and the fact that using say 16 colours is very useful in portraying large amounts of information successfully, a character based graphics display was chosen. This is a fast and efficient graphics system but it places severe limitations on the shapes of the functions and upon the routes taken by interconnecting wires on the screen. Both of these constraints are not as important as they may first appear since they force a form of standardisation of function shape and prevent cluttered interconnections on the screen. The final display used 2000 character cells, 16 foreground, 8 background colours and instant hardware selection of 4 complete screens at any time.

The screen limitations imposed restrictions upon how

function inputs and outputs could be drawn connected to junctions. Only horizontal and vertical lines are possible with a fixed minimum distance between adjacent lines. All of this is drawn by the program with the user simply guiding the desired connecting line around the screen. This semi-automatic routing of interconnections is not only much more straightforward to implement, but ultimately also more visually correct than a full auto-routing system.

The interface between the development environment and the software designer was given considerable thought. It was finally designed as a command driven system with full prompting during commands. This type of interface uses one line only of the display which permits a single screen to be used for circuit display and command input. Pressing a single letter initiates one of more than 20 commands which then proceed by asking a series of questions until the desired action is complete. At all times data entry is verified. At any time a command may be terminated by pressing the ESC key which returns the program back to waiting for a command. It was decided that any action, including aborting a command, which caused a significant loss of data should then give an option not to continue.

All commands are executed concurrently with any other update actions. If monitoring of junction contents on the screen is taking place then this will continue at all times.

8.2 Storage requirements

A system suitable for industrial process control should not rely upon mechanical storage for its normal execution. The development environment which is not used during normal process control execution does however depend upon disk storage. All information which is only required for development is stored on disk and is recalled whenever a new page of functions is loaded. This information includes screen information and junction monitor information. The actual function information which will be inserted into the function list is also saved onto disk but only has significance when a system is loaded initially.

Facilities were provided for loading, saving, erasing and copying pages of functions as well as loading in a completely new list of functions to develop an alternative application. Protection features have been included to prevent unintentional loss of information and changes within a page.

8.3 Function entry

Functions are drawn onto the screen in a suitably large free area by selecting the desired function either from a full page list or by typing in all or part of its name. A sequence of questions is then asked which positions the functions within the execution order within a page and

guides the user towards connecting every input and output to a valid analogue or digital junction. It is not possible to carry out any other operation until a function is completely drawn and connected. Once complete, the function is placed into the function list without stopping or slowing down the execution of the function list to enable program changes to be made on-line.

8.4 Function selection

A function may be selected using one of 2 methods:

1. Typing the function name in response to the "Name of function" question. Only part of a name has to be given since a search is made of the list of function names for a match between the name entered and all or part of each function name; the first match found will decide the function. For example, typing "**and**" will select a "2 I/P **and**" function since this is a match and it occurs before "4 I/P **and**" and "sample **and** hold".
2. Pressing the <enter> key in response to the "Name of function" question will display a complete list of functions available. To select a function, enter the number of the function which is displayed to the left of the name.

8.5 Function numbering

Each function is assigned a reference number which is displayed in a different colour at the bottom right of the function; this number indicates the execution order of the function in the function list. Normally, functions are numbered in a logical order from input to output such that an input signal is processed as fast as possible (i.e. one scan of the function list) to give a new output.

A function may be inserted into a page as the last function executed by responding to the "Enter the function execution order" question with the <enter> key. A function may be inserted between existing functions by entering the desired reference number. If a function is inserted into a page containing other functions then the reference numbers are incremented for every function after the new one.

8.6 Drawing connecting wires

Once a function has been drawn onto the screen and numbered, it must be completely wired up before any other operation can be performed. Each input/output connection of the function must be joined by a line to an appropriate junction (analogue or digital); the connection order is from the top to the bottom and from the left to the right of the function. A flashing `diamond´ shape indicates the input/output function point currently being connected.

A line is drawn by positioning the cursor and pressing any key except <Esc>, <j> and <d>. A line is automatically drawn from the last line end to the cursor subject to the following rules:

- a line will first of all continue in its original direction as far as necessary.
- The additional section of line can have a maximum of one corner. An exception is the very first section of line connected to the function which may have two corners under certain circumstances.
- The new line may cross over an existing line or character but it may not cross over more than one adjacent line or character.
- If a blockage is detected (i.e. more than one adjacent line or character, a function, junction, monitor or word), a Beep will sound and no line will be drawn.

An input/output point on a function is considered connected when it is linked into the correct type of junction.

8.7 Selections after connecting

If the input/output connection is to a digital junction then an option is given to invert the connection or not.

The constant function is the only function which is not completely connected by linking to junctions. This

function requires a constant value to be entered in decimal or hex in response to the appropriate question presented after connecting the functions´ only input/output point. The format of the number displayed in the function is the same as the format that was used to enter the number (i.e. decimal or hex).

8.8 Program Documentation

Projects using conventional text based languages must be supported by extensive documentation and flowcharts. The graphics based language described here greatly reduces the need for extensive documentation due to ones ability to understand more from a picture indicating signal flows than a page of text. It is possible to compare a page of interconnected functions to a flowchart which of course implies that the design can be accomplished at a higher level than a general purpose text language.

In order to enhance the usefulness of the page display as a fully documented program, facilities were added to the building program to allow text to be entered in any available foreground and background colour, even flashing text. This text is simply part of the picture seen by the user and forms no active part in the control software.

In addition to text, graphics characters can also be included. This enables not only comments to be added to a page of functions but also complete pages of comment. This

latter extension can produce complete mimic panel type pictures showing a block graphics picture of an industrial plant with the addition of important parameter or alarm displays as discussed later under junction monitors. Several commented displays of this type can form the first form of assistance to the process control engineer when the control software does not give the correct response in some circumstances after installation.

It should be noted that a page is an arbitrary divider of functions in the function list. A page need not contain any functions but can still be loaded as normal from disk storage. One recommended use for the first page in a system is for a list of contents for all other pages in the system. This would take the form of a text only display showing the use of each page.

9 Testing Facilities

Testing and fault finding are perhaps the most important aspects of designing a good program. Most languages allow programs to be designed using powerful facilities but largely neglect the important task of providing test facilities.

The majority of process control computers will have their software modified after installation. This situation can create many problems which do not arise during the initial design:-

1. The computer to be modified may not be able to be taken off-line completely.
2. Control outputs may have to be tested without actually sending these signals to the plant being controlled.
3. Inputs have to be simulated when they may be being held in one state by sensors.
4. Conventional program debugging usually requires the program to stop temporarily whilst variables are examined and changed.

To overcome these problems, extensive testing facilities have been incorporated into the design environment to assist in the initial design phase as well as when problems are encountered or when changes are necessary after installation.

9.1 Junction Structure

Junctions are not just simple locations in RAM and do not act like variables in a text based language. Junctions are specially structured memory locations which hold 3 items; an actual value, a blocked value and a blocked flag.

Actual value	A function always outputs to the actual value part of a junction. A function will take the actual value as an input unless the blocked flag is set.
Blocked value	If the blocked flag is set to 1 then this value will be used as input data to a function.
Blocked flag	Set to 1 if a junction is blocked.

There are 2 types of junctions which share the above structure but differ in the maximum size of the value:

Digital Junctions

These occupy 1 byte each and contain a value of 0 or 1. Bit 0 is the actual value, bit 1 is the blocked value and bit 2 is the blocked flag.

Analogue Junctions

These occupy 5 bytes each and contain a value between -32768 and +32767 stored as a 2 byte signed integer. The first byte is

the blocked flag (only bit 0 is used for this purpose); bytes 1 and 2 contain the actual value and bytes 3 and 4 contain the blocked value. The numbers are stored least significant byte first.

10 Monitoring Junctions

A monitor is a real-time display of a junctions´ contents, both actual and blocked. Any junction may be monitored in the masterstation or any outstation.

The display may be in decimal or hex; this is selected using the output radix toggle (<i>) command when the cursor is over the monitor value block or the monitor junction number. A decimal number is displayed as a signed value with up to 5 digits; a hex number is displayed as an unsigned value between 0H and FFFFH. The <i> command operates as a toggle between decimal and hex.

Up to 20 junctions may be monitored on the screen at any one time. These may display the contents of junctions on the screen or may be monitor-junctions set up to display the contents of junctions off the screen.

Both types of junction monitor display share a common format to display the actual and blocked junction contents. The values are contained within an inverse video block 2 lines high by 1 wide for a digital junction and by 6 wide for an analogue junction. The top line contains the actual value and the bottom line the blocked value. If the bottom line is empty, the junction is not blocked.

If an attempt is made to display more than 20 monitors, a warning message is given.

10.1 On screen junctions

Typing <m> with the cursor over a junction or junction number when the main command prompt is displayed will define a new monitor displaying that junctions contents. The program will prompt for the cursor to be moved to the display position for that monitor which will be displayed when a key is pressed. As is usual with all display positioning, the monitor block will be positioned with its top left corner on the cursor position; a warning message is given if there is not enough free screen space for the monitor in the chosen position.

10.2 Off screen junctions

To display the contents of a junction not on the screen, type <m> with the cursor over an empty section of screen in response to the main command prompt. This will define a new monitor consisting of a monitor junction, monitor junction number and monitor value block; The monitor junction and number are not the same as normal junctions and cannot be connected to functions.

The program first of all requests the outstation containing desired junction and then proceeds to request the junction type and number in the same manner as placing a normal junction on the screen using the <j> command. The final prompt is the same as given when monitoring an

on-screen junction. An off screen junction monitor number has a prefix to indicate the outstation relevant to that monitor. The prefix ´m´ indicates masterstation and the letters ´A´ to ´Z´ indicate outstations.

11 Displaying, changing and blocking junctions and monitors

The contents of a junction cannot be displayed or changed without the junction being present on the screen as either a connectable junction or a monitor junction.

11.1 Displaying a junctions contents

The monitor (<m>) command is used to continuously display the contents of a junction. If the contents of a junction is required without continuous monitoring, the cursor may be positioned over the required junction or junction number and the <?> key pressed. The junction contents will be displayed on the bottom prompt line and will remain there without being updated until a new command is given.

11.2 Changing the actual value

The actual contents of a junction can be changed by positioning the cursor over the required junction point or a junction monitor and pressing <c>. The program will prompt for a new actual value which, for an analogue junction, may be entered as up to 5 decimal digits with a leading -ve sign if required or as up to 4 hex digits followed by <h>.

11.3 Blocking and unblocking

The blocked contents of a junction can be changed by positioning the cursor over the required junction point, junction number or junction monitor block and pressing <b>. The program will prompt for a new blocked value which, for an analogue junction, may be entered as up to 5 decimal digits with a leading -ve sign if required or as up to 4 hex digits followed by <h>. Pressing <enter> only in response to the prompt will clear the blockage from that junction.

All junctions may be unblocked using the junction reset command (<r>) and the junction unblock command (<u>).

11.4 Test example

A simple example will illustrate this versatility.

> A section of the control program is checking several digital inputs being in an alarm state. If so, then an alarm siren should sound.

To test this program on-line, the output junction corresponding to the siren could be blocked to an off state to avoid unnecessary alarms. This junction is also displayed on the logic diagram drawn on the VDU screen. Combinations of the relevant digital inputs may then be

blocked into alarm states and the effect upon the output observed by watching the actual junction value corresponding to the siren on the screen.

All of this may be done without leaving the keyboard and screen and in confidence that any input can be simulated and checked whilst maintaining outputs, driving motors and valves for example, in a safe condition.

The ability to simultaneously observe the dynamic contents of up to 20 selected junctions is a very powerful debugging aid. When a page is displayed on the screen, the contents of one or more junctions can be displayed in real-time in their correct circuit locations; other junctions not on the current page display can also be observed.

12 Function Interpreter

The function interpreter task will normally run continuously. The linear function list is interpreted from the beginning until the end at a fixed frequency. The time interval between beginning each scan of the list is determined by the contents of a reserved junction (Analogue Junction 1); the contents of this junction gives the list period in units of 1/600 seconds. When this junction contains 0, the list will be executed every 0.1 seconds. When this junction is forced to 0, the list will not be executed.

If the list is still being interpreted when another list execution is demanded then the interpreter will begin interpreting the list from page 1 immediately after completing the present list scan in an attempt to catch up and synchronise again. If the contents of Analogue junction 1 does not allow enough time for the interpreter to get through all of the function list, a "catch-up counter" is updated to control how many complete list executions must be performed before the list is synchronised again.

The system is therefore tolerant of occasional long list execution times without internal clocks or counters becoming incorrect. The building program allows the catch-up counter and the actual list execution time to be

displayed so that the designer can measure quite accurately how heavily loaded the computer is.

The function interpreter task is one of many tasks which operate under a prioritised task scheduler to be described later. If the list is being interpreted too often, at no time will any other task actually stop. The effect of allowing more time to be given to the function interpreter task (which must have the highest priority) is simply to slow down other tasks such as the builder program and junction monitor updates.

The function interpreter itself is a very simple task. A pointer is maintained which points to the next item in the list to be interpreted. The interpreter simply reads the contents of the list pointed to by this pointer and uses the value obtained as an index into a list of function routine addresses. It is the responsibility of each function to use the information held after the function code in the list and return back to the interpreter once the function is complete. The coding of the functions is very important and must follow the following rules:-

1. Return with the list pointer to the next function to be interpreted, under all circumstances.
2. Execute as fast as possible.
3. Be tolerant to all combinations of input values and always produce predictable results.
4. Be re-entrant; this means that internal values which must be maintained between different executions of a

function in a particular position in the list, cannot be held in simple variables. Junctions will have to be specified for such value storage.

These rules, once understood, do not create serious programming problems since the function is a small self-contained module which can be easily tested. Once developed a function need not be modified again and adds to the list of available functions for all future development. This results in much more reliable software since for all applications, functions can be used but not modified without the use of other development equipment.

12.1 Sub-lists

With many industrial control situations, the major part of the control will tolerate a delay of say 0.2 seconds maximum between an input change causing an output to change. This means that the bulk of the process control circuit can exist in the main list of functions executing at the default period of 0.1 seconds as previously described. Sometimes however, a faster response time of several milliseconds is required to one or more particular inputs. To accommodate this, a group of reserved pages within the function list are executed by the interpreter at a programmable rate which may be as frequent as 100Hz or greater. Obviously the number of functions that can be included within these pages has to be limited at high

execution rates so that the main function list can be fully executed in the given time interval. Fortunately, in practice, many process control applications do not require excessive quantities of high speed control logic.

12.1.1 Sub-list-1 (last page)

When a non zero number is put into analogue junction 2, a task is enabled which will commence execution of the list from the start of the last page at an intervals determined by the contents of analogue junction 2.

When analogue junction 2 contains 0 or is blocked to 0, the last page sub-list is disabled and will not operate.

This sub-list is useful when some operations need to be carried out more often than the period of execution of the main function list (e.g. counting pulses). Care must be exercised to prevent this sub-list from using an excessive amount of time; this sub-list must only contain enough functions to do the required operation and must not be executed more often than is necessary.

To prevent the main list also executing the last page, a jump function may be added at the end of the penultimate page which jumps out of the function list (Jump to last page + 1).

The last page sub-list has the highest priority of the sub-lists and consequently is most suited to the fastest execution time required in a system.

12.1.2 Sub-list-2 (last page - 1)

When a number is put into analogue junction 3, a task is enabled which will commence execution of the list from the start of the penultimate page at an intervals determined by the contents of analogue junction 3. The operation and requirements of this sub-list are the same as the last page sub-list.

To prevent this task overunning into the last page, a Jump to the (last page + 1) function can be added at the end of the (last page - 2).

This sub-list has a lower priority than the last page sub-list and is suited to tasks such as analogue input´s with slow conversion times.

13 Multi-plant control

Often an application demands the control of several similar items of plant. One could simply design the logic for one item and repeat it for the others but this is inefficient for many reasons. If logic is repeated, errors can made more easily and changes are more difficult to implement.

Special purpose functions are provided which permit pages of logic to be called as subroutines; this is of little use however if the same data is being processed each time through the subroutine. Before calling a subroutine to control another similar item of plant, an offset function should be executed which will cause a programmable number to be added to every subsequent junction reference number until a new offset function is executed or the function list re-started. Each run through a subroutine can be organised so that a different block of junctions is accessed.

A practical application would be the control of 20 conveyors; the design can be implemented and tested for one item with the control functions terminated by a "Return" function. Twenty subroutine functions, each preceded by offset functions, will duplicate the control for each conveyor using data unique to each conveyor.

13.1 Using offset functions

Offset functions when used with Jump and Subroutine functions enable the programmer to easily control more than one similar item of plant without duplicating the control functions used for one item of plant.

When the function list begins execution each time, there are no offsets on any junction type, i.e. all junction reference numbers address the specified junction. When an offset function is encountered, the contents of the junction attached to that function is added to every junction reference number of the junction type specified by the offset function chosen. This enables a page of functions which constitute a subroutine to be called several times with a new offset function between each call to the subroutine. With careful choice of offsets, each execution of the subroutine will operate with different input´s and output´s; this enables many similar processes to be controlled using one subroutine with as many subroutine calls and offsets as required.

It is necessary to define a block of junctions for each item of plant and to make careful note of these areas. The offset functions are very powerful programming aids but great care must be taken when they are used. It is a good practice to enter the offset functions that restore the offsets to zero before entering the other offset functions and subroutines. This may prevent accidental execution of

functions with unwanted offsets.

When an offset is added to a junction reference number, the resultant junction number is not checked for limits. This is desirable to optimise for speed but it is also possible to write to a junction that is outside the memory assigned for that junction type. This would have possibly dangerous consequences and great care must be exercised when using offsets.

Offset functions present in the main function list will have no effect upon the operation of the sub-lists. The sub-lists may incorporate offset functions if necessary. As with the main list, sub-lists always begin execution with all offsets at zero.

When designing a control system that includes similar blocks of control functions (e.g. the control of 4 vessels or 20 conveyors), the design should as far as possible be implemented and tested for one item with the control functions entered as a subroutine. Once the junction usage is known for this subroutine, a safety margin can be added and offset functions included along with more subroutine calls to control the additional items.

13.2 Designing with Offsets

When an offset function is executed the functions following do not access the junctions specified on the screen display. To overcome the considerable monitoring and debug

problems created by this mismatch, an offset command <o> is provided to compensate for the difference between displayed junction contents and actual junction contents being referenced by the functions.

Pressing <o> when the main command prompt is displayed will result in two questions being asked. The first requests an offset to be used when displaying digital junctions and the second questions requests the offset for analogue junctions. Pressing <0> <CR> or <CR> only will clear the offset facility; pressing <Esc> will leave the offset value unchanged. The current offsets used within the display of a page are indicated on the main command prompt line.

A non zero offset results in that offset number being added to every junction number of that type on the screen and the result used as the junction number to display.

For example, if an offset function producing a digital offset of 500 is executed before the function being displayed on a screen page, then all digital junctions being displayed will not normally correspond to those actually being referenced. A digital junction 10 connected to a function results in junction 510 being accessed by that function. If junction 10 is displayed using a monitor or the <?> command then the contents of junction 10 are displayed, not the contents of 510. If the <o> command is used to set a digital page offset of 500 then the junction contents being displayed will be 510 even though the screen still shows a junction number 10.

If a sequence of pages operate as a subroutine which is called several times with different offsets set by offset functions then each execution will operate upon a different group of junctions. This is like a procedure using local variables but with the facility to still pass and return any value(s) to other parts of the function list. The <o> command enables the operator to examine each execution of this procedure or subroutine without changing the monitor displays or calculating the actual junctions being referenced.

All actions upon junctions affected by a non-zero screen offset will take the offset into account. This includes all of the commands to change and block an individual junction value but not the commands to list (and unblock) all junctions or the junction cross reference command.

The offset command affects only the monitors on a page which display values in the oustation being displayed. Monitors of other junctions not in the current outstation will be unaffected.

It is possible to monitor a junction which, combined with an offset, refers to a junction outside of the allowed junction data base. This will be indicated by a flashing actual value monitor or an off-line message.

When a page is saved, the offset status is also saved and will be automatically restored on reloading.

14 Circuit design

The design of programs using functions is much closer to hardware design than conventional software design. There are several characteristics of this system which must be understood before successful designs can be produced.

1. All functions are periodically executed unless specifically skipped using Jump type functions.
2. Functions are executed in page order and in numerical order within a page.
3. All junctions contain zero when the system is started and must be loaded with a value if a constant is required.

Because this system is so different from other languages, several example designs will be explained as illustrations of the techniques developed. These are of course only trivial designs, a more comprehensive practical design is given in Appendix C.

14.1 Oscillators and Timers

There are no specific timer functions in the system. A timer can be produced by connecting the output from an oscillator into a counter in much the same way that

oscillators drive digital divider circuits for hardware design.

Oscillators can be easily produced giving a variety of frequencies which are directly related to the execution rate of the function list. One of the simplest oscillators is an inverter, (a buffer function with its output inverted) with its input connected to the same junction as its output. This is shown on page 4 of appendix C. The period of oscillation for such a circuit is twice the function list execution period.

A simple variable frequency oscillator can be produced using the digital integrator function connected as shown on page 4 of appendix C. The period of oscillation for this circuit is twice the function list execution period multiplied by the integration constant.

A typical control system would have several junctions whose contents oscillated at frequencies such as 0.5Hz and 1Hz. These junctions can be joined to the inputs of counter functions to create any number of timers. Timers may be cascaded to form timer circuits having any duration in seconds, minutes, hours, days or years. Page 5 of appendix C shows 3 cascaded counters forming a 24-hour real-time clock with reset facility.

14.2 Alarm Detector

Alarm conditions within a process control system are very

important. Many different signals may cause an alarm condition to occur and these conditions must usually be scanned on a regular basis alongside all other control activities. This type of requirement can be systematically handled in a straightforward way using the scanner function. An example alarm detect circuit is shown on page 21 in appendix C which also includes protection against transient alarm conditions if necessary. The scanner function is an example of a sophisticated function mainly designed for a single application. It allows any number of digital junctions to be scanned to produce an alarm code if one is set. The disable facility allows alarm states to be accepted one at a time and any further alarm codes will then appear. Obviously there is a built in priority in that junctions with lower numbers will be detected first and if a more urgent alarm condition is detected, it will be reported in preference to an existing alarm code.

The exclusive OR functions used to produce the alarm junctions themselves are being used as digital comparators. In this example, alarms are produced if the feedback signals from valves do not agree with control signals being sent to the valves both in their open and closed position.

The timer circuit prevents alarms being generated whilst valves are moving.

14.3 Set-Point Calculator

Many processes require ramp functions to be generated. An example may be to produce a steadily rising temperature from an initial valve to a final valve over a period of minutes. This type of calculation often arose in the Authors´ experience with Malting Kiln controllers where temperature set-point ramps and dwells were being generated over long periods to ultimately control gas burners.

Page 22 of appendix C shows a circuit which will produce a set-point output for positive or negative ramps or dwells. The time left input would come from a programmable down counter which is loaded with the stage duration at the start of a stage in the process. Any parameter may be changed at any time and the correct set-point is automatically calculated.

Such a circuit could be entered in a single page and then executed as a subroutine. The use of subroutines and offset functions allows the calculator to be used for simultaneous control of many concurrent ramp and dwell processes.

14.4 Inputs and Outputs

Programs can be tested successfully without actual inputs and outputs by blocking input junctions and monitoring output junctions. At this stage there need be no

consideration given to how these signals will actually be brought into or out of the computer.

Interfacing to the "outside world" is accomplished by input and output functions which have to be specific to the actual input/output hardware used. The type of functions necessary include such facilities as 16 bit digital input and output and analogue input. Examples of these functions have been developed specifically for the IBM PC/AT microcomputer and bus interface cards but of all of the functions in the system, these would have to be changed for use with practical isolated inputs and outputs.

All inputs and outputs are via analogue junctions. Most digital inputs and outputs are however required as single bit quantities in digital junctions. Inputs placed into an analogue junctions by input functions can be divided into sixteen digital junctions using the data divider function. Sixteen digital junctions can be assembled into one analogue junction using the logic assembler function. This composite analogue value can then be output using the analogue output function. Examples of input and output can be found in Appendix C pages 3 and 40.

14.5 Distributed Process Control

The initial phase of the project developed a single computer language and development environment suitable for large and small control problems. Many large control

systems however, require intelligent outstations joined by telemetry links to a masterstation. This type of system can be very difficult to design due to different and often smaller computers being used at the outstation compared with the central station. There is rarely a unified development approach, and this can make such systems expensive to develop and install.

Because the heart of the new language is an interpreter, the function list is computer independent. A system in the form of a function list could be developed on one computer and then transported without change to another totally different computer. The only requirement of the target computer would be a suitable real-time executive and function interpreter. Once these have been developed, applications in the form of function lists can be transported from computer to computer without change.

This portability aspect of the language is a very powerful feature that will allow future improvements in computers without discarding existing designs and techniques.

14.6 Remote development

A typical large process control system will consist of a relatively powerful central computer or master station and several smaller intelligent outstations scattered around the factory or plant. Intelligent outstations imply that

they function as controllers for local inputs and outputs but obtain control information in the form of set-points and remote variables over the telemetry link.

The most convenient way to develop such a system is to give the masterstation facilities for remote program development such that the masterstation user can simultaneously develop and test programs for the masterstation and any outstations. These facilities have been included as features of the new language.

Communications between the masterstation and the outstation fall into two categories:-

1. Regular information exchange for control purposes.
2. Requested information exchange for development purposes.

Since the status of a system is determined by the contents of junctions, control information can be exchanged by exchanging junction contents between the masterstation and the outstation. This would take the form of allocating a group of analogue junctions in the outstation and transmitting their contents to the masterstation on a regular basis. Similarly, a group of analogue junctions in the masterstation would be transmitted to the outstation. In this way, a gateway or window is produced between the masterstation and the outstation. Information can be sent and received by simply placing values into the junctions assigned to the gateway; a separate telemetry task would handle the actual communications. Suitable telemetry

systems exist operating upon continuous or change of state principles, but this has no direct effect upon the principles and applications of junction exchange.

During remote program development, many additional details need to be communicated between the masterstation and the outstation. This information is not strictly required in real-time although delays would make development tedious. All of the development facilities can be broken down into the following fundamental requirements of the function list and junction data base.

- System information

    * number of pages in the list
    * number of free bytes in the function list
    * number of analogue junctions
    * number of digital junctions

- Timer information

    * main list current execution time
    * main list maximum execution time
    * main list catch-up counter
    * sub-list 1 catch-up counter
    * sub-list 2 catch-up counter

- Read from specified digital junction
- Read from specified analogue junction
- Write to specified digital junction
- Write to specified analogue junction

- Search specified page in function list for specified junction usage.
- Replace existing page of functions with new functions held in a buffer.

These requirements are divided into eight high level request functions which are requested by the building program regardless of whether development is of the masterstation or the outstation. At all times the masterstation has control over information request, the outstation simply accepts data or replies to data requests.

14.7 Command Breakdown

A control system must be designed to include outstations when the software is initially configured. If outstations are present then many of the builder program commands ask additional questions to request the desired outstation to operate upon. Apart from this additional question, there is no difference between local development and remote outstation development.

Originally, the action of updating on-screen monitors was handled by junction contents requests made by the building program. This was built into the building program as a non time sliced task which allowed for continuous update during all commands. When the outstation facilities were added, the time delay added due to junction contents

requests being made over a communications link slowed down all builder facilities. To overcome this, monitor updates in both the main screen, and the window screen were still handled by the builder, but the junction contents were obtained by a separate task on the main task executive. This allowed the builder to continue at full speed despite communications delays.

Because of the presence of communications links, other problems could be encountered. The most serious problem occurs when there is a break in communications when for any reason an outstation is taken off line. All of the communications commands check for this and report when an off line condition occurs. This will result in the off line condition being reported in the building program dialogue, and will cause a change in colour of the relevant monitor display.

It was central to the design concepts of remote program development that the system should be uniform. To this end, no limits have been placed upon which junction in a system can be monitored or changed even when displaying monitors from a different outstation. It is therefore possible to add a function into for example, outstation A, and simultaneously see the effects in outstation B, outstation C, and the masterstation.

15 Hardware

This project was envisaged to be applicable to a wide range of situations. Obviously, aspects such as performance, cost and hardware availability have to be considered, but most importantly, anything developed must be expandable from an inexpensive computer for simple simulation, to a full and powerful computer used to control a large control situation.

Fundamentally, the language is interpreted, which requires a more powerful computer than otherwise to achieve a satisfactory speed of operation. The complete system is also very large in memory requirements (well over 100K bytes of code), which precludes easy implementation on anything smaller than a 16-bit computer.

The requirements for colour block graphics in a computer which may require future upgrading, pointed towards using a personal computer within a range of compatible machines. The obvious choice was the IBM PC which could later be upgraded to machines based upon the 80286 and 80386 microprocessors if more speed was required. The development was all carried out using an 8086 based IBM compatible computer which was also used as the target for the developed software. This configuration allows easy upgrade to more powerful computers and computers with memory management for large industrial

applications.

Great efforts were made to use the standard colour graphics IBM PC type configuration without any form of hardware modification. This ensures that the greatest barrier to program portability is as small as possible.

The minimum configuration necessary is as follows:-

- IBM PC/AT or compatible
- Colour graphics adaptor and display
- IBM/Epson compatible parallel printer
- PCDOS or MSDOS operating system

To make use of standard software running alongside the new language required that the PCDOS operating system and IBM PC BIOS remain unmodified. To make the language system run concurrently alongside an unmodified PCDOS based computer proved to be one of the most difficult aspects of the entire project.

16 Real-time executive

The heart of the software is a prioritised time slice real-time multi tasking executive. The requirements of such an executive can be summarised as follows.

1. "Concurrent" execution of a number of tasks.
2. One or two tasks operating system dependent.
3. Ability to allocate different percentages of the CPU time to each task.
4. Ability to dynamically deactivate tasks and change their CPU percentage use.
5. Ability to relinquish and exit a task under the control of the task.
6. The executive must take the initiative to switch tasks if the tasks themselves do not relinquish or exit.
7. Handling of high speed tasks with frequencies of around 5ms. This enables functions to handle fast inputs without resorting to special drivers written in another language.

To give some indication of what this executive had to do, the following is a minimum list of tasks to be run:-

1. PCDOS operating system (as a task)
2. Building program
3. Function List (high priority)

4. Sub-lists (highest priority)
5. Monitor updates

If processing time is short, the list tasks must take priority since they are executing the control algorithms. An executive had to be developed which allows this priority without completely depriving the other tasks of all computer time.

16.1 Existing executives

Some real-time executives do exist which will do some of the things required. They usually do not provide the following features:-

1. Fine control of task execution time and priorities. Most executives operate on a fixed 50Hz task switching rate which is too slow for real-time process control.
2. Management of tasks which are not well behaved. That is, tasks may revector interrupts or access hardware directly. This is true for many of the current compilers, interpreters and application programs and would lock out an unwary executive.
3. Fast task switching. Executives sometimes provide many facilities which are rarely used; this slows down task switching and is exaggerated by the first item above.

Executives designed for 8 bit microcomputers are unsuitable in that their addressing range is insufficient

for sophisticated tasks. The two larger tasks require in excess of 100K bytes each and this will only rise in the future.

16.2 Executive operation.

The complete software system is divided into a series of modules or tasks. Each task is assumed to be running at the same time as the other tasks and requires its own stack.

The most significant problem however, was implementing the Building program and another operating system (PCDOS/MSDOS) as tasks. Both of these assume that they are the only tasks in the computer; PCDOS/MSDOS is a single tasking operating system and the Building program was designed using a compiler which assumed the presence of PCDOS/MSDOS.

There were two possible implementations:-

1. A simple "round robin" executive program can run through a table of tasks giving complete control to each task in turn. It is the responsibility of the task to return control back to the executive at regular intervals to enable other tasks to run.
2. An interrupt driven executive which relies upon a regular timer interrupt (say 600 times per second) to asynchronously break execution of one task and resume execution of the next.

The round robin executive is relatively easy to write and would be less susceptible to programming errors. Its success however relies upon every task relinquishing control on a regular basis back to the executive; this requirement would be impossible to satisfy when running existing single task PCDOS/MSDOS programs.

The interrupt driven executive has the considerable advantage of allowing single task programs to be run, without modification, in a multitasking environment. The penalty for this is a more complex executive and the provision of a separate high priority interrupt. Previous experience with a round robin type executive confirmed the desirability of an interrupt driven executive and so this was developed.

16.3 Executive implementation.

The following list itemises the implementation of the real time executive.

1. Each task has its own stack.
2. When a task is not running, the stack contains all of the processor registers and the address of the next instruction once the task resumes.
3. Several tables keep track of the tasks as follows:

    3.1. location of top of stack.

    3.2. Task maximum execution time before relinquish.

3.3. Task execution timer.

4. To initialise a task, its stack area must be loaded with initial register contents and the task start address.
5. To allow execution of a task, its Task execution timer must be loaded with the maximum execution time required.
6. To prevent task execution, the appropriate task execution timer is set to 0.

This implementation was aimed at supporting the language and its development tools. It was not intended to be a general purpose executive with extensive resource allocation, memory management and data access control.

16.4 Task Control

It is not normally necessary to remove a task altogether since they can be effectively eliminated by preventing their execution. This gives a very simple method of task switching without complex task manipulation.

The complexity of often installing new tasks is not required but the ability to exit a task to avoid time wasting when there is nothing more to do is very necessary. A task can prematurely relinquish control to the main executive by simply "faking" the timer interrupt using a subroutine call. This prevents time wasting when a

task is waiting for another event to occur.

If the task sets its own task execution timer to zero before relinquishing, it is effectively disabling itself.

Dynamic priorities can be allocated by manipulating the table of maximum execution times.

16.5 Executive Task Time Control Example

An interrupt rate of about 600Hz was chosen as a compromise between time wastage due to frequent task switching and the ability to handle one or two short duration tasks often.

The following example shows how task priority is handled without locking out less urgent tasks. Assuming a 500Hz interrupt rate for the executive, a total of 4 tasks and the following maximum execution time table:

```
TASK1    DW    100
TASK2    DW    2
TASK3    DW    10
TASK4    DW    10
```

Task 1 has the ability to run for 100/500 seconds or 200mS before control is removed. This will give approximately 100/122*100% of the CPU time to that task which gives it the highest priority. If Task 1 does not require this amount of processor time, it can relinquish back to the executive anytime before the 200mS is up.

Task 2 has the lowest priority and cannot run for more than 4 ms at a time.

Tasks 3 and 4 have equal priority and cannot run for more than 20mS at a time.

At an instant in time, the task execution timers may be as follows:

```
TASK1   DW      25
TASK2   DW      2
TASK3   DW      0
TASK4   DW      10
```

This shows that task 1 is active and is 25% of its way through its allotted 200mS time slot. Task 3 has been disabled and will not run until a non zero value is loaded into this table. Note that no task is ever locked out; less urgent tasks simply run slower when more urgent tasks require more time.

16.6 Integrating DOS

A disk operating system (PCDOS/MSDOS in this case) can be run as one task. If the DOS is not re-entrant, as is the case with MSDOS and PCDOS, then it is not possible to run the same DOS as more than one task to create a multitasking DOS.

This application requires 2 DOS tasks in addition to the 6 other non DOS tasks. The following two different approaches are possible:-

1. Two tasks running the same DOS but never concurrently.

Switching from one DOS task to the other occurs in a synchronous way which enables the complete graphics screen to be saved and a copy restored for the new DOS task. This enables two DOS programs to reside in memory and execution to be switched by a command from the keyboard.

2. Two tasks running the same DOS concurrently. This requires an additional screen and keyboard and additional DOS handling routines for these devices. During task switching, the vectors for the appropriate screen and keyboard driving routines are inserted into the operating system. This permits 2 users to access the same computer at the same time (multi-user) while other tasks are also running (multi-tasking).

A design decision was made very early on to run the Building program or the operating system alone but never both. To run both simultaneously would require re-writing the BIOS and operating system to be re-entrant as well as providing an additional keyboard and screen. This decision greatly simplified the problems and allowed the use of an unmodified operating system and a standard computer.

This decision can be defended by recognising that development of the function list using the Building program is a separate activity from developing the operator interface. The operator interface would be developed using existing languages and debugging tools which can run under PCDOS/MSDOS. Switching from this development to the

Building program was implemented by detecting the pressing of a "hot key" inside the keyboard interrupt handler (Control L was used for this purpose).

16.7 Problems with DOS integration

A standard version of MSDOS or PCDOS was used on a standard unmodified computer. This creates significant problems:

1. Conflict with the timer interrupt. This interrupt is already in use and had to be integrated and speeded up without changing real-time clocks and general operating system timers.
2. Re-vectoring of interrupts (in particular, the timer interrupt) by application programs running under DOS. Considerable effort was given to this since it is common practice for applications programs to revector interrupts for their own purposes and even resetting the interrupt rate for the timer. This of course will have disastrous consequences for the real-time executive.
3. Memory allocation. Loading and running programs under a single task operating system usually means the task assumes it has the entire memory of the computer at its disposal. This is not the case with multi-tasking systems.

17 Other Applications

The main use for this project is for real-time process control of actual industrial plant. This does not mean however that there are no other suitable areas for its use which fall mostly in the field of simulation.

17.1 Control Simulation

Because of the extensive ability to change inputs and monitor outputs, it is possible to set up and test controllers without actual inputs and outputs. It is also possible to simulate the operation of a particular process with for example, filter, pure time delay and arithmetic functions and attempt to control this process simulation with other functions, for example, a 3-term controller. Obviously the process simulator and the controller itself need not be linear and could easily interact with other programs running under the operating system task.

To assist in the development of functions, a full graphics plotting program was designed to allow the simultaneous display of any junction contents against time. This program effectively acted as a 3 pen chart recorder with facilities for changing scales, junctions plotted and resolution. A brief description of the program

is given later.

The use of plant simulation and the plotting program means that an unmodified IBM PC computer can be used to test control algorithms.

17.2 Logic Simulation

Since many functions operate in an analogous way to actual logic devices, networks of such functions could be used to simulate real digital circuits. Obviously such a simulation is limited to the functions available, but many circuits can still be designed and the results either monitored or plotted using the plotting program described later. To illustrate the types of circuits possible, the following list outlines some of the relevant functions:-

1. AND and OR functions (also simulates NOT,NAND,NOR)
2. Flip-Flop function
3. Latch function
4. Comparators
5. Programmable up/down counters
6. Decoder

This type of simulation does not take into account practical electronics problems such as power supplies, signal loading or propagation delay, but could be a useful method for checking simple circuits.

18 Function Characteristics

Functions are defined as a header byte indicating the particular function followed by a list of 2-byte junction addresses associated with that function. When a function executes, the function routine itself reads the addresses of each of its junctions and uses these to point to data for use as inputs and to point to junctions to store results. A function is therefore a well defined program module which processes data and exits pointing to the next function in the list to be executed.

Functions can use any mixture of two types of junction; analogue or digital. Once a function has been designed, the type of each junction is fixed along with its use. Function data tables are used to inform the building program of the shape of a function, which junctions are digital, which are analogue and where on the shape they are positioned.

The currently developed functions were selected to provide sufficient facilities for most process control tasks without providing specialist functions which could be easily generated by combining a few simpler functions. Throughout the design, the functions were made as flexible as possible so that a particular function could have many uses. The AND function for example could be used as a NAND, OR, NOR or any permutation of inverted inputs and

outputs.

All functions contained extensive error detection. All input data is checked for valid range and all inputs produce predictable outputs.

The following sections describe the operation and use of the functions developed so far. Appendix B includes a diagram for each function as it is drawn on the VDU screen.

18.1 Digital Logic Functions

The most basic functions required for process control tasks are those which process digital inputs and produce outputs which depend upon logical combinations of these inputs. The **AND function** is provided in 2 or 4 input form; any other number of inputs can be produced by chaining together several AND functions and connecting unused inputs to logic one.

Because any connection to a digital junction can be inverted, the AND function can be configured as an OR function by inverting all inputs and outputs. This has the conceptual problem of not "looking like" an OR to most people so an **OR function** was also provided in a 2 or 4 input form.

A **buffer function** was included to allow junctions to be copied with or without inversion into another junction. Again, a 2 input AND could have been used for this but it

was such a common requirement that the clarity of a dedicated buffer function was deemed worthwhile.

A 2 input **exclusive OR function** was included since it provided a compact way of comparing two digital values. This is very useful when, for example, feedback signals from a valve positioner must be compared with the control output to check for correct valve operation. The most common use for this function is therefore in alarm detection circuits.

The **flip-flop function** is a simple single bit memory device with set and reset inputs. It operates in much the same way as an SR flip-flop I.C. with the exception that the undefined state when both the set and reset inputs are active (logic 1), causes the output to be reset to logic 0. As with its hardware equivalent, the output remains unchanged when both the set and reset are at logic 0.

The **latch function** is a simple digital switching function. The data input is transferred to the output only when the control input is active otherwise nothing is done. Since a junction retains the value last written into it, this is another form of memory device.

18.2 Miscellaneous Digital Functions

This group of functions basically process digital information but in a more complex way than the simple digital logic group. They all include one or more analogue

junctions to either control digital data or produce digital data.

The **integrator function** can be defined as follows; if the input is not the same as the output for (integrator constant) scans of the function then the input is transferred to the output. The actual count junction contains the running count value which is decremented to zero.

In use, the integrator function would normally have a constant loaded into the integrator count junction and the actual count junction would be otherwise unused. The effective action is to delay the change in the output until the input is consistently present for the integrator constant number of times through the function. This provides a very useful filter for noisy digital inputs, the degree of filtering being set by the contents of the integrator constant junction.

The **scanner function** scans down (max) consecutive digital junctions starting at the junction connected to the input. The output is reset to 0 and the number of input detected junction is reset to 0 unless otherwise stated as follows:

1. If an input junction contains zero then the corresponding disable junction is set to zero; scanning continues.
2. If an input junction contains 1 and the corresponding disable junction contains 0 and the accept input

contains 1 then the corresponding disable junction is set to 1 and scanning continues.

3. If an input junction contains 1 and the corresponding disable junction contains 0 and the accept input contains 0 then the output is set to 1 and the input junction number relative to the first input junction is put into the number of input detected junction. No more inputs are scanned.
4. If an input junction contains 1 and the corresponding disable junction contains 1 then scanning continues.

This function is primarily provided to scan a consecutive list of digital alarm junctions and report when one is in an alarm condition. The input junctions must be ordered in alarm priority; the number of input detected junction is set to a non-zero value once an alarm condition, which has not been accepted, has been found. This value can be passed to an alarm display program to produce a message to an operator.

A series of alarm conditions can be handled by this function. The most urgent, non-accepted alarm will be reported; any less urgent alarms being reported after accepting the current alarm.

The only input and output functions available transfer into and out of analogue junctions. Digital inputs and outputs are usually collected into groups of 8 or 16 by the hardware and must be ultimately input and output in these groups by the software. To read digital inputs into

digital junctions requires two separate functions. The first is an input function to bring normally 16 digital inputs into one analogue junction. The second, a **data divider** function, divides the binary contents of the input junction into 16 consecutive digital output junctions. The first output junction contains the least significant bit of the input junction.

In a similar way, digital outputs are generated by the control program as separate digital junctions. The **logic assembler function** combines the contents of 16 consecutive digital junctions into the analogue output junction. The least significant bit of the output junction comes from the first input junction. Once combined, an output function may be used to drive the actual hardware outputs.

The final function in this group is the **decoder function.** This takes an analogue junction and uses its value to set one out of a consecutive list of digital junctions. To be precise, limit consecutive outputs are reset to 0 except for the one corresponding to the number held in the input junction. If the input is greater than the contents of the limit junction then there will be no change in any outputs.

18.3 Analogue Functions

This is the largest group of functions and contains some of the most sophisticated functions. They are characterised

by handling analogue junctions as inputs but most will still use digital junctions to control the operation of the function and produce logical outputs.

Three **compare functions** are provided; these test for

(i) A equals B,

(ii) A greater than B,

(iii) A greater than or equal to B.

By inverting the outputs of these three functions, the other compare requirements can be provided:-

(iv) A not equal to B,

(v) A less than or equal to B,

(vi) A less than B.

As always, an output of logic 1 is regarded as true.

Three **counter functions** are also provided. These are extremely versatile functions which may be used as timers, counters or waveform generators. The counters differ in the test used to compare the elapsed count junction contents with the set count junction contents; this test can be one of the three compare options as described earlier for the compare functions.

Before examining the usefulness of the functions, their operation must first of all be defined. All of the three counters will increase the elapsed count junction value, if the up input junction contains 1, or decrease, if the up input junction contains 0, by one each time a positive transition (0 to 1) occurs on the input. The last-input-store is used to determine when a positive

transition occurs.

If the reset input is set to 1 then if the up junction contains 1 the elapsed count is reset to 0 or if the up junction contains 0 the elapsed count is loaded with the contents of set count.

If the counter is counting up then the output will be 0 unless elapsed count is equal to the set count when it will be 1.

If the counter is counting down then the output will be 0 unless the elapsed count is set to zero.

At this point, assuming the counter is counting up, one of the three compare tests is performed and if the comparison is true, the output is set to 1, otherwise 0. If the counter is counting down, then the comparison is between the elapsed count and zero which will set the output as true otherwise the output is reset to 0.

If a fixed frequency oscillator is connected to the input of a counter, a timer function is created. If the output is connected back to the same junction as the reset input, the timer will automatically reload and begin again. If the counter/timer is allowed to free run, the elapsed count junction value will "end stop" at either -32768 if counting down or +32767 if counting up.

The counter functions may be used as general purpose edge triggered programmable up/down counter/timers, one-shot timers, ramp generators or sequence controllers. Obviously, several counter functions may be cascaded to produce a multistage counter/timer counting to almost any

value in any chosen units. For example, hours:minutes, day:month:year, feet:inches and many others.

The **switch function** is a straightforward analogue equivalent to the digital latch function; as such it shares the same characteristics.

The **multiplexer function** is basically a data selector. The contents of the select input determines which one of the inputs will be transferred to the output. If the contents of the select junction equals zero or is greater than 10 then the output is not changed.

This function is valuable in processes which step through many stages with different control parameters for each stage. The stage number (from a stage counter) is connected to the select input to give the desired data on the output of the multiplexer. Multiplexers can be combined to give 20, 30 or even more ways by adding functions which subtract 10 between one select input and the next.

The **demultiplexer function** is the opposite to the multiplexer. The contents of the input junction are copied into the output junction specified by the contents of the count junction. A count value of 1 will transfer the input to the output junction defined in the function; a count value of n will transfer the input to the output junction number + (n - 1).

No transfer will occur if the count junction contains 0 or greater than the contents of the limit junction.

The **PID Controller function** is a complete three term

digital controller function. It would normally be used with a subtract function on its front end to calculate the error input value from an actual value and a set point value. The facilities include differential, integral and proportional gain inputs as well as a dead-band control.

The control algorithm uses the well recognised approximations to differentiation and integration in the discrete time domain. The equation realised in the function is:-

$$dV = P[(E_n - E_{n-1}) + IE_n + D((E_n - E_{n-1}) - E_{n-1} + E_{n-2})]$$

Where
- dV = change in output value
- P = proportional gain
- I = integral
- D = differential
- E_n = present error
- E_{n-1} = previous error
- E_{n-2} = previous to previous error

In order to allow more precise control of the P,I and D terms, these are stored in analogue junctions and are assumed to be to 2 decimal places. Using this scaling and rearranging the equation to simplify the mathematics produces the form of the equation actually implemented in assembly language.

$$V_n = V_{n-1} + P/100[D*(E_n - 2*E_{n-1} + E_{n-2}) + (100 + I)E_n - 100E_{n-1}]$$

where
- V_n = new positional output
- V_{n-1} = previous positional output

The ranges of the various values associated with the three

term controller function are as follows:

Error input	+/- 32767 range
Positional output	+/- 32767 range
Proportional gain	+/- 327.67
integral	+/- 327.67 secs/list rate
differential	+/- 1 / 327.67 secs/list rate
dead band	+/- 32767 in units of input

Differential time constant = D * dT / 100 Secs,

Integral time constant = (dT / I) * 100 Secs

Where dT = list period.

list rate = the number of times the function is executed per second.

The error input would normally come from a subtract function which calculates (Set_point - Measured_input). Non-linear controllers can be generated by operating upon this error term before it is passed into the PID function.

If the dead band value is greater than the error input, an error value of zero is used by the PID function. If this is not the case, the dead band value is subtracted from the error magnitude to produce the error used in the calculation.

The outputs labelled -1, -2 and f are used for temporary storage by the function; they are not normally connected to any other function. The outputs -1 and -2 hold the previous error and the error before that respectively. The output "f" holds a signed fractional output after correcting for rounding in the positional

output.

The output and all internal calculations are limited upon overload. The output saturates at about +/- 32767 and no integral windup is produced.

An incremental type output can be produced by loading zero into the PID output and the "f" output immediately before executing the PID function. A 2-term controller can be formed by setting the differential gain to zero.

The **filter function** provides spike rejection and a first order digital filter on an analogue input value. The first operation is to apply the following digital filter algorithm to the new input value.

$$L_n = ((T/10000) * L_{n-1}) + ((1 - T/10000) * I)$$

where I = input value

Ln = new last output value

Ln-1 = previous last output value

T = "time constant" stored to 4 decimal places

The actual time constant is related to the value T in the time constant junction by

$$\text{Time constant} = \frac{(T/10000) * dt}{1 - (T/10000)} \quad \text{seconds}$$

where dt is the function list execution period in seconds.

Rather than the new value Ln being directly transferred to the output junction, a simple spike rejection algorithm is applied. The value Ln is rounded and copied into the output junction only if the absolute difference between the new value and the last output is less than the contents of

limit the junction.

If the limit value is less than or equal to zero then the above spike rejection algorithm is not performed and Ln is rounded and copied directly into the output junction.

If a positive limit value is used then the output will not be updated until the filtered value falls within a band around the existing output; the width of this band being +/- limit.

To preserve accuracy, the value of Ln is calculated to 32 bit precision and is saved to that precision between function executions in the L.S. and M.S. last output junctions attached to the function.

An improved algorithm for spike rejection and filter operation has been investigated and tested. This has been described in a paper by the author.

The **delay function** produces a pure time delay between its input junction and its output junction. The time delay in units of the list period is determined by the contents of the junction connected to the T input. The input junction is the first of (T) + 1 consecutive analogue junctions which are used to store the input samples before being fed to the output. The sequence of (T) + 1 junctions starting at the input junction is in effect a tapped delay line.

If (T) is greater than 100, the function will have no effect and the output is not changed. If (T) is less than or equal to zero, the output is set to the input with no delay.

18.4 Maths Functions

A few mathematical operations are commonly required in process control applications but many more may be required on rare occasions. The fundamental mathematical operation such as add, subtract, multiply and divide have been provided as functions along with conversion between binary and decimal. A general purpose define function is provided for less common calculations. In general, most mathematical operations use signed 2 byte integer values which limit accuracy to between +32767 and -32768. All possible overflow and underflow conditions are trapped by the functions and result in outputs limiting to +32767 or -32768 rather than an unpredictable output value.

The **add function** and the **subtract function** are straightforward signed arithmetic calculators whose outputs are limited to +32767 or -32768 upon overflow or underflow.

The **abs subtract function** calculates the difference between the two input values as well as producing a sign indicator into a digital junction. The -ve result junction will contain logic 1 if input B is greater than the value of input A.

The **multiply function** and the **divide function** differ from other maths functions in that they can handle numbers having 32 bit precision. The multiply function produces a

32 bit result and the divide function will take this 32 bit result as one of its inputs. This expansion to 32 bit precision allows accurate scaling of values to take place by first multiplying by a constant and then dividing by another constant. Without this, scaling of the form shown below could be very inaccurate.

output = input * X/Y

Because analogue inputs often have to be scaled by non-integer constants, the ability to retain the intermediate calculated value of a scaling operation to its full precision minimises any errors.

If only 16 bit numbers are required, the most significant output of the multiply function can be ignored and the most significant input A to the divide function can be set to 0.

The divide function is more complex in its operation than the multiply function. Signed division is performed of a 32-bit number A by a 16-bit number B giving a 16-bit result. Division by 0 or a result larger than a signed integer will give a result and remainder of either 32767 or -32768.

The remainder ALWAYS has the same sign as A. This condition will require consideration when a remainder is generated since there are mathematically 2 correct result/remainder combinations possible.

There is often a requirement to handle Binary Coded Decimal (BCD) numbers particularly with respect to inputs

and outputs. Some input devices present their values in parallel BCD form which will require the **dec to bin function** to convert a 4 digit packed BCD input to a binary output suitable for other analogue functions

The complementary **bin to dec function** will convert a binary number into a packed 4 digit BCD number in its output junction. Since this function would commonly be used to interface to a 7 segment LED type display, a method for leading zero suppression was included. The input is converted to Binary-Coded-Decimal packed into the output junction as 4 digits. The contents of the leading-zero junction is inserted in place of leading zero´s in the output.

An -ve input is converted to +ve. An input greater than 9999 is still converted but the most significant digit is lost.

The **define function** provides provides one of a miscellaneous set of arithmetic conversions which is specified by the contents of the function-type junction. The function types may be expanded but include:

function type	function	input range	output range
0	no action		
1	SIN	+/- 3.100 rad	+/- 1.0000
2	COS	+/- 3.100 rad	+/- 1.0000
3	TAN	0 - 1.560 rad	+/- 320.00
4	ARC TAN	+/- 320.00	0 - 6.300 rad
5	natural LOG	1 - 32000	0 - 4.500
6	exponent	0 - 4.500	1 - 32000
7	square root	0 - 32000	0 - 180.00
8	read memory	0 - 65535	0 - 65535

18.5 List Control Functions

Normally all functions in the function list are executed in order on a periodic basis. It is possible, and sometimes desirable, to break this linear operation and conditionally skip some functions, or execute a group of functions as a subroutine.

The **jump forward function** takes 2 parameters, a digital control input and a page number input. If the control junction contains zero or the contents of page-number is less than or equal to the current page or greater than the maximum number of pages + 1 then no action is taken by this function.

If the control junction contains 1 then the next function to be executed will be the first function in the specified page.

If the control junction contains 1 and the contents of

the page-number junction equals the maximum number of pages + 1 then the current execution of the main list is terminated.

The **subroutine function** may be used in conjunction with the return function to execute the same group of functions several times. If the control junction contains zero or the contents of page-number is less than or equal to the current page or greater than the maximum number of pages or subroutines are already nested 10 deep then no action is taken by this function.

If the control junction contains 1 then the next function to be executed will be the first function in the specified page.

The **return function** has only one control input and will only have an effect within a subroutine. If the control junction contains zero or if there was no previous subroutine function which has not been returned then no action is taken by this function.

If the control junction contains 1 then the next function to be executed will be the function just after the last subroutine function executed.

The **digital offset function** and the **analogue offset function** are primarily used for multi-plant control as described in section 13. The contents of the offset-value junction will be added to all digital/analogue junction reference numbers of every function after this function until either another digital offset function is executed or the main list terminates.

18.6 Input/Output Functions

Only the basic hardware input and output functions have been developed. The input functions enable the input from a specified 8 bit port (**1 byte input function**) and the input from a specified 16 bit port (**2 byte input function**). The **1 byte output function** puts the least significant 8-bits of the value junction into the specified port, and the **2 byte output function** puts the entire 16-bits of the value analogue junction into the specified port.

Combinations of these 4 functions allow most inputs and outputs to be handled. Special input/output requirements, for example, to handle a multichannel analogue input system, would require dedicated driver functions to be written. This is the area where the software is very dependent upon the hardware being used.

Two other functions exist within this group. The **constant function** puts a signed decimal or hex number into an analogue junction. This function is unusual in that it is the only function to contain data as part of the function definition. Constant values are created by placing a number into a junction by the constant function; this junction can then be used as a constant by other functions.

The **BASIC array function** is specifically designed to

directly access arrays of numbers set up within a concurrently executing program running under the Microsoft Basic Interpreter. This function is able to access a specified single dimension integer array defined in the Basic interpreter program. A Basic program must dimension suitable arrays; no action is taken by this function if suitable arrays have not been defined.

An array number of n (less than 256) will access the n´th integer array to be dimensioned in the Basic program. The contents of (entry) element will be placed in the output.

If the write control input is set to 1, the input is put into the array and also placed into the output.

If the array number is greater than 255 then the contents of the array number junction is considered to contain the address of an array in the data segment. This allows data arrays to be manipulated by the functions without access to a Basic program.

19 User Interface

The interface between the computer and the user exists at two distinct levels. The `programmers´ view of the control system is provided by the building program which allows any part of the control system to be monitored and changed. The user or operators view of the control system must be at a much higher level without the detailed low level abilities provided by the building program. This operator interface must reflect the application and provide facilities such as parameter display and change using parameters familiar to the operator.

Although this project does not attempt to develop an operator interface, it must provide facilities to allow another program to request and modify parameters held in junctions. Partly to test this ability and to provide a convenient method for testing functions and systems, a multi-trace plotting program was developed to run as an operator interface.

The plot program provides an on-screen, 3 parameter plot of value against time in much the same way as a conventional hardware plotter. Any combination of 3 analogue or digital junctions can be plotted simultaneously in real-time with control over the value and the time axis.

The plot program has the following features:-

1. Display of up to 3 junction values using different colours for each plot.
2. Display using the 4 colour medium resolution screen or the 2 colour high resolution screen.
3. The X span can be set in units of the function list execution period.
4. The Y axis upper and lower limits can be set for analogue junction display.
5. The display can be cleared and the beginning of a plot synchronised to a change in value in one of the junctions being plotted.
6. Any junction can have its actual value changed.
7. A title line can be defined to replace the list of command options line. The screen can then be printed on a standard dot matrix printer with the different colours being represented by different dot patterns making up the lines of the plot.

The interaction between the plot program and the user is similar to the function building program. All commands are initiated by pressing one letter or number key; any further information is prompted for by an appropriate message printed on the bottom line of the display.

The plotting of values to the screen is synchronised to the execution rate of the function list. This is achieved by adding a function which simply puts a logic 1 into digital junction 9. The plot program waits until it detects a logic 1 in this junction and then clears it to logic 0

and plots the contents of the 3 junctions onto the screen. This technique ensures that the most efficient use is made of the screen without losing any changes in the value contained in a junction.

The plotting program enabled the operation of the functions to be accurately monitored in time and proved to be invaluable when testing complex functions such as the 3-term controller.

19.1 Plot Program language selection

During this project, compiled Basic was used as the high level language with any low level routine written in 8086 assembly language. The choice of high level language was made mainly from the point of view of availability and the provision of suitable language facilities for graphics.

After the plot program was developed using compiled Basic, a Turbo Pascal compiler was purchased. This version of Pascal provided sufficient graphics facilities to permit a second version of the plot program to be written in Pascal. This would allow a direct comparison to be made between the two languages with particular interest in their relative speeds of execution. The pascal version was designed to provide identical facilities to the Basic version so that a direct comparison could be made.

When the two versions were compared with respect to their speeds of plotting, the Basic version executed about

twice as fast as the Pascal version. This speed difference in favour of the Basic program was interesting but probably said more about the particular compilers used than any fundamental speed differences between the two languages. Nevertheless, this comparison along with inspecting the actual assembly language produces by the Basic compiler proved that the original language choice was satisfactory from an execution speed point of view.

When comparing the source code for the Basic and the Pascal versions, the actual length of the Pascal program was nearly three times the Basic source program length. This fact however, along with the use of named procedures and local variables resulted in a more understandable Pascal listing as compared with the rather compressed and difficult to follow Basic listing.

20 Performance Evaluation

The design of the complete language system was orientated towards its use as a real-time control language in the real world. It is very important that real-time events should be responded to quickly without obvious side effects such as the slowing down of operator interfaces or other aspects of the control system.

The design of the function interpreter, the functions themselves and the real-time executive were optimised for maximum speed of execution. Assembly language was used for these three sections of the program and a great deal of effort was put into optimising the assembly language code to increase execution speed. Since the function building program is a lower priority task than the control tasks, compiled Basic was used for ease of development and portability.

In order to be able to estimate the performance of the language, several measurements were taken to determine how long various parts of the program took to execute.These measurements were all taken on an IBM AT computer running at 6 MHz; this is certainly not the fastest possible machine that can run the system but it does represent a practical compromise between cost and speed.

The first performance parameter measured was the overhead imposed by the real-time executive alone. To

measure this overhead, a system was set up using no functions and a simple assembly language timing program was run under the operating system. The time taken by this timing program alongside the executive was compared with running the same program on the same computer running PCDOS alone. From these two measurements, the percentage of microcomputer time used by the real-time executive with all tasks enabled but not executing functions was determined to be 7.3%.

Because the functions are interpreted, their speed of execution has a major impact upon the size of control system that can be run. In some cases the actual execution time taken by a function depends upon the contents of junctions connected to that function. For example, if the reset input on a counter function is set to logic 1, the function will take different actions to those performed if the reset input is set to logic 0.

The execution times for the most commonly used functions were measured using a more direct method than that used to estimate the overhead of the real-time executive. The building program provides a command (command T) which displays the list period in units of milliseconds. A group of functions was built in an otherwise empty system to permit a function to be executed 100 times. The time taken by these functions alone was measured using the T command and subtracted from the time displayed when a function to be tested was added within the loop. This gives a sufficiently accurate execution time

for a function which includes any overhead in scanning down the list of functions. The approximate execution times for some of the functions running on a 6MHz IBM AT computer are as follows:-

Function Type	Execution Time (microseconds)
2 i/p and	40
flip-flop	40
integrator	85
compare =	50
counter =	100 to 125
switch	25 to 40
multiplexer	50
PID controller	225
filter	140
add	50
multiply	60
divide	90
bin to dec	90 to 175
digital offset	25
2 byte input	35
constant	17

Some of the above functions do not have a fixed execution time. This is because functions such as the counter or the switch perform different actions depending upon the contents of one or more junctions.

It is difficult from these figures alone to estimate the overall efficiency of this language. To give some idea

of execution time in a practical system, the list period for all of the pages shown in Appendix C is less than 4.5mS. This indicates that in normal process control situations, the execution speed is more than sufficient. The only times when a faster execution speed would be necessary is when very many fast events (less than 10mS response time required) must be handled or if the control system is very large. Many fast events may have to be handled by a dedicated assembly language written task running under the real-time executive but this should always be avoided if possible.

A very large control system could be divided up into sections relating to the response time required and each section run at different rates. Alternately, a more reliable system may be generated by running parts of the control system on separate intelligent outstations linked to a masterstation. This would not only increase the maximum size of system but would also increase the overall reliability.

One final method of increasing the speed and size of control system possible is to use a faster microcomputer. The figures for execution speed were taken using a 6 MHz IBM AT computer which uses an 80286 microprocessor. The same software can be run on the more recent and powerful 80386 computers which would improve the overall speed of everything by a factor of between 3 and 5.

References

1. Wyss, C. R. "A Conceptual Approach to Real-Time Programming". pp 452-462.: Byte May 1983.
2. EEUA "Guide to the engineering of microprocessor based systems for Instrumentation and control". Engineering Equipment Users Association 1981.
3. IBM "PC Technical Reference". IBM Corporation 1983.
4. IBM "BASIC Reference". IBM Corporation 1984.
5. IBM "Disk Operating System Technical Reference". IBM Corporation 1984.
6. Rector, R. and Alexy, G. "The 8086 Book". Osborne/McGraw Hill 1980.
7. Transmitton Ltd. "Energy Savings and Control in Malting". Symposium: June 1981.
8. McCabe, M. A. "Microflex Process Control System Handbook". Transmitton Ltd.: August 1981.
9. Roach, N. "Concurrent Programming". EXE Vol 1 Issue 7 pp 12-22.: 1986.
10. King, R A. "The IBM PC-DOS Handbook". Sybex: 1983.
11. Norton, P. "Inside the IBM PC". R. J. Brady: 1983.
12. Cargill, C.G. "Interfacing Microsystems with people". Colloquium "The engineering of industrial Microprocessor-based systems - the users´ point of view". March 1981.

13. Negretti and Zambra. "MPC80 Specification TS 7399-23". 1982.

14. GEC Industrial Controls Limited. "GEM 80 T100". 1981.

15. Foxboro. "Display Block Configuration". September 1983.

16. McDermott, R. M. "The design of an advanced Logic Simulator". pp 398-426.: Byte April 1983.

Appendix A

Development Facilities

This appendix defines the setup and use of the full real-time system; this includes the format, function and use of all commands within the building program.

Table of Contents

1 Entry and exit 1

1.1 Signing on 2

1.2 Quiting 3

2 Wrong commands and mistakes 4

3 Cursor movement 4

3.1 Using the keyboard 5

3.2 Using a joystick/mouse 5

4 Outstations 6

5 Loading pages 6

5.1 What is loaded? 7

5.2 Using pages from a library 8

6 Saving pages 8

6.1 What can be saved? 9

7 Erasing pages 9

8 Loading a new function list 10

9 Window into another page 10

10 Inserting junctions 11

11 Inserting functions 12

11.1 Function selection 12

11.2 Function numbering 13

11.3 Drawing connecting wires 14

11.4 Commands available when connecting functions 15

11.5 Selections after connecting 15

12 Inserting comments 16

12.1 Changing colours 16

13 Monitoring junctions 18

13.1 On screen junctions 19

13.2 Off screen junctions 19

14 Offsets 20

15 Displaying, changing and blocking junctions and monitors 22

15.1 Displaying a junctions contents 22

15.2 Changing the actual value 23

15.3 Blocking and unblocking 23

16 Renumbering junctions 24

17 Junction cross referencing 24

18 Clearing all junctions 25

19 Changing and rewiring functions 26

19.1 Changing inversions 26

19.2 Changing the execution order 27

19.3 Changing the constant function value 27

19.4 Changing a connection 27

20 Deleting 28

20.1 Deleting functions 29

20.2 Deleting junctions 29

20.3 Deleting junction numbers 30

20.4 Deleting monitors 30

20.5 Deleting characters and words 30

21 Printouts 31

21.1 Page printouts 31

21.2 Cross reference printouts 32

22 Blocked junction information 32

23 List execution information 33

23.1 list execution times 33

23.2 List catch-up counters 33

23.3 Function list page length 34

24 Memory display 34

25 Colour - monochrome displays 35

Appendix A

Development Facilities

1 Entry and exit

When a system is initialised, the function lists are loaded into the masterstation and, if relevant, the outstations. Control is then returned to DOS which is then operating as a task under the main task executive. A batch file may be used to initiate a power up sequence and if required, the Basic interpreter may be loaded and automatically start a Basic program. If an attempt is made to break a Basic program without being signed on with an access code greater than 2, it is ignored. A valid sign on code must be given to gain entry into the development program; an incorrect code will return the operator back to the Basic program.

A typical MSDOS batch file which enables a system to load and execute a Basic program called EXAMPLE.BAS could be:

```
BUILDER
INIT
BASICA EXAMPLE
```

If only one VDU is in the system, the engineer must press <control-1> to switch between Basic and the

development program; a correct sign-on code must be given before program changes can be made.

1.1 Signing on

In a single VDU system, the operator is taken into the development program by pressing <control-1>; <control-1> will also return back to Basic/DOS. The first action within the development program must be to sign on.

There may be many sign on codes in an application; each sign on code is associated with an access code between 1 and 3 (or 9 for debug access). This access code determines the facilities available under a particular sign on code as outlined below:

Access code 1	Commands **H, I, L, M, O, Q, T, U,** (list only), **W, X, ?, text entry.**
Access code 2	As code 1 and commands **B, C** (actual value only), **D** (comments and monitors only), **R.**
Access code 3	As code 2 and commands **A, C, D, E, F, J, K, N, P, S.**
Access code 9	As code 3 and commands (for debug purposes only) **G, V, Y, Z.**

To summarise:

Access code 1	Permits pages and junctions to be displayed only.
Access code 2	Also permits junction contents to be

changed.

Access code 3 Also permits functions to be changed and saved and the BASIC interpreter program to be stopped and changed.

The sign on code may consist of up to 40 alphanumeric and punctuation characters and is not echoed on the screen. The codes themselves are stored in a simple text file called SIGNON.DAT held on the system disk. This file may be edited to contain one access code and sign on code per line - there is no limit to the number of codes in use.

After successfully signing on, the date, time and sign on code of the user is recorded in a text file called LOGIN on the system disk.

1.2 Quiting

When the main command prompt is displayed, a user may quit the development program by typing <q>. The user is given the opportunity to not quit and to save the current page if it has not been saved. After successfully quiting, the date and time is recorded in a text file called LOGIN on the system disk and the user is returned to the Basic program/DOS.

Note that monitors present on the screen are still updated even when the operator is not signed on. A page containing important monitors may be left on the screen

after quiting which provides a simple display which may be viewed by an operator but not changed.

2 Wrong commands and mistakes

Most commands may be aborted by pressing <Esc>; this will restore the situation just prior to entering the command. If this action causes a significant loss of data (such as during the entry of a function) then an option is given to the user to not abort. If an abort is requested after deleting something, then the abort command is refused (for example, after deleting a function connection where an abort would leave the function incompletely connected).

When invalid commands or data are entered, a short "beep" is sounded as a warning.

During data entry, the delete key may be used to delete the last character entered. If more characters are entered than the program expects in the reply, all characters are deleted and a "beep" is sounded. If a number is requested, pressing a non-numeric character will have no effect.

3 Cursor movement

The keyboard and/or the joystick/mouse may be used to move

the cursor on the screen; the cursor cannot be moved beyond the screen boundaries. In general, anything that is to be placed onto the screen is positioned with its top left corner on the cursor position.

3.1 Using the keyboard

The following keys move the cursor:

up arrow	up one character
down arrow	down one character
left arrow	left one character
right arrow	right one character
home	top left corner of the screen
End	bottom right corner of the screen

3.2 Using a joystick/mouse

The joystick/mouse may be used to move large distances on the screen. If cursor movement keys are pressed, the joystick/mouse no longer represents the true position of the cursor; the joystick/mouse will regain control again once it is moved.

4 Outstations

If the system has been configured to contain outstations then there will at times be additional questions to select the outstation to be used for a particular command. The system held in memory in the same computer as the development DRAW program is designated the Masterstation; other systems which communicate over a link are called outstations and are individually identified by a single letter from ´A´ to the maximum number of installed outstations.

If during communications with an outstation a reply is not received from that outstation then the outstation is regarded as off-line. In most instances this will not cause any loss of data at the masterstation or any mismatch of list data between the masterstation and the outstation except in some unavoidable and rare circumstances when a non-reversable change is made that cannot be communicated down to the outstation.

5 Loading pages

When the main command prompt is displayed, pressing <1> will load a new page from disk. If the current page has

not been saved since changes were made to any function, then a warning message is given and an opportunity to exit from a page load.

If the selected page does not exist on disk then the screen is cleared and any new functions entered will be placed into the selected page of the list. This is the normal way of beginning a new page.

If any functions are in the list under the selected page, these are deleted and replaced with the new functions from disk.

5.1 What is loaded?

There are 4 files on the data (B) disk for each page; these are as follows where ?? indicates a page number:

PS??.DAT	image array of the screen (4050 bytes).
PA??.DAT	memory image of SCRN array (4050 bytes).
PM??.DAT	memory image of MONITOR array (210 bytes). This will still be present even if no monitors were on the screen.
PF??.DAT	memory image of page ?? in the function list. This will not be present if the page has **never** contained any functions.

If the page is part of an outstation then the file name will have the outstation letter (A-Z) appended to the end of the page number.

5.2 Using pages from a library

When a page is loaded, only that page in the function list is affected. It is possible to load a page from a different system without deleting part of the current program provided the selected page is unused. If it is used, then after loading, the new page can be saved under its new page number and the displaced page reloaded from disk.

It is possible to develop a collection or library of useful pages which may be selectively loaded into a system. After loading, the junction numbers will have to be changed to link into the existing function list.

6 Saving pages

Pages may be saved to disk in drive B using command <s> when the main command prompt is displayed. Normally the page is saved under the current page number and outstation, but it is possible to save under another page number for the purposes of library creation and copying pages. WARNING - if the page number or outstation is changed before saving, the functions do not exist in that page in the current list in memory.

6.1 What can be saved?

Up to 4 files may be saved on disk (see previous subsection titled "What is loaded?").

If monitors are present on the screen, an option is given to delete these before saving. If the page is being saved under a different outstation then the monitors will be deleted anyway.

A file containing the portion of the function list will only be saved if there are functions in that page in the list; if a file containing functions for this page is already on the disk, it will not be deleted or reloaded.

7 Erasing pages

Typing <e> when the main command prompt is displayed will erase the current page. The screen will be cleared along with all monitors and the functions in the current page in the list will be deleted. Any information held on disk under the current page is not deleted and may be reloaded if necessary.

8 Loading a new function list

Typing <n> when the main command prompt is displayed will erase all pages currently held in the function list and load a complete new set of pages from disk.

If outstations are present then a sequence of questions are asked whether to load each particular outstation.

Before loading, the main function list and the sub-lists are stopped by blocking analogue junctions 1,2 and 3 to zero. To restart the new system, these must be unblocked using command <u> or by selectively clearing the blockages using <b>.

9 Window into another page

As well as the current page being displayed, another page may be loaded and viewed along with its monitors. This 'window' page is selected by pressing <w> when the main command prompt is being displayed and does not change any information in the currently active page. The window page can only be viewed and any changes are not permitted without first loading it in as an active page.

To load a new window page, select <w> followed by <l>. This will initiate a sequence of questions to select the

desired page in the same way as active page loading. Selecting any other command except <l> whilst a window page is being displayed will be ignored and will cause the active page to be displayed.

Using the window facility enables up to 40 monitors to be viewed without reloading pages. The window page can be any page in any outstation using, if necessary, different offsets as defined when the page is loaded.

10 Inserting junctions

Typing <j> when the main command prompt is displayed will place a junction on the screen at the cursor position. It is not possible to place a junction which is not in the current outstation being displayed. The following sequence of questions will be asked:

1. "Type <a> for analogue or <d> for digital junction". Any other key will give an error message.
2. "Enter analogue/digital junction number". Pressing <enter> only, will cause the next number after the last reply to be selected. A warning message is given if too large a number is entered. Since the warning message gives the maximum number of junctions of that type in the system, entering of a large number may be used to determine the size of the junction data-base in

any particular system.

3. "Put cursor to display point for junction". The junction number will be placed on the screen at the cursor position when any key is pressed. There must be sufficient empty screen for the number on and to the right of the cursor. Always try to place the number close to the associated junction symbol.

11 Inserting functions

Typing <f> when the main command prompt is displayed will begin a sequence of questions to place a function on the screen and connect it up to junctions. The function is not put into the function list for execution until it is completely wired-up.

11.1 Function selection

A function may be selected using one of 2 methods:

1. Type the function name in response to the "Name of function " question. Only part of a name has to be given since a search is made of the list of function names for a match between the name entered and all or part of each function name; the first match found will

decide the function. For example, typing "**and**" will select a "2 I/P **and**" function since this is a match and it occurs before "4 I/P **and**" and "sample **and** hold".

2. Pressing the <enter> key in response to the "Name of function" question will display a complete list of functions available. To select a function, enter the number of the function which is displayed to the left of the name.

11.2 Function numbering

Each function is assigned a reference number which is displayed at the bottom right of the function; this number indicates the execution order of the function in the function list. Normally, functions are numbered in a logical order from input to output such that an input signal is processed as fast as possible (i.e. one scan of the function list) to give a new output.

A function may be inserted into a page as the last function executed by responding to the "Enter the function execution order" question with key <enter>. A function may be inserted between existing functions by entering the desired reference number. If a function is inserted into a page containing other functions then the reference numbers are changed for every function after the new one.

11.3 Drawing connecting wires

Once a function has been drawn onto the screen and numbered, it must be completely wired up before any other operation can be performed. Each input/output connection of the function must be joined by a line to an appropriate junction (analogue or digital); the connection order is from the top to the bottom and from the left to the right of the function. A flashing `diamond´ shape indicates the I/O function point currently being connected.

A line is drawn by positioning the cursor and pressing any key except <Esc>, <j> and <d>. A line is automatically drawn from the last line end to the cursor subject to the following rules:

- a line will first of all continue in its original direction as far as necessary.
- The additional section of line can have a maximum of one corner. An exception is the very first section of line connected to the function which may have two corners under certain circumstances.
- The new line may cross over an existing line or character but it may not cross over more than one adjacent line or character.
- If a blockage is detected (i.e. more than one adjacent line or character, a function, junction, monitor or word), a Beep will sound and no line will be drawn.

An I/O point on a function is considered connected when it is linked into the correct type of junction.

11.4 Commands available when connecting functions

The commands available during the connection of a function permit:

- new junctions to be inserted (<j>)
- a word, monitor, junction or junction number to be deleted (<d> over a word, monitor, junction or junction number)
- the line drawn so far to be deleted (<d> anywhere else)
- <Esc> gives the option to delete the entire function drawn so far along with its connections.

Only a junction of the correct type for the I/O connection may be inserted using the <j> command.

11.5 Selections after connecting

If the I/O connection is to a digital junction then an option is given to invert the connection or not.

The constant function is the only function which is not completely connected by linking to junctions. This function requires a constant value to be entered in decimal or hex in response to the appropriate question presented

after connecting the functions´ only I/O point. The format of the number displayed in the function is the same as the format that was used to enter the number (i.e. decimal or hex).

12 Inserting comments

Characters, words and graphics symbols may be put onto the screen by first of all entering the text-entry-mode. To enter the text-entry-mode from the command-mode, press the <Ins> key once; press this key again to return to command-mode. An alternative to the <Ins> key is to press control-w.

In the text-entry-mode, the cursor may be moved in the usual way; characters are inserted at the cursor position provided the position is currently occupied by a blank or a character. Existing characters can therefore be overwritten if required.

Graphics characters are produced by generating ASCII codes greater than 127 from the keyboard.

12.1 Changing colours

Comments may be entered using any foreground, background and surrounding colour combination; foreground colours may

be highlighted and/or flashed.

To change the foreground colour, press control-f in the text entry mode. The program will prompt for a code number for the new colour where 0 is black and 7 is white. Add 8 to obtain highlighted versions of these colours. Add 16 to obtain a flashing character.

To change the background colour, press control-b in the text entry mode. The program will prompt for a code number for the new colour where 0 is black and 7 is white.

To change the surround colour, press control-s in the text entry mode. The program will prompt for a code number for the new colour where 0 is black and 7 is white. Add 8 to obtain highlighted versions of these colours.

Pressing control-k in the text entry mode will preset the foreground and background colours to the same colours used to display monitor actual values. These colours should be used to display the numerical contents of constant junctions next to the relevant junction; this helps in the understanding of circuits.

Pressing control-n in the text entry mode will preset the foreground and background colours to their normal or initialised state.

The selected colours will remain in operation for all comments until they are changed again.

13 Monitoring junctions

A monitor is a real-time display of a junctions´ contents, both actual and blocked. Any junction may be monitored in the masterstation or any outstation.

The display may be in decimal or hex; this is selected using the output radix toggle (<i>) command when the cursor is over the monitor value block or the monitor junction number. A decimal number is displayed as a signed value with up to 5 digits; a hex number is displayed as an unsigned value between 0H and FFFFH. The <i> command operates as a toggle between decimal and hex.

Up to 20 junctions may be monitored on the screen at any one time. These may be displaying the contents of junctions on the screen or may be monitor-junctions set up to display the contents of off-screen junctions.

Both types of monitor display share a common format to display the actual and blocked junction contents. The values are contained within an inverse video block 2 lines high by 1 wide for a digital junction and by 6 wide for an analogue junction. The top line contains the actual value and the bottom line the blocked value. If the bottom line is empty, the junction is not blocked.

If an attempt is made to display more than 20 monitors, a warning message is given.

13.1 On screen junctions

Typing <m> with the cursor over a junction or junction number when the main command prompt is displayed will define a new monitor displaying that junctions contents. The program will prompt for the cursor to be moved to the display position for that monitor which will be displayed when a key is pressed. As usual, the monitor block will be positioned with its top left corner on the cursor position; a warning message is given if there is not enough free screen space for the monitor in the chosen position.

13.2 Off screen junctions

To display the contents of a junction not on the screen, type <m> with the cursor over an empty section of screen when the main command prompt is displayed. This will define a new monitor consisting of a monitor-junction, monitor-junction number and monitor value block; The monitor-junction and number are not the same as normal junctions and cannot be connected to functions.

The program first of all requests the outstation containing desired junction and then proceeds to request the junction type and number in the same manner as placing a normal junction on the screen using the <j> command. The final prompt is the same as given when monitoring an on-screen junction. An off screen junction monitor number

has a prefix to indicate the outstation relevant to that monitor. The prefix ´m´ indicates masterstation and the letters ´A´ to ´Z´ indicate outstations.

14 Offsets

When an offset function is executed the functions following do not access the junctions specified on the screen display. To overcome the considerable monitoring and debug problems created by this mismatch, an offset command <o> is provided to compensate for the difference between displayed junction contents and actual junction contents being referenced by the functions.

Pressing <o> when the main command prompt is displayed will result in 2 questions being asked. The first requests an offset to be used when displaying digital junctions and the second questions requests the offset for analogue junctions. Pressing <0> <CR> or <CR> only will clear the offset facility; pressing <Esc> will leave the offset value unchanged. The current offsets used within the display of a page are indicated on the main command prompt line.

A non-zero offset results in that offset number being added to every junction number of that type on the screen and the result used as the junction number to display.

For example, if an offset function producing a digital offset of 500 is executed before the function being

displayed on a screen page, then all digital junctions being displayed will not normally correspond to those actually being referenced. A digital junction 10 connected to a function results in junction 510 being accessed by that function. If junction 10 is displayed using a monitor or the <?> command then the contents of junction 10 are displayed, not the contents of 510. If the <o> command is used to set a digital page offset of 500 then the junction contents being displayed will be 510 even though the screen still shows a junction number 10.

If a sequence of pages operate as a subroutine which is called several times with different offsets set by offset functions then each execution will operate upon a different group of junctions. This is like a procedure using local variables but with the facility to still pass and return any value(s) to other parts of the function list. The <o> command enables the operator to examine each execution of this procedure or subroutine without changing the monitor displays or calculating the actual junctions being referenced.

All actions upon junctions affected by a non-zero screen offset will take the offset into account. This includes all of the commands to change and block an individual junction value but not the commands to list (and unblock) all junctions or the junction cross reference command.

The offset command affects only the monitors on a page which display values in the oustation being displayed.

Monitors of other junctions not in the current outstation will be unaffected.

It is possible to monitor a junction which, combined with an offset, refers to a junction outside of the allowed junction data base. This will be indicated by a flashing actual value monitor or an off-line message.

When a page is saved, the offset status is also saved and will be automatically restored on reloading.

15 Displaying, changing and blocking junctions and monitors

The contents of a junction cannot be displayed or changed without the junction being present on the screen as either a connectable junction or a monitor junction.

See the section on junction monitors to toggle the displayed monitor value between decimal and hex.

15.1 Displaying a junctions contents

To continuously display the contents of a junction, use the monitor (<m>) command. If the contents of a junction is required without continuous monitoring, position the cursor over the required junction or junction number and press <?>. The junction contents will be displayed on the bottom prompt line and will remain there without being updated

until a new command is given.

15.2 Changing the actual value

The actual contents of a junction can be changed by positioning the cursor over the required junction point or a junction monitor and pressing <c>. The program will prompt for a new actual value which, for an analogue junction, may be entered as up to 5 decimal digits with a leading -ve sign if required or as up to 4 hex digits followed by <h>.

15.3 Blocking and unblocking

The blocked contents of a junction can be changed by positioning the cursor over the required junction point, junction number or junction monitor block and pressing <b>. The program will prompt for a new blocked value which, for an analogue junction, may be entered as up to 5 decimal digits with a leading -ve sign if required or as up to 4 hex digits followed by <h>. Pressing <enter> only in response to the prompt will clear the blockage from that junction.

Note that all junctions may be unblocked using the junction reset command (<r>) and the junction unblock command (<u>).

16 Renumbering junctions

A junction number can be changed by positioning the cursor over the number and pressing <c>; all occurances of the junction on the screen and in the current page of the list will be changed.

The program will request a new junction number in the same manner as inserting a new junction. All occurances of the old junction number are first of all deleted followed by prompts to enable the new number to be placed in any desired free space again, in the same manner as inserting a new junction.

If a monitor exists for the old junction number then this will now display the new junction contents.

17 Junction cross referencing

Typing <x> with the cursor over a junction, junction number or monitor when the main command prompt is displayed will display a list of functions connected to that junction. The list is displayed on the prompt line at the bottom of the screen and will remain until a new command is given.

The list consists of page numbers followed by function reference numbers within the page. If more connections are

present than can be displayed on one line, a key must be pressed to display the rest of the connections.

WARNING! Take care with this command since it only displays actual connections between functions and junctions and not the implied connections. An implied connection is one where a function accesses a junction without a direct connection being shown on the diagram. For example, the scanner function can scan down a list of many junctions but only the first junction will be displayed as connected to the scanner function. Other functions using implied connections include:

- scanner
- data divider
- logic assembler
- decoder
- demultiplexer
- delay

This command does not take account of screen offsets; i.e. a screen offset of zero is assumed.

18 Clearing all junctions

Typing <r> when the main command prompt is displayed will first of all block analogue junctions 1,2 and 3 to disable the function list and both sub-lists. All other analogue

and digital junctions are then reset to 0 and unblocked. This command initialises the junctions to their power-on state and when used with the unblock command (<u>) enables the system to be restarted as though it had just been powered up.

If there are outstations in the system then these may be selected for junction reseting.

19 Changing and rewiring functions

Functions may be changed in a variety of ways as described below. The change does not actually take effect until it has been completed.

19.1 Changing inversions

Typing <c> with the cursor over an I/O point on a function when the main command prompt is displayed will invert the connection to a digital junction. If the connection is already inverted then this command will normalise it.

If an attempt is made to invert an analogue junction connection, a warning message is given.

19.2 Changing the execution order

Typing <c> with the cursor over any part of a function except an I/O point when the main command prompt is displayed allows the execution order of that function to be changed. After entering a new execution order number in response to the prompt, other functions on the screen are renumbered as necessary to reflect the functions new order.

No two functions can have the same number; renumbering to an existing function number will result in the existing function and function with numbers above being renumbered up by one.

19.3 Changing the constant function value

If the change execution order command (<c>) is entered with the cursor over a constant function, an additional option is presented which permits the constant function value to be changed. The entry of the constant value is the same as given under the subsection "selections after connecting".

19.4 Changing a connection

The operation to change a connection between a function and a junction is in fact to delete the connecting wire and

reconnect.

Typing <d> with the cursor over any part of a connecting line when the main command prompt is displayed will erase that complete line from the screen. The function I/O point will now flash to prompt for the function to be reconnected in the same way as entering a connection for a new function. This operation cannot be aborted using the <Esc> key.

If a junction is adjacent to a function I/O point then there is no visible line to delete. In this instance, position the cursor over the I/O point and type <d>.

When a line which crosses over other lines is erased, gaps are left in parts of the other lines. The program will attempt to "repair" the other lines and it will be successful in virtually all cases. There are some rare instances where a "repair" results in an incorrect shape being inserted, particularly at crossed corners of lines near to junctions. This does not effect the logical circuit operation and of course, any line may be tidied up by deleting it and reconnecting to the same junction.

20 Deleting

The <d> command is used to delete a variety of things depending upon what is under the cursor at the time.

20.1 Deleting functions

Typing <d> with the cursor over any part of a function when the main command prompt is displayed will erase that complete function along with its connections from the screen and the function list. An option to abort from this command is given before the function is actually deleted.

All of the functions in the page after the deleted function are renumbered down by one.

20.2 Deleting junctions

Typing <d> with the cursor over a junction point when the main command prompt is displayed will erase that junction from the screen. If the junction is physically connected to a function on the screen then that junction cannot be deleted and a warning message is given.

If the junction is being monitored then that monitor is also deleted.

If there is more than one occurance of the junction to be deleted on the screen and these are close together, there is no way of knowing which junction number corresponds to which junction. The program therefore moves the flashing cursor to each number and prompts to ask whether to delete it or not. Only one number can be deleted.

20.3 Deleting junction numbers

Typing <d> with the cursor over a junction number when the main command prompt is displayed will erase that junction number from the screen. The cursor must be moved, in response to the prompt, to the new position for the number. A junction number cannot actually be deleted from the screen, only repositioned.

20.4 Deleting monitors

Typing <d> with the cursor over a monitor junction, monitor junction number or monitor value block when the main command prompt is displayed will erase that complete monitor from the screen.

20.5 Deleting characters and words

Typing <d> with the cursor over a character when the main command prompt is displayed will erase the complete word containing that character.

Pressing the delete key when the main command prompt is displayed will erase the single character to the left of the cursor and move the cursor one position to the left.

21 Printouts

The use of a printer requires the correct control codes for bold printing and line spacing control being in the DRAW program and the presence of a printer data file called "PRINTER.DAT" on the system disk. This file programs the printer to display the special graphics shapes used in the screen displays.

When printing graphics on an FX80, the line spacing is reduced to allow the shapes to join. This also has the effect of reducing the vertical spacing between comments. An alternative version of PRINTER.DAT is available for the IBM Proprinter.

21.1 Page printouts

Typing <p> when the main command prompt is displayed will send an image of the screen to the printer. An option is first of all given to cancel the command.

Three blank lines are printed followed by an emphasised and underlined title giving the page number. After one further blank line, the 24 lines of screen image is printed which is finally followed by three blank lines.

21.2 Cross reference printouts

A full junction to function cross-reference printout can be produced on a printer. Each connected junction of a chosen type between a specified range of junction numbers is listed along with its function connections in a similar way to the single junction cross-reference using the <x> command.

The selection of the junction range is made by first of all locating the cursor onto the first junction of the desired type in the desired range and pressing <a>. The program will then prompt for the final junction reference number in the desired range; this will default to the maximum reference number if either no number is entered or a number greater than the maximum is entered.

An emphasised title is then printed followed by a list between the chosen limits. This list exhibits the same limitations as outlined in the section titled "Junction Cross Referencing".

22 Blocked junction information

All blocked junctions can be listed on the prompt line by typing <u> when the main command prompt is displayed. An option is given to also clear all blockages as they are

listed. Any key should be pressed in response to the "<key>" message to allow lists of blocked junctions longer than the prompt line to be displayed. Pressing the <Esc> key will abort the search at any point.

23 List execution information

Typing <t> when the main command prompt is displayed will give information about the operation of the function lists on the prompt line. There are three groups of information given which are described below.

23.1 list execution times

The current main list execution period is given in mS to an accuracy of about +/- 1 timer-interrupt-period (i.e. +/- 1.7mS). This is how long it took to execute the complete function list the last time it was executed. This may be used to determine the maximum execution rate that can be applied to a particular list of functions.

23.2 List catch-up counters

Three numbers are displayed indicating by how many list periods the system has fallen behind the execution rate set

up by the list-rate-control analogue junctions 1, 2 and 3. These should normally show 0 and sometimes 1; numbers greater than these indicate that the execution rate has been set too fast for reliable operation.

23.3 Function list page length

The number of bytes taken up in the function list by the currently displayed page is shown. This is for general information only.

24 Memory display

Typing <g> when the main command prompt is displayed will result in a sequence of questions to select an area of memory for display.

The first question selects the segment address which may be entered in decimal or hex. The following characters will automatically determine and select the following segments:

<c> <CR>	segment address for the Task executive code; i.e. the TASKS segment.
<d> <CR>	the Data segment.
<CR> only	will select whatever segment was last selected.

The second question selects the address within the selected segment to be displayed.

<b> <CR>	the start of the function page buffer.
<c> <CR>	the start of the communications buffer, OS_DATA.
<a> <CR>	the start of the page address offset table in the masterstation.
<l> <CR>	the start of the function list in the master station.
<CR> only	will select the next 16 bytes after the last selected.

The display will then show the 16 bytes starting at the selected address in both hex and ASCII.

When a memory display is on the command line, the next 16 locations may be displayed by pressing the <space> key.

25 Colour - monochrome displays

The program is initialised to use a colour display. Pages stored on disk in monochrome will not be displayed in colour; likewise, if pages have been stored in colour, they may cause patterning on a monochrome display.

Typing <y> when the main command prompt is displayed will toggle between colour and monochrome modes. This does

not affect pages stored on disk or the current display screen as it only changes the colours used when new information is added to the screen.

Appendix B

Function Diagrams

This appendix defines the shapes and input/output points for all of the functions currently developed. The diagrams are in the same form as displayed on the VDU screen and as printed out from the building program.

Function Diagrams

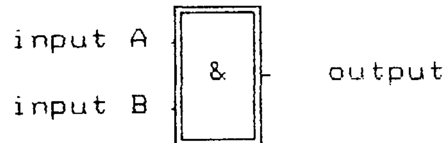
A digital I/O point is indicated by a single line protruding from the function shape (e.g. ╢ or ╟). The connection to a digital junction may be inverted if required.

An analogue I/O point is indicated by a double line protruding from the function shape (e.g. ╣ or ╩). The connection to an analogue junction cannot be inverted. A function using an analogue junction will operate using signed 2-byte arithmetic unless otherwise stated.

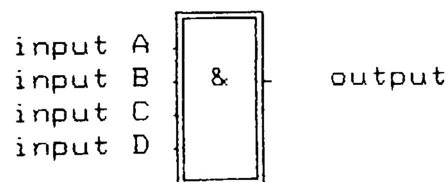
DIGITAL LOGIC

Function "buffer"

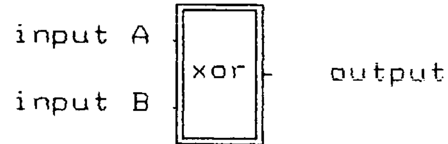
Function "2 i/p and"



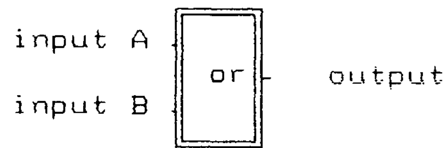
Function "4 i/p and"



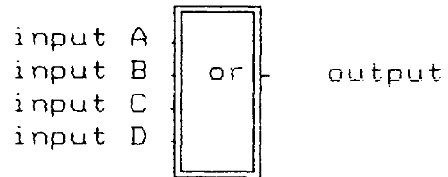
Function "exclusive or"



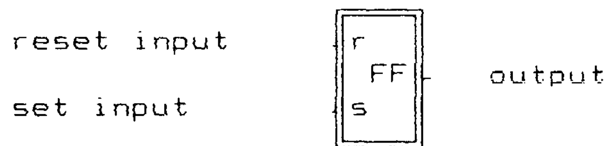
Function "2 i/p or"



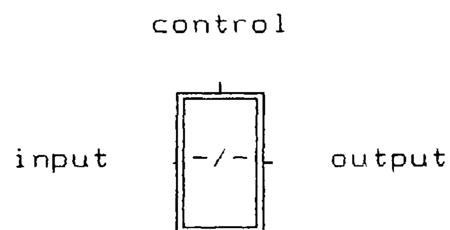
Function "4 i/p or"



Function "flip flop"

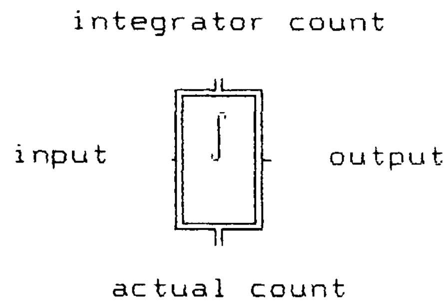


Function "latch"

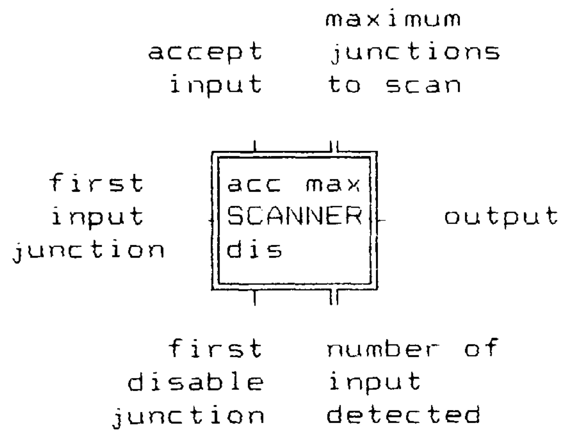


MISCELLANEOUS DIGITAL FUNCTIONS

Function "integrator"



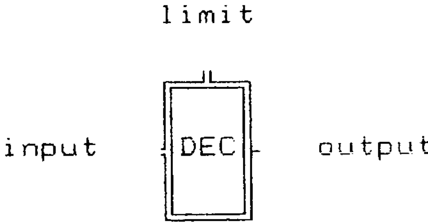
Function "scanner"



Function "data divider"

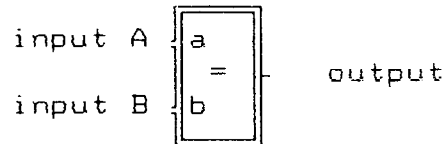
Function "logic assembler"

Function "decoder"

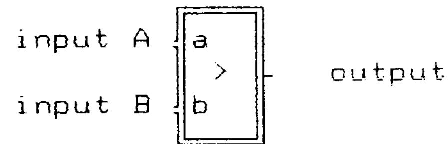


ANALOGUE FUNCTIONS

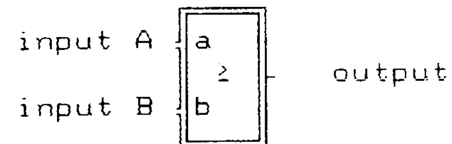
Function "compare ="



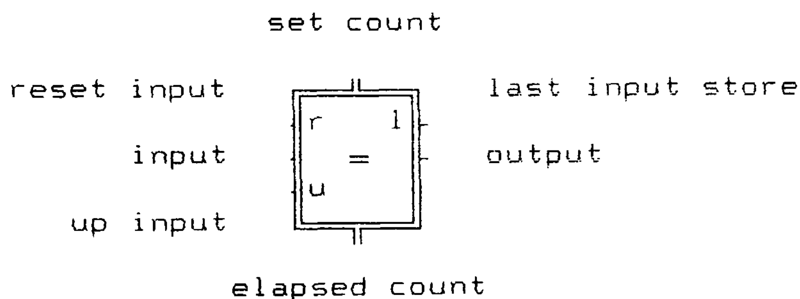
Function "compare >"



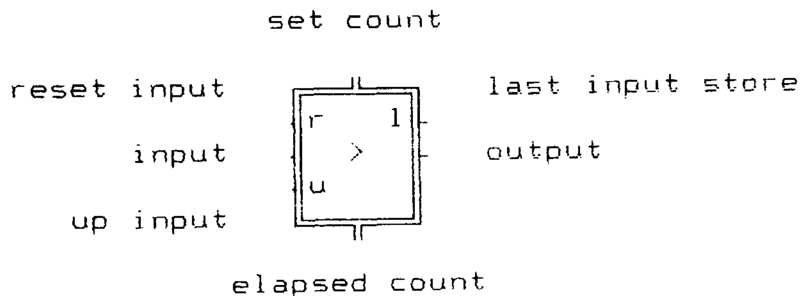
Function "compare ≥"



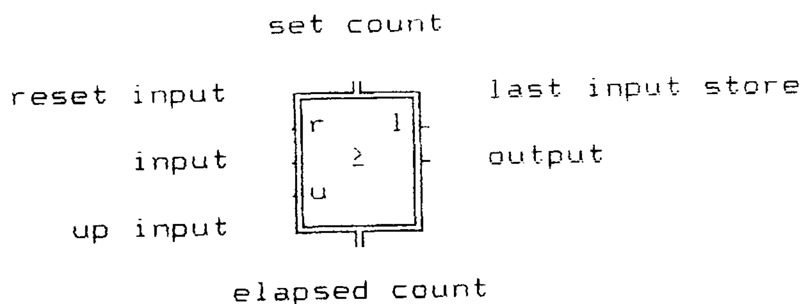
Function "counter ="



Function "counter >"

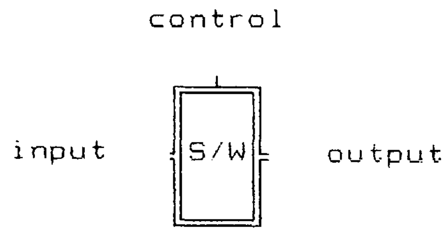


Function "counter ≥"

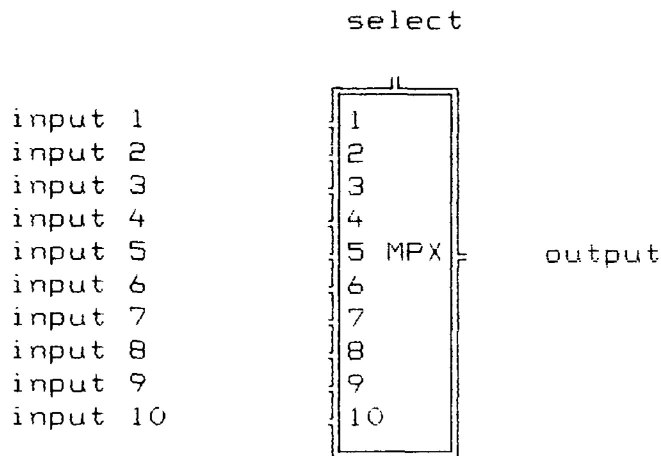


Function Diagrams

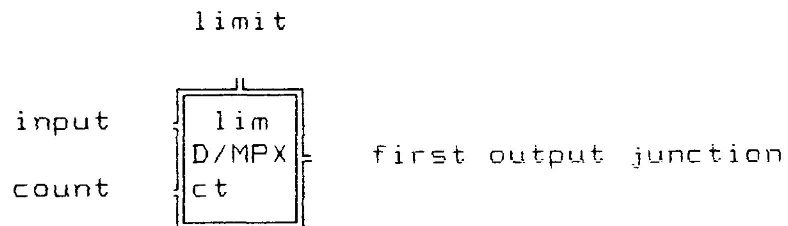
Function "switch"



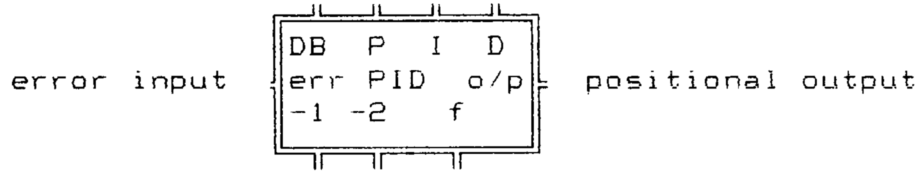
Function "multiplexor"



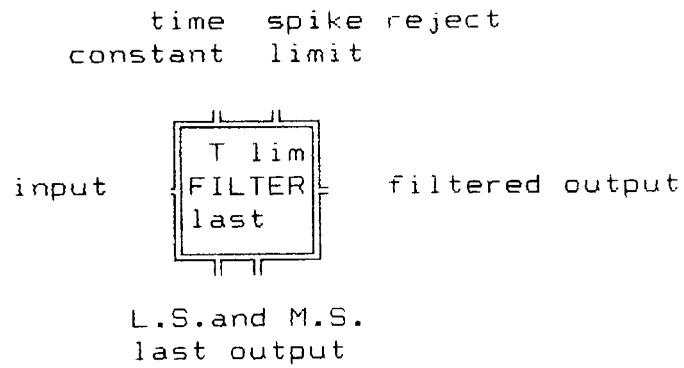
Function "demultiplexor"



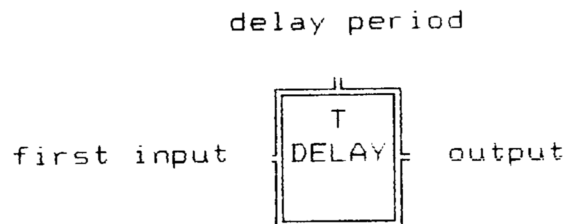
Function "PID Controller"



Function "filter"

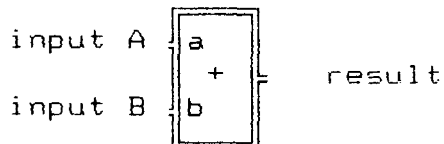


Function "delay"

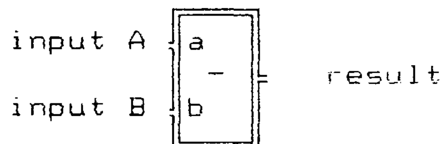


MATHS FUNCTIONS

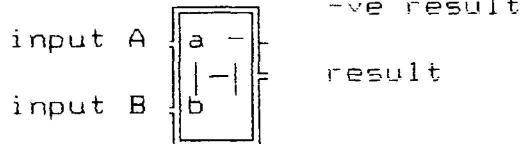
Function "add"



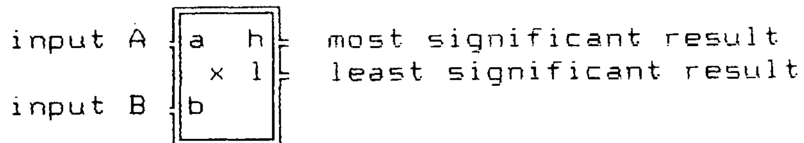
Function "subtract"



Function "abs subtract"

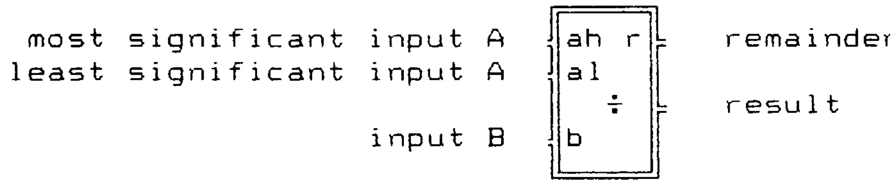


Function "multiply"

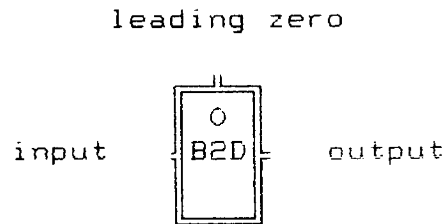


Function Diagrams

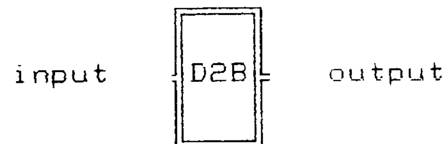
Function "divide"



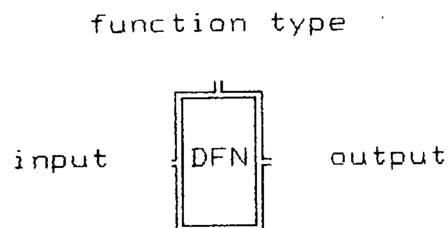
Function "bin to dec"



Function "dec to bin"

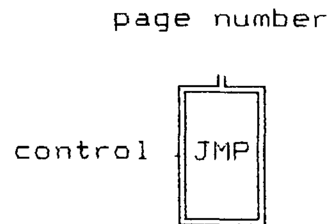


Function "define function"

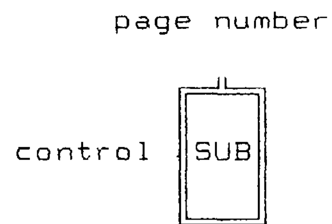


LIST CONTROL FUNCTIONS

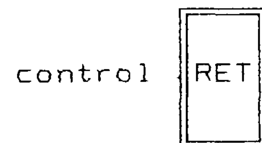
Function "jump fwd"



Function "subroutine fwd"



Function "return"

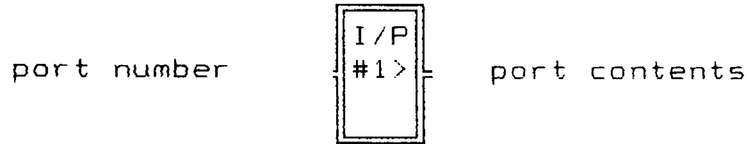


Function "digital offset"

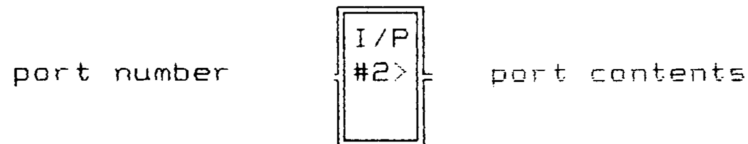
Function "analogue offset"

INPUT / OUTPUT FUNCTIONS

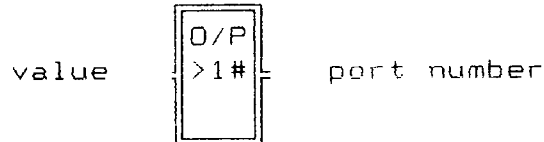
Function "1 byte input"



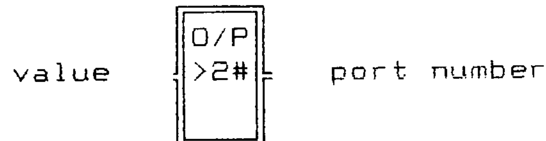
Function "2 byte input"



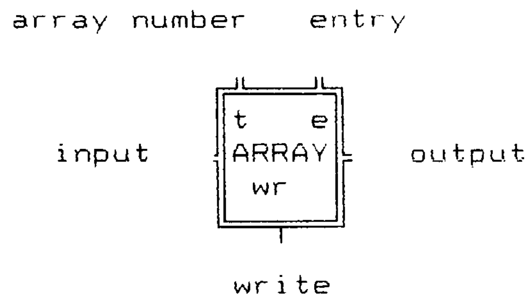
Function "1 byte output"



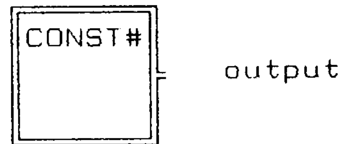
Function "2 byte output"



Function "BASIC array"



Function "constant"



Appendix C

Program Examples

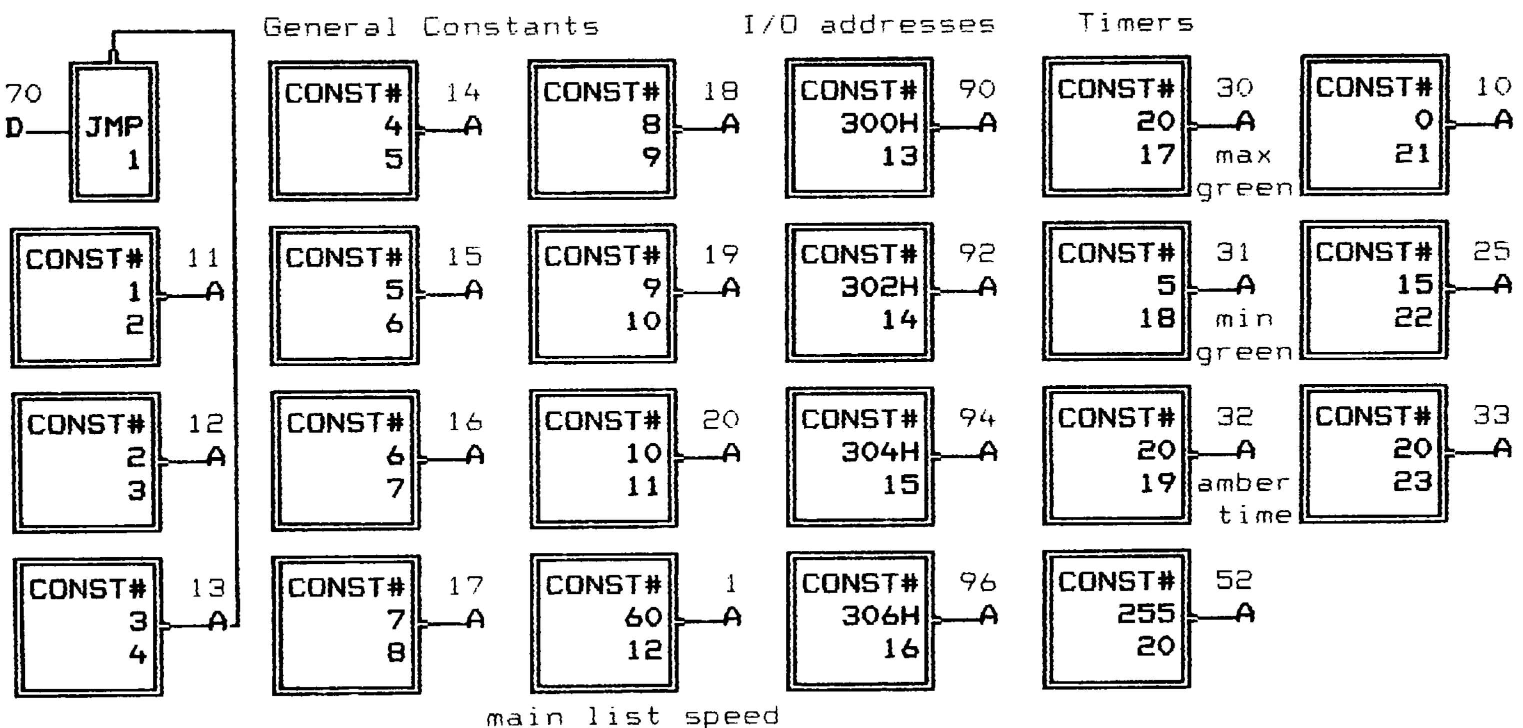
This appendix contains actual printouts produced by the building program. The system printed here contains several control simulation examples as well as a full traffic light simulator example.

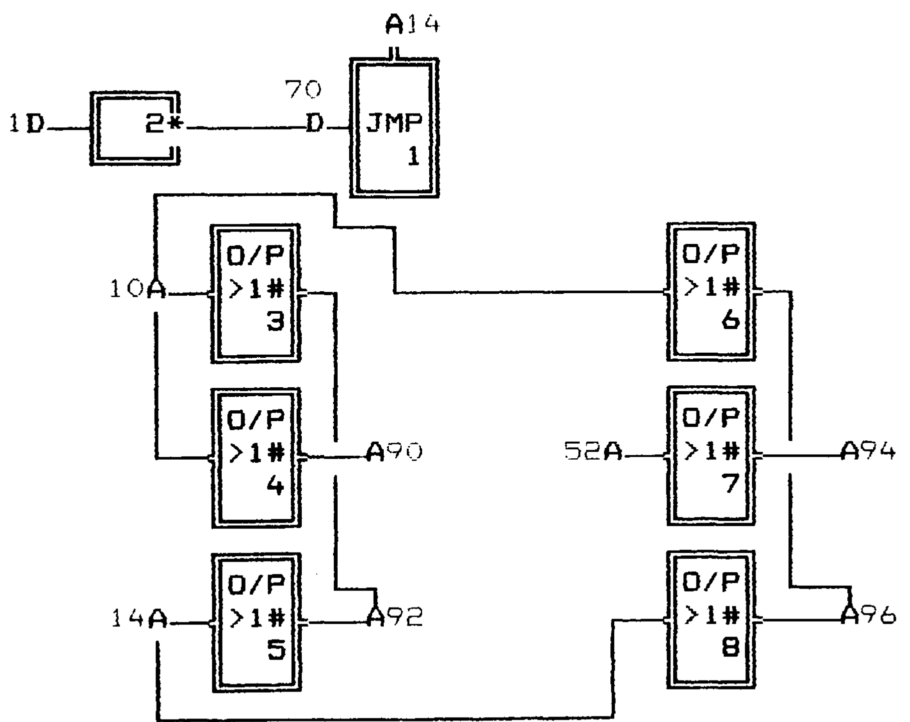
FUNCTIONS. Masterstation, Page 1

Index to Pocess Control Demonstration

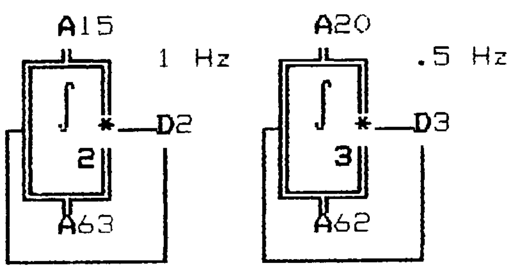
1	Index	21	Alarm Detector
2	Constants	22	Set-Point Calculator
3	PIA Initialisation	23	
4	Oscillators and Digital Inputs	24	
5	24 Hour Clock	25	
6	PID Test Page	26	
7		27	
8		28	
9		29	
10	Main Request Logic	30	
11	Pedestrian Crossing Logic	31	
12		32	
13	Subroutine caller	33	
14		34	
15	Lights sequence control subroutine	35	
16		36	
17		37	
18		38	
19		39	
20	Traffic Lights Mimic Diagram	40	Digital Outputs

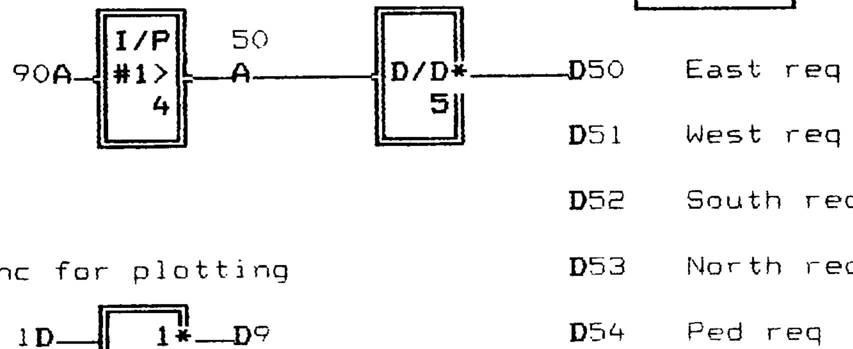
FUNCTIONS. Masterstation, Page 2



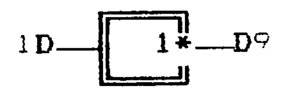

A14
70
1D
2*
D
JMP
1
10A
O/P
>1#
3
O/P
>1#
4
A90
14A
O/P
>1#
5
A92
O/P
>1#
6
52A
O/P
>1#
7
A94
O/P
>1#
8
A96

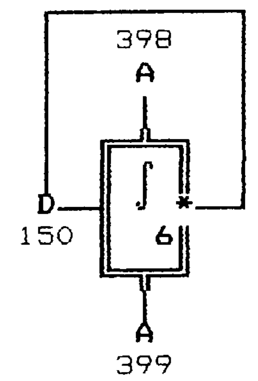
FUNCTIONS. Masterstation, Page 4





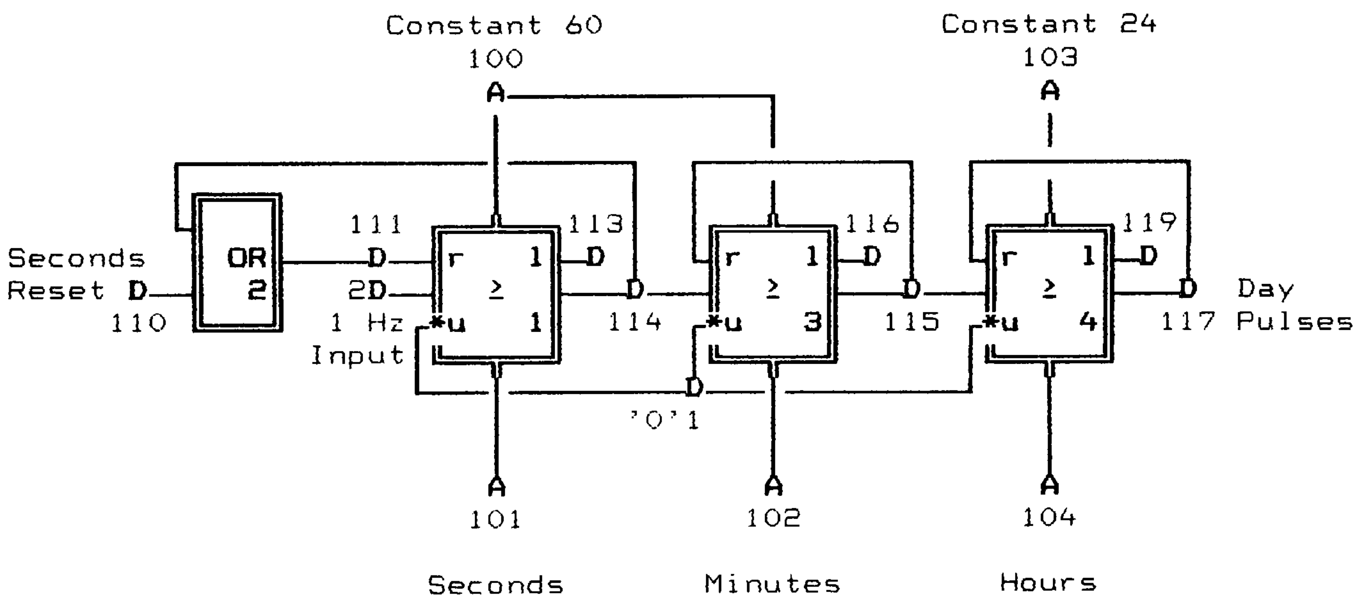
Sync for plotting



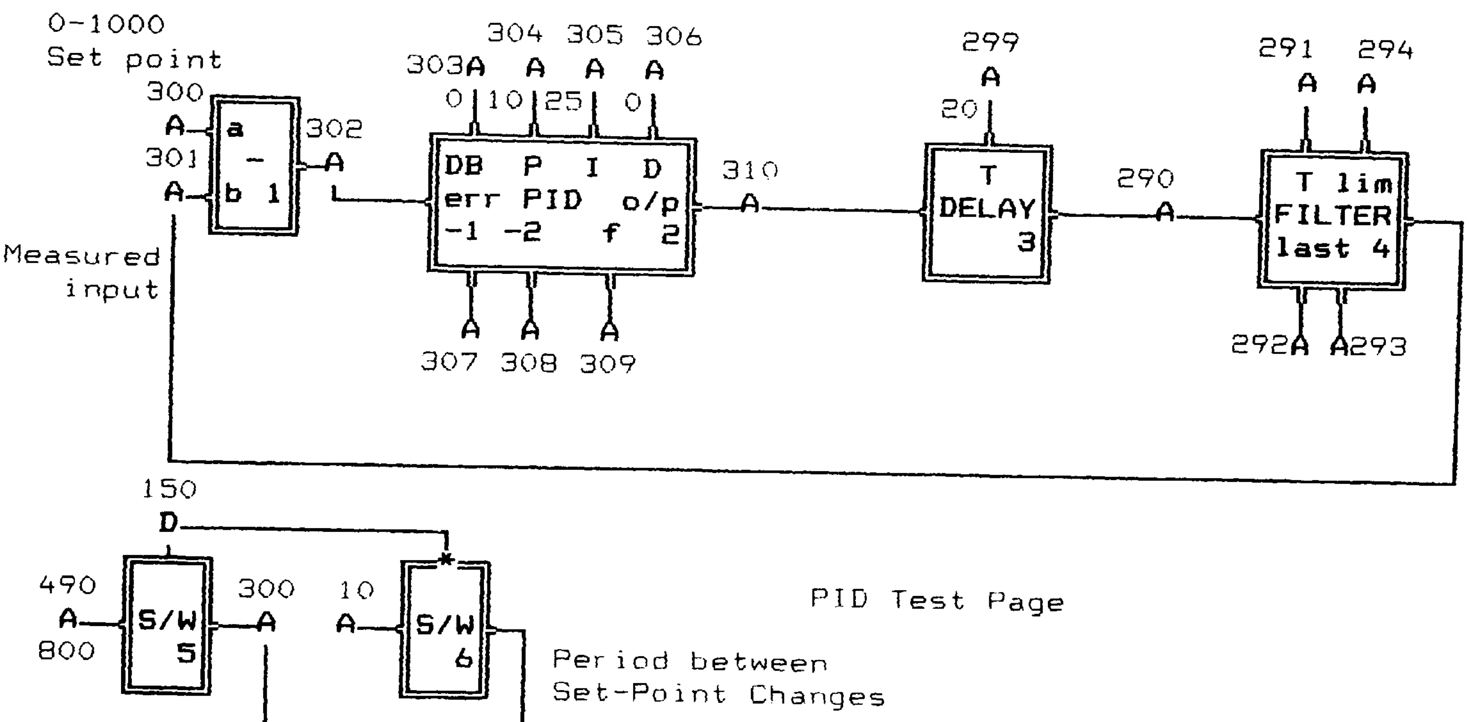


FUNCTIONS. Masterstation, Page 5

24 Hour Clock

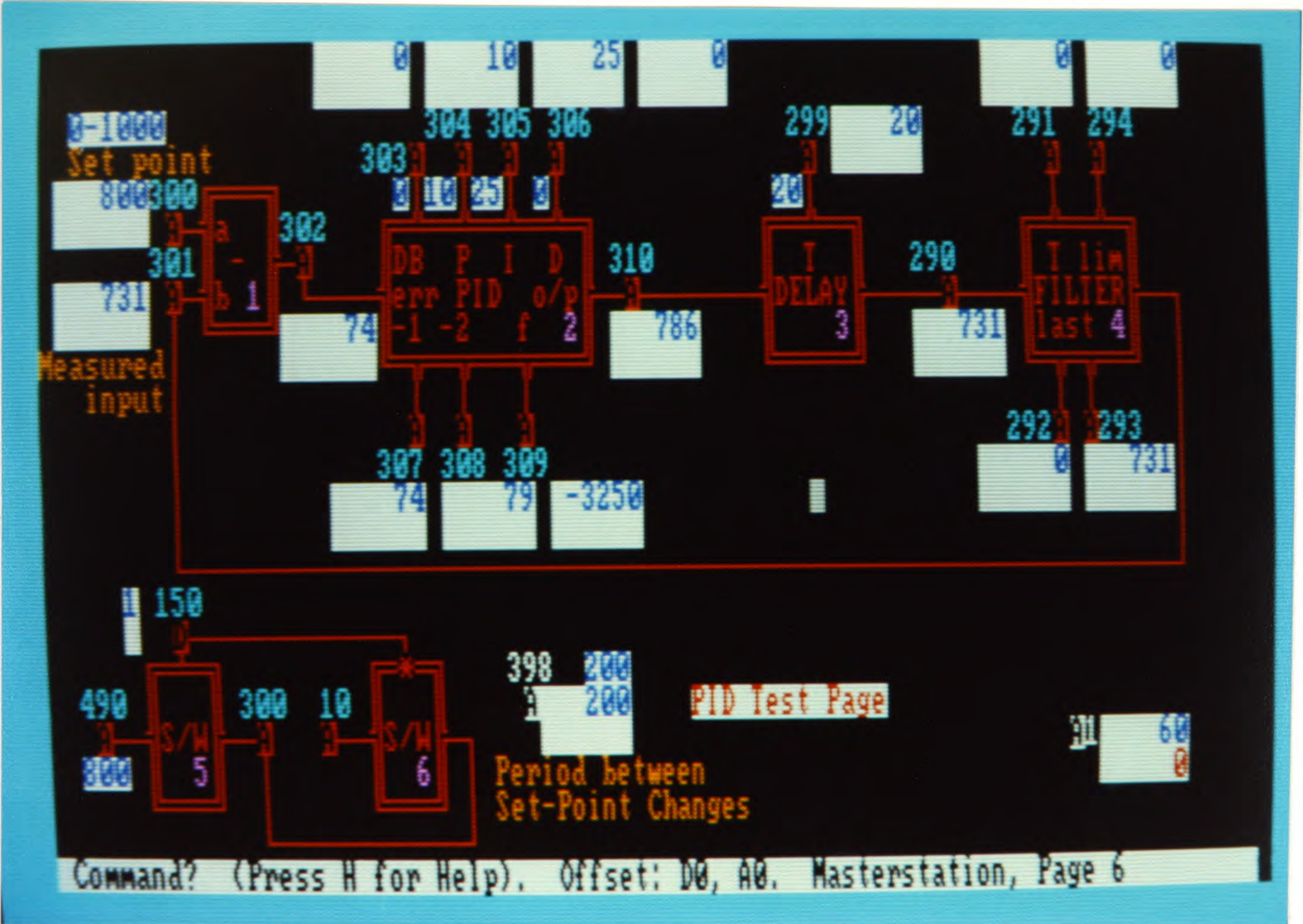


FUNCTIONS. Masterstation, Page 6

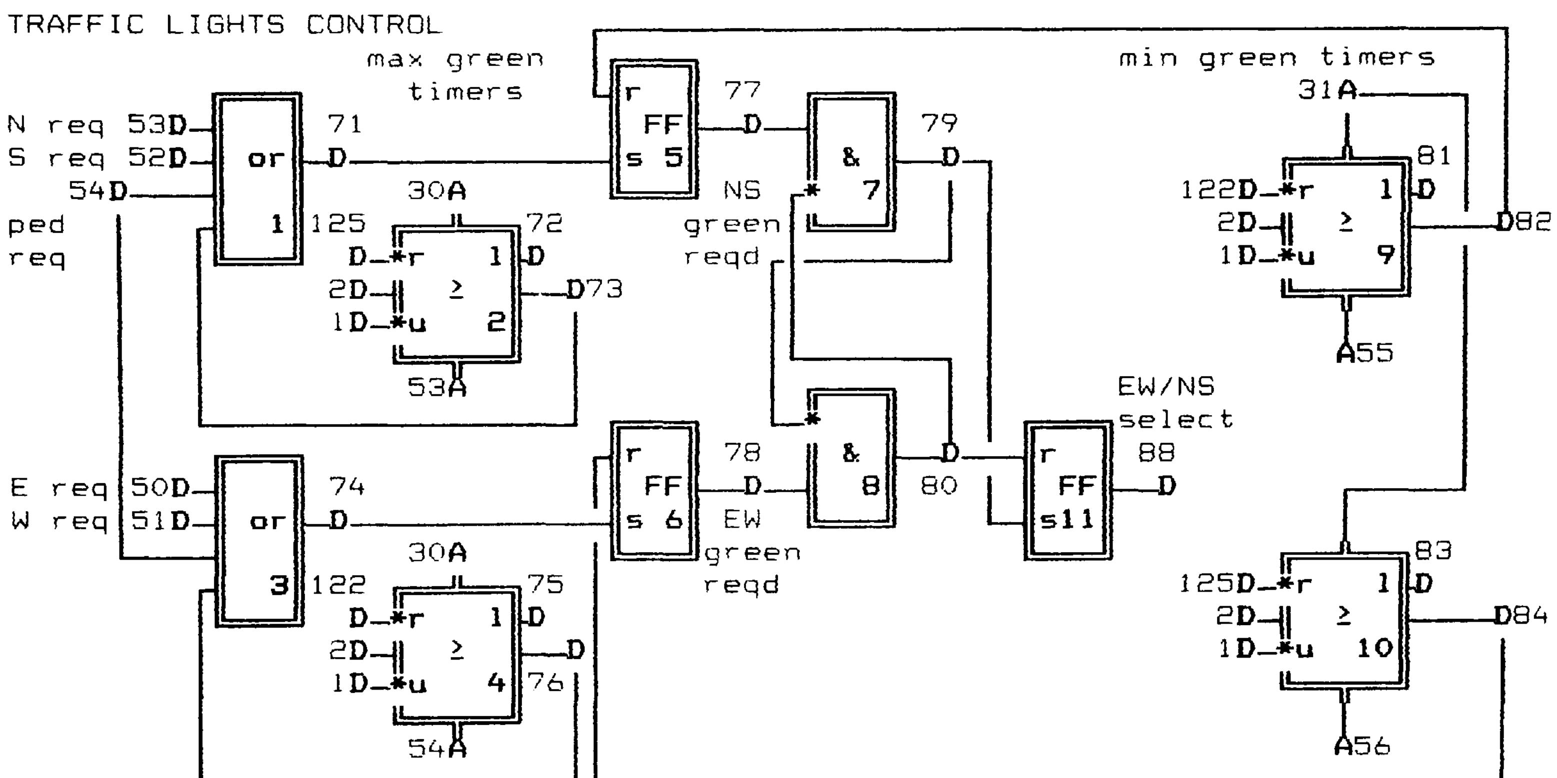


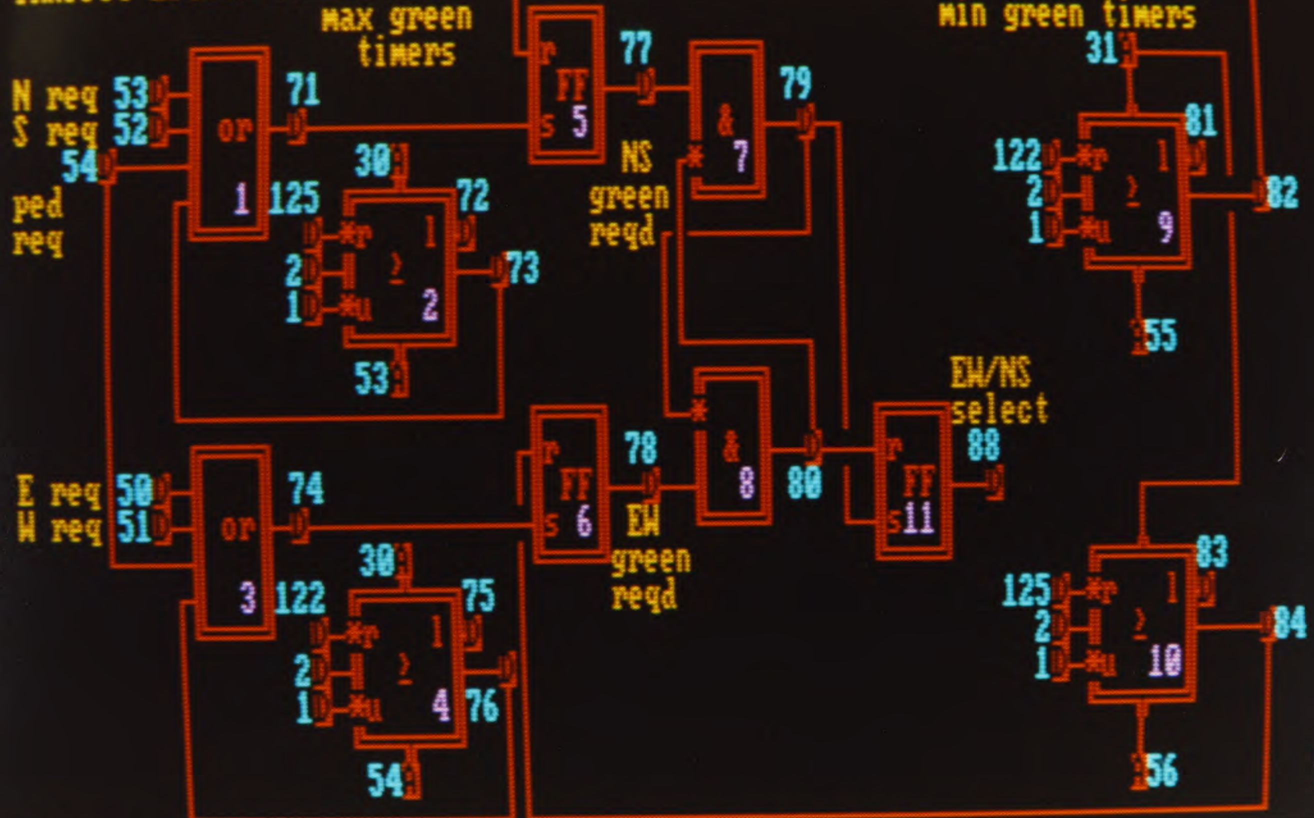
PID Test Page

Period between
Set-Point Changes


0-1000
Set point
Measured input
DB P I D
err PID o/p
-1 -2 f 2
I DELAY 3
I lim FILTER last 4
S/H 5
S/H 6
PID Test Page
Period between Set-Point Changes
Command? (Press H for Help). Offset: D0, A0. Masterstation, Page 6

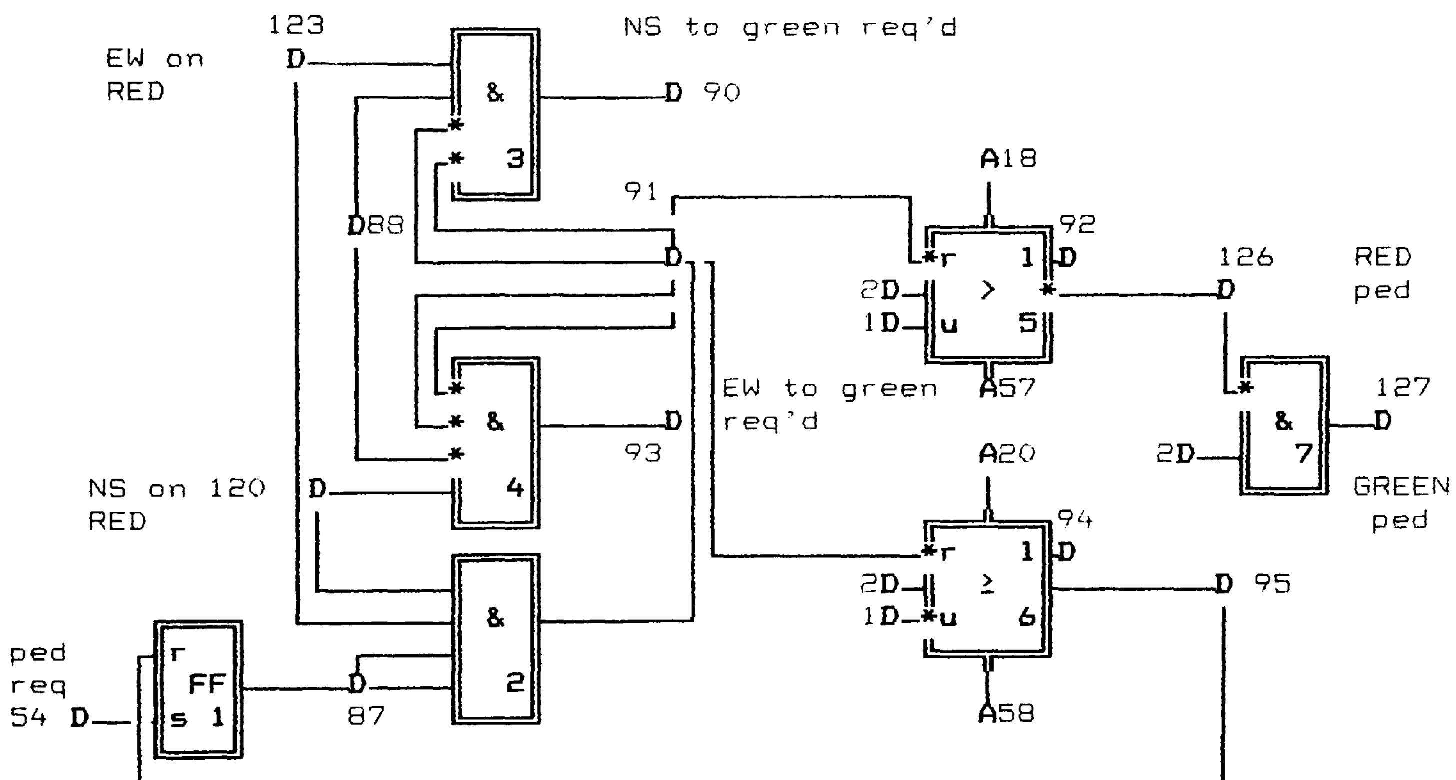
FUNCTIONS. Masterstation, Page 10



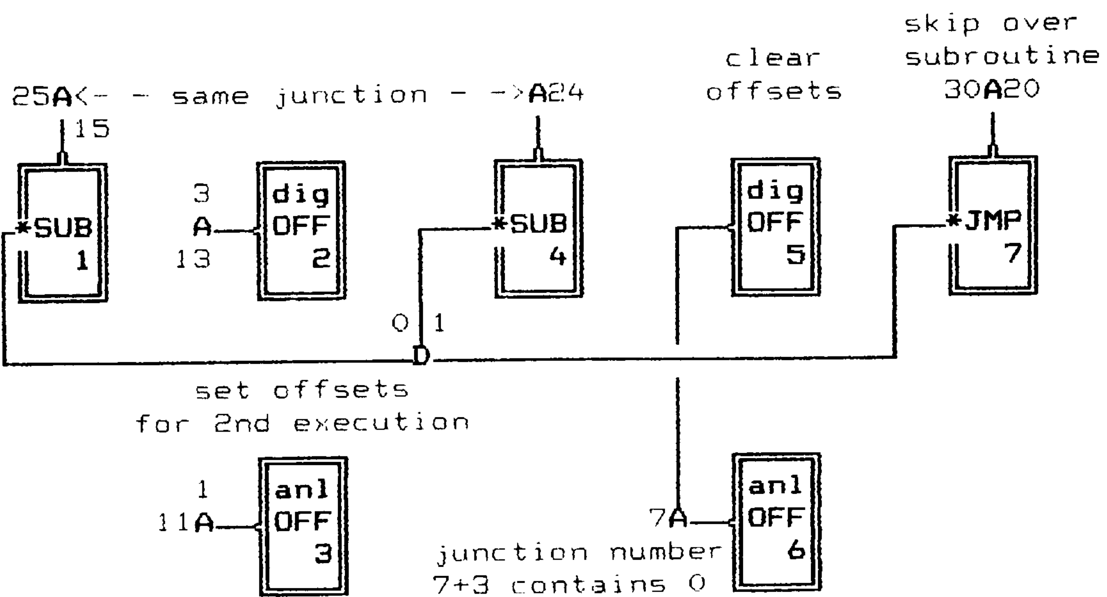


TRAFFIC LIGHTS CONTROL
max green timers
min green timers
N req
S req
ped req
E req
W req
NS green reqd
EW green reqd
EW/NS select
Command? (Press H for Help). Offset: D0, A0. Masterstation, Page 10

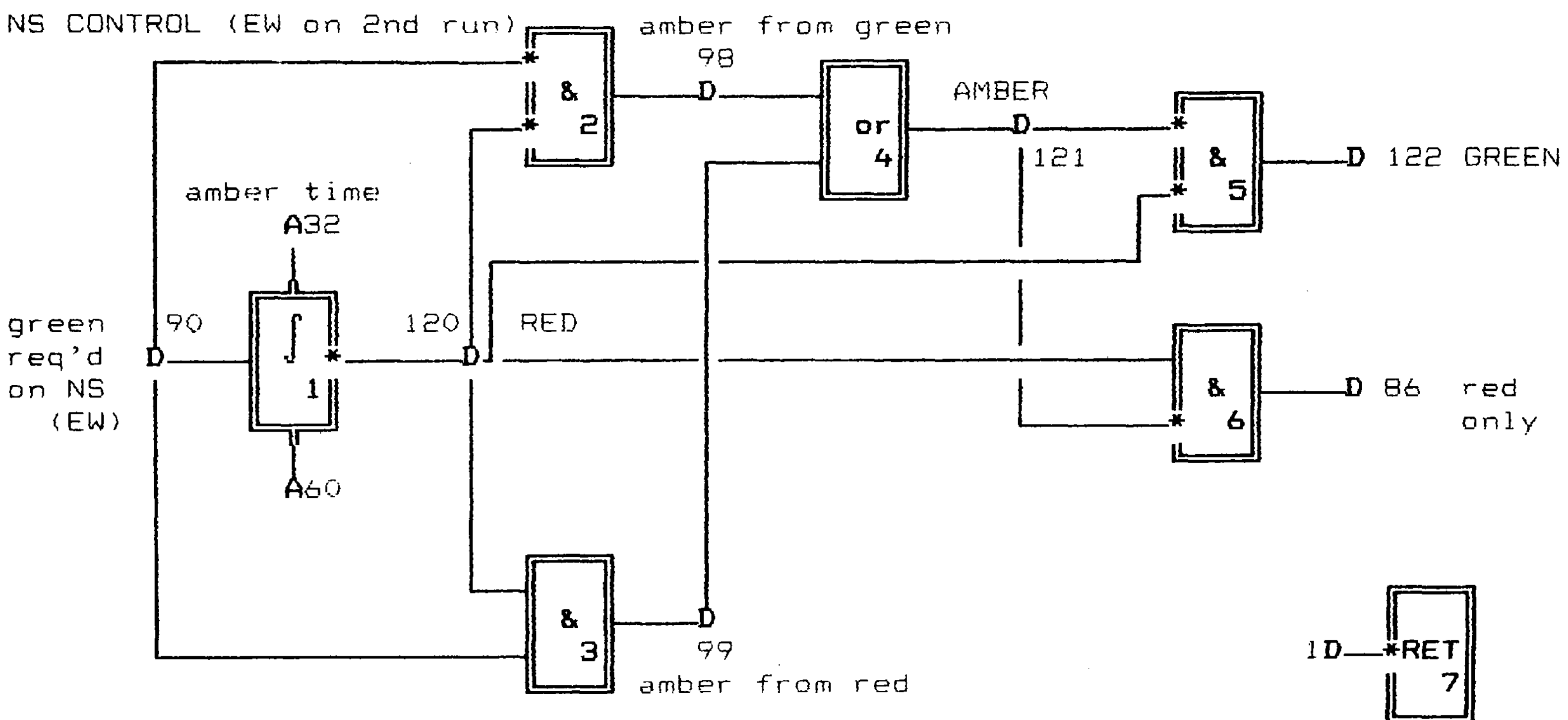
FUNCTIONS. Masterstation, Page 11



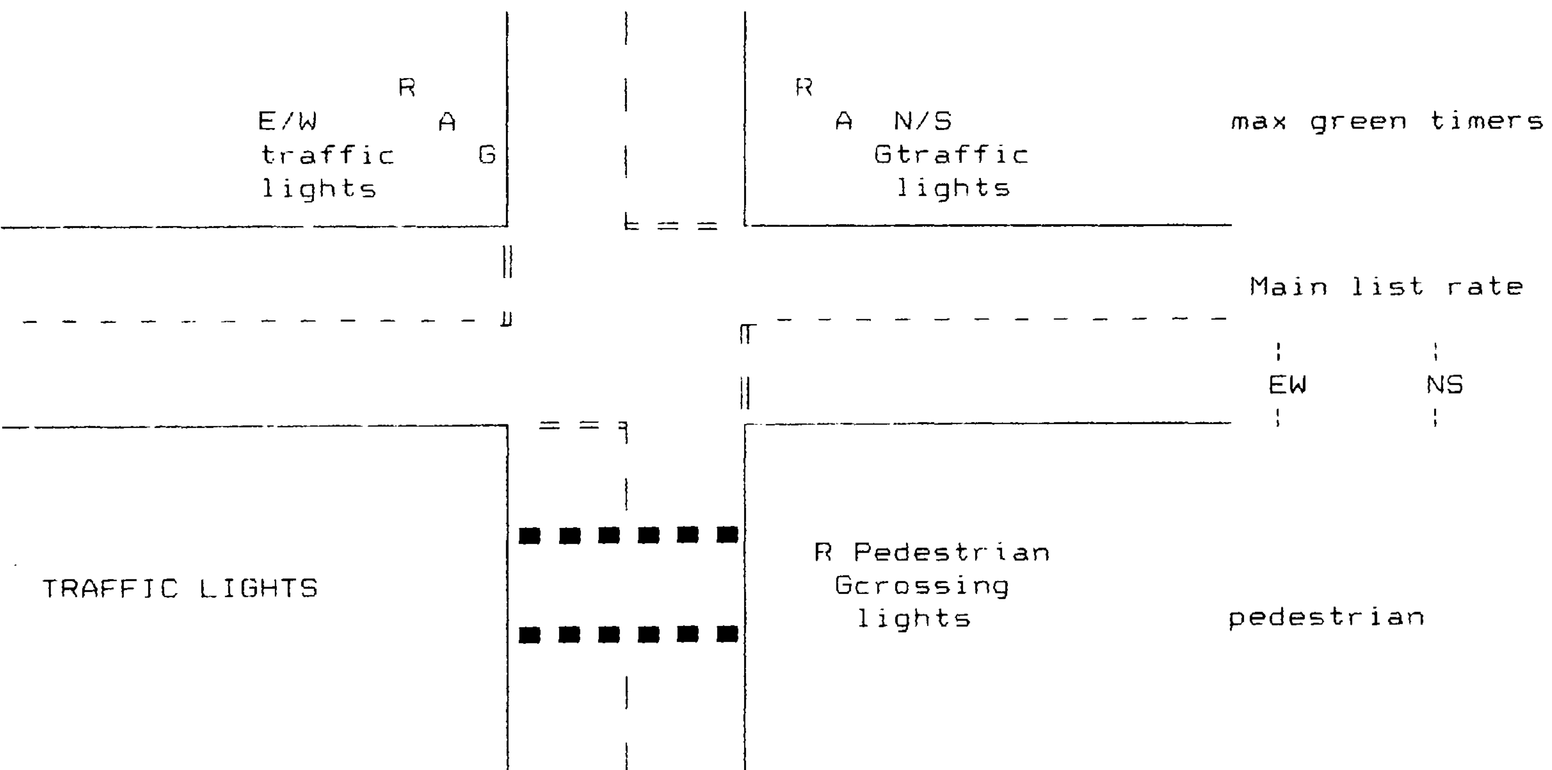
Execute page 15 as a subroutine, twice.



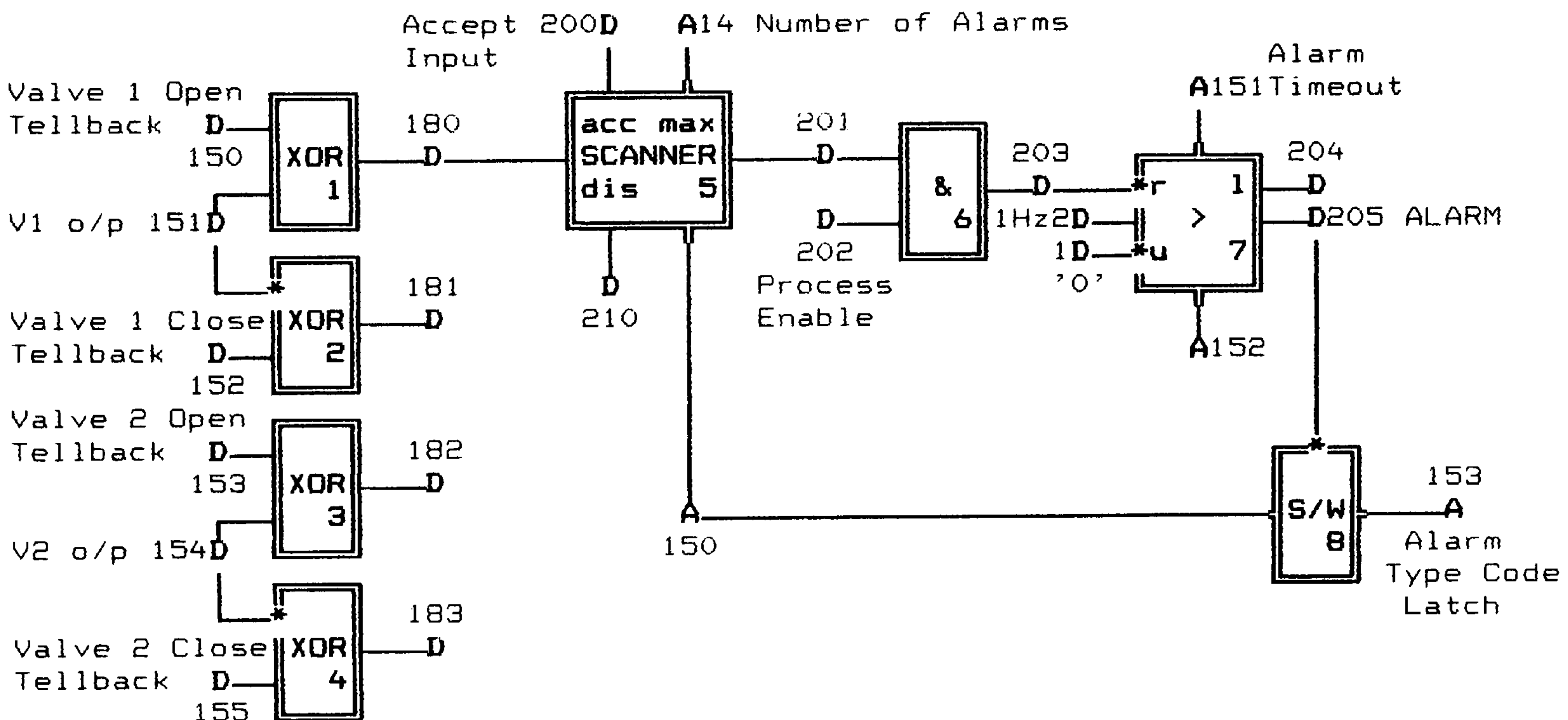
FUNCTIONS. Masterstation, Page 15



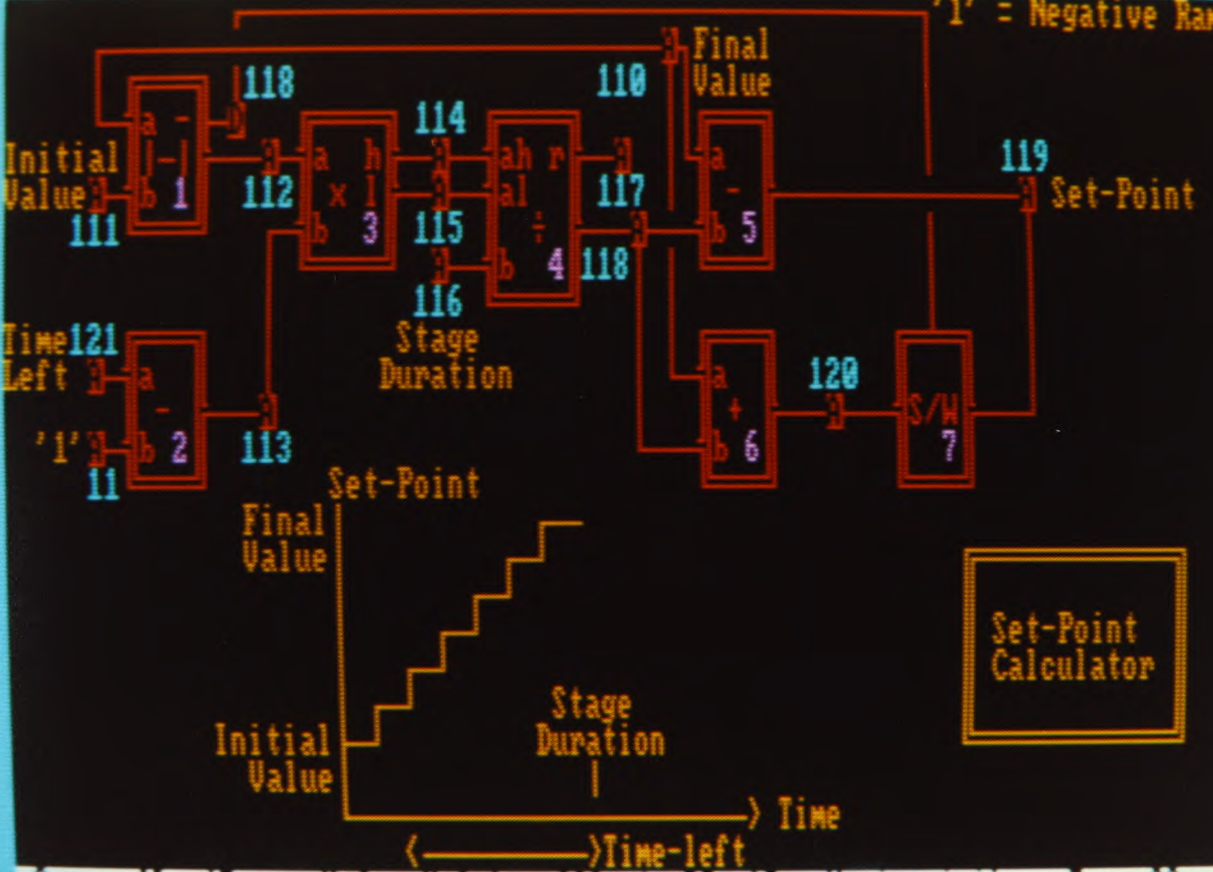
subroutine called from page 13. Executed with D3, A1 offsets 2nd time.


R
E/W
A
traffic
G
lights
R
A N/S
Gtraffic
lights
max green timers
Main list rate
EW
NS
TRAFFIC LIGHTS
R Pedestrian
Gcrossing
lights
pedestrian

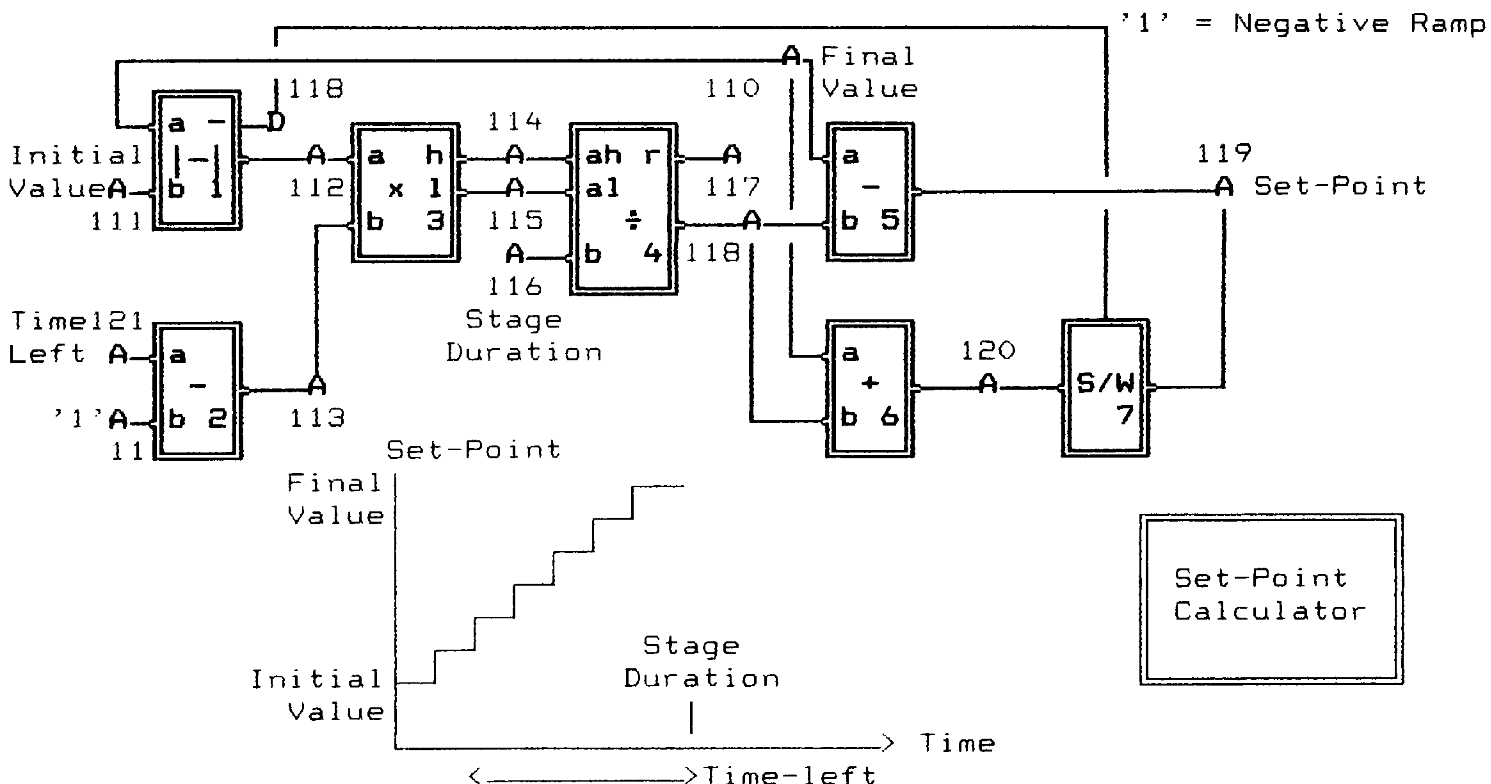
FUNCTIONS. Masterstation, Page 21

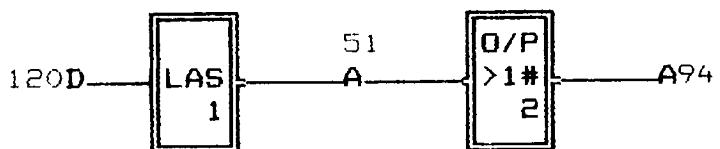


Alarm Scanner with Fault Timeout


'1' = Negative Ramp
Final Value
118
110
Initial Value
111
112
114
115
116
117
118
119
Set-Point
Stage Duration
Time Left
121
'1'
11
113
120
S/H
Set-Point
Final Value
Initial Value
Stage Duration
Time
Time-left
Set-Point Calculator
Command? (Press H for Help). Offset: D0, A0. Masterstation, Page 22

FUNCTIONS. Masterstation, Page 22




120D
LAS
1
51
A
O/P
>1#
2
A94

Masterstation Digital Junction Cross Reference

Digital 1 connected to 3:2 4:1 5:1,3,4 10:2,4,9,10 11:5,6 13:1,4,
Digital 2 connected to 4:2 5:1 10:2,4,9,10 11:5,6,7 21:7
Digital 3 connected to 4:3
Digital 9 connected to 4:1
Digital 50 connected to 4:5 10:3
Digital 51 connected to 10:3
Digital 52 connected to 10:1
Digital 53 connected to 10:1
Digital 54 connected to 10:1,3 11:1
Digital 70 connected to 2:1 3:1,2
Digital 71 connected to 10:1,5
Digital 72 connected to 10:2
Digital 73 connected to 10:1,2
Digital 74 connected to 10:3,6
Digital 75 connected to 10:4
Digital 76 connected to 10:3,4
Digital 77 connected to 10:5,7
Digital 78 connected to 10:6,8
Digital 79 connected to 10:7,8,11
Digital 80 connected to 10:7,8,11
Digital 81 connected to 10:9
Digital 82 connected to 10:5,9
Digital 83 connected to 10:10
Digital 84 connected to 10:6,10
Digital 86 connected to 15:6
Digital 87 connected to 11:1,2
Digital 88 connected to 10:11 11:3,4
Digital 89 connected to 11:2,3,4,5,6
Digital 90 connected to 11:3 15:1,2,3
Digital 92 connected to 11:5
Digital 93 connected to 11:4
Digital 94 connected to 11:6
Digital 95 connected to 11:1,6
Digital 98 connected to 15:2,4
Digital 99 connected to 15:3,4
Digital 100 connected to 5:2
Digital 103 connected to 5:1
Digital 104 connected to 5:1,2,3
Digital 105 connected to 5:3,4
Digital 106 connected to 5:3
Digital 108 connected to 5:4
Digital 109 connected to 5:4
Digital 111 connected to 5:1,2
Digital 118 connected to 22:1,7
Digital 120 connected to 11:2,4 15:1,2,3,5,6 40:1
Digital 121 connected to 15:4,5,6
Digital 122 connected to 10:4,9 15:5
Digital 123 connected to 11:2,3
Digital 125 connected to 10:2,10
Digital 126 connected to 11:5,7
Digital 127 connected to 11:7

Masterstation Analogue Junction Cross Reference

Analogue 1 connected to 2:12
Analogue 4 connected to 13:5,6
Analogue 10 connected to 2:21 3:3,4,6 6:6
Analogue 11 connected to 2:2 13:3 22:2
Analogue 12 connected to 2:3
Analogue 13 connected to 2:1,4 13:2
Analogue 14 connected to 2:5 3:1,5,8 21:5
Analogue 15 connected to 2:6 4:2
Analogue 16 connected to 2:7
Analogue 17 connected to 2:8
Analogue 18 connected to 2:9 11:5
Analogue 19 connected to 2:10
Analogue 20 connected to 2:11 4:3 11:6
Analogue 24 connected to 13:4
Analogue 25 connected to 2:22 13:1
Analogue 30 connected to 2:17 10:2,4 13:7
Analogue 31 connected to 2:18 10:9,10
Analogue 32 connected to 2:19 15:1
Analogue 33 connected to 2:23
Analogue 50 connected to 4:4,5
Analogue 51 connected to 40:1,2
Analogue 52 connected to 2:20 3:7
Analogue 53 connected to 10:2
Analogue 54 connected to 10:4
Analogue 55 connected to 10:9
Analogue 56 connected to 10:10
Analogue 57 connected to 11:5
Analogue 58 connected to 11:6
Analogue 60 connected to 15:1
Analogue 62 connected to 4:3
Analogue 63 connected to 4:2
Analogue 90 connected to 2:13 3:4 4:4
Analogue 92 connected to 2:14 3:3,5
Analogue 94 connected to 2:15 3:7 40:2
Analogue 96 connected to 2:16 3:6,8
Analogue 100 connected to 5:1,3
Analogue 101 connected to 5:1
Analogue 102 connected to 5:3
Analogue 103 connected to 5:4
Analogue 104 connected to 5:4
Analogue 110 connected to 22:1,5,6
Analogue 111 connected to 22:1
Analogue 112 connected to 22:1,3
Analogue 113 connected to 22:2,3
Analogue 114 connected to 22:3,4
Analogue 115 connected to 22:3,4
Analogue 116 connected to 22:4
Analogue 118 connected to 22:4,5,6
Analogue 119 connected to 22:5,7
Analogue 120 connected to 22:6,7
Analogue 121 connected to 22:2

Appendix D

Builder Screen Displays

The function building program includes 2 full screen displays which can be requested by the user. This appendix shows a screen copy of these displays.

```
                                builder
B - Block a junction.
C - Change under curser: over digital I/O  =  change inversion.
                         over function no  =  change function number.
                         over junction     =  change actual value.
                         over junction no  =  change junction number.?
                         over monitor      =  decimal <> hex.
D - Delete under curser: delete function with connections, delete connection
                         and reconnect, delete unconnected junction, delete
                         junction number, delete a monitor, delete comment.
E - Erase current page on screen and in the list.
F - Function insert:     only commands ^J and ^D are available during insert.
J - Junction insert.
K - Kill all monitors.
L - Load a new page and complete function list.
M - Monitor a junction:  over junction     =  monitor this junction.
                         over empty screen =  monitor off screen junction.
Q - Quit
S - Save current page and complete function list.
? - Display junction contents.

    Pressing any printable character will enter the character input mode.
      'Del' deletes character to left. Press CR to exit character input mode.
PRESS 'Esc' TO ABORT A COMMAND. Shapes drawn during the command will be deleted
```

```
                                   functions

Digital Logic:    1  - Buffer           2  - 2 i/p and        3  - 4 i/p and
                  4  - exclusive or     5  - 2 i/p or         6  - 4 i/p or
                  7  - flip flop        8  - latch

Misc. Digital:    9  - integrator       10 - scanner
                  11 - data divider     12 - logic assembler 13 - decoder

Analogue:         14 - compare =        15 - compare >        16 - compare ≥
                  17 - counter =        18 - counter >        19 - counter ≥
                  20 - sample and hold 21 - multiplexor      22 - demultiplexor
                  23 - PID controller   24 - filter           25 - BASIC array

Maths:            26 - add              27 - subtract         28 - abs subtract
                  29 - multiply         30 - divide
                  31 - bin to dec       32 - dec to bin       33 - define function

List:             34 - jump             35 - subroutine       36 - return
                  37 - digital offset   38 - analogue offset

Input/Output      39 - 1 byte input     40 - 2 byte input     41 - analogue input
                  42 - 1 byte output    43 - 2 byte output    44 - constant

 Enter funtion number
```

Appendix E

Published Paper

This paper was presented at the "EUROCON 84" Conference on

"Computers in Communication and Control"

Brighton, 26th to 28th September 1984.

A LOGIC BASED REAL-TIME LANGUAGE SYSTEM FOR PROCESS CONTROL

M A McCabe

The Polytechnic of Wales, UK

INTRODUCTION

The real-time microcomputer control of an industrial process involves many considerations which make the software design particularly difficult. There are usually many signals to be simultaneously monitored to control outputs which must often respond to these changes in a short period of time. As well as this real-time control function, there may be one or more operator interfaces which must handle complex displays and receive information usually via keyboards and switches.

The design of reliable and maintainable software to perform rapid and simultaneous control as well as providing an interface to the operator or manager, is complex and very prone to error. The task can be simplified by regarding the control section as a separate task to the man/machine interface and selecting the most efficient language for each. The two tasks can run concurrently and communicate via a shared data base.

An often underestimated fact is that process control software is not a fixed entity. Industrial processes often change due to extra equipment being installed or a revised idea of how the process can be controlled. The ease by which a program may be reliably changed and tested on site is reflected in the cost of such changes and must be considered during the initial design.

The sustem to be described in this paper seeks to reduce the software design effort by giving the designer a combination of two languages which may be used on-line. In particular, the real-time control language described offers a novel approach which overcomes many restrictions of conventional high level languages and makes use of logic diagrams with powerful debug and monitoring features. A subset of this system has been successfully used for several years; this paper seeks to outline a graphics based version which simplifies the conversion from ideas to a computer program.

LANGUAGE SELECTION

The selection of suitable languages is considerably eased by partitioning the overall task into three discrete sections as follows:

Operating environment. The operating environment is a real-time task scheduler which also provides facilities such as power-on system checking and background RAM/ROM verification. This section is not over-complex and will remain standard for many applications. A complied real-time language such as CORAL can be used but the operating speed of assembly language coding is better.

Operator interface. The operator interface requires a language which has versatile display capabilities and produces code which may be easily changed, preferably on-line. Execution speed is not a problem here so an interpreted language could be considered; this also offers considerable savings in software development and enhancement. A modern interpreted BASIC is quite suitable since it is now orientated towards easily producing interesting displays using graphics and colour. Despite its critics, BASIC is a firm favourite with many people because of its programming friendliness and built in "changeability".

Real time control. A language suitable for real-time control must be able to execute many tasks concurrently as well as providing some control over task priority. A general purpose compiled language could be used but this would require an experienced programmer for all but trivial tasks. On-site changes and fault finding would be very difficult and certainly too time consuming for the average process control engineer.

An original logic based language called LOGICON has been developed by the author to offer the less experienced programmer a powerful process control language based upon familiar logic circuit concepts.

BASIC LANGUAGE

The operator interface is controlled by an enhanced BASIC interpreter running as a task on a time shared basis under the control of the task schedular. When considering operator communications via a VDU, much of the time is spent waiting for an input from the keyboard. When displays are required, time, within reason, is not critical. Operating a BASIC interpreter for these functions on a time shared basis, even with an 8-bit processor, does not in practice unduly slow the system from the operators viewpoint.

The BASIC interpreter takes no direct part in the real time control; its sole task is to receive information and present information on the screen and via a printer. Communication with the LOGICAN control program is by accessing the junction database used by the control program.

A typical action may be to allow the operator to change the temperature set-point of a 3-term controller. The BASIC program would possibly display an option menu where one option would be to change this temperature. The option selection and operator input bounds checking would be performed by the BASIC program. The resulting set-point is passed to the real-time LOGICON program by loading the number into a mutually agreed location within the LOGICON junction database. The LOGICON program will automatically use this new value as the input to, in this example, its 3-term controller program.

LOGICON PHILOSOPHY

LOGICON is a real-time semi-compiled language based upon continuously executing a linear list of software functions. An on-line building program is run in place of the BASIC interpreter to build up a coded sequence of functions which are continuously being executed by a separate concurrent LOGICON interpreter.

A built in library of about 40 functions ranging from 2 input AND gates to 3-term controllers is available. To use a function, it must be drawn onto the VDU screen as part of a circuit "page" and its inputs and outputs connected to numbered junctions. Information is trans-

ferred from one function to another by joining the output of one function to the same junction as the input of the next function. This is of course the same as conventional hardware logic design but without any soldering.

Once a function has been entered, it is included into the function list which is being continuously interpreted. The LOGICON interpreter simply works down this list of functions on a regular time basis (say once every 0.1 seconds) and would normally interpret every function in that list. Each function appears to the interpreter as a code to identify the function type (OR gate, multiplier etc) followed by a list of I/O junction addresses in RAM. Data is taken from the junctions connected to the function inputs, processed by the specified function program, and the results written into the junction connected to the function output.

All connections are simply between a function and a junction. Since a junction is only a RAM location, all circuit connections may be monitored on line as a debugging aid.

LOGICON BUILDING PROGRAM

The LOGICON builder enables the engineer to build up a function list which is interpreted in real-time. The design is drawn onto a VDU screen using graphics to represent the functions, junctions and connecting wires. A keyboard and joystick are used to select the desired function and to "wire" it up to junctions on the screen. The screen positions of the functions, junctions, connecting wires or indeed any text comments about the operation of the circuit are determined by the engineer. The only information required by the LOGICON interpreter is the function code and the junction numbers to which each input/output line connects. The rest of the information drawn onto the screen is to permit easy understanding of the circuit and to provide a commented and readable printout for documentation.

Effectively then, a logic circuit is drawn onto the screen in a similar manner to hardware design using a CAD computer. Two additional items of information are required to complete a function linkage:

The function reference number. The function list is a linear sequence of function type and I/O information. This is divided into sections or pages for the convenience of displaying on a VDU. The list is executed regularly (by the interpreter) in a fixed order from the start to the end. The execution order is important to ensure that input changes work their way to the outputs as rapidly as possible and preferably in one pass of the interpreter through the list. The function number determines its position within the execution order for that page.

The type and reference number of a junction. A junction may be either an analogue or digital type depending upon the type of data being held. A digital junction can contain either a 0 or a 1 and would be used for example, to connect two AND gates together. An analogue junction can contain an integer from 0 to over 16 million. Each junction is given a reference number which is used to determine its actual location in memory.

The action of drawing a circuit onto the screen does not in itself enter the functions into the function list being interpreted. A complete sub-circuit can be constructed before being included as part of the function-list which avoids excessive list manipulation and spurious outputs due to incomplete circuits being executed.

Once a page of logic has been constructed, it may be stored on floppy disk to enable another to be constructed or loaded from disk. The maximum number of pages that can be included in a system is largely determined by disk capacity, but a maximum of between 40 and 60 permits quite large controllers to be designed. This page information is not required by the interpreter and so disks need not be on-line once an applications has been fully developed.

Being able to recall and edit pages of logic from the disk enables a designer to build a library of commonly used circuits. For example, a set-point ramp generator may be developed, tested and saved. Any application requiring this facility will simply necessitate loading this page into the LOGICON building program, re-numbering the junctions and saving as part of the new application.

LOGICON INTERPRETER

The LOGICON builder constructs a list of function descriptions in memory which must be executed by the interpreter program. The format of a function within the list is very simple and may be illustrated by considering a 2 input AND gate. This occupies 7 bytes as follows:

(a) 1 byte function code for an AND gate,
(b) 2 byte "input A" junction address,
(c) 2 byte "input B" junction address and a
(d) 2 byte "output" junction address.

The junction address is in fact the relative address of the RAM location used for input or output data. Included in this address is a reserved bit to indicate when the data should be inverted before use or inverted before output. This inversion capability enables a basic AND gate to be used as a NAND or in fact an OR gate.

The function list is a continuous linear sequence of such function descriptions. The interpreter simply works down this list, reads the next function code and jumps to a function handling routine determined by that code. The function handling routine will execute a predetermined function (in the above case, a digital AND function) using the data locations specified in the function list. After executing this function, control is passed back to the interpreter which proceeds to the next function description in the list.

Function types. The digital functions provided include:

(a) simple logic; 2 or 4 input AND, OR, EXOR, buffers etc.,
(b) memory functions; flip-flops and latches,
(c) integrator.

Analogue functions are described in the same way except that the provision for data inversion with an analogue junction is not included. All analogue functions operate with integers to 3-byte precision - it is rare that this limitation becomes a problem in process-control with sensible number manipulation.

The type of analogue functions provided include:

(a) arithmetic functions; multiply, divide, square root etc.,
(b) comparisons; equality, greater than etc.,
(c) counters/timers,
(d) data tables,
(e) multiplexors, demultiplexors,
(f) I/O; digital and analogue with filtering,
(g) special purposes; 3-term controller etc.

Functions may consist of a mixture of digital and analogue inputs and outputs. Compare functions for example use 2 analogue junctions for input data and use a digital junction for the result.

Although the present list of functions provided will meet the demands of many applications, it is possible to extend the system if other special purpose functions are necessary.

LOGICON CIRCUIT DESIGN

Many of the functions operate in an analogous way to well known logic elements; this enables boolean description and simplification to be applied prior to circuit design. Many constraints of hardware logic design obviously do not exist however, such as fan-out limitations, power supply loading and additional component cost. Other hardware problems such as race conditions are reduced to a predictable effect which may even be useful.

Consider an oscillator for example - this may be constructed by linking the output from an inverter back to its input. Since the function is executed on a regular basis, say every 0.1 seconds, then the junction used as the common point between input and output will change state each scan of the functions. This junction will therefore oscillate with a frequency of 5Hz.

Timing circuits are formed by using this oscillator junction as an input to a pre-settable up/down counter function. Very long programmable time delays may be produced by cascading counter funtions.

Response time. With many industrial control situations, the major part of the control will tolerate a delay of say 0.2 seconds maximum between an input change causing an output to change. This means that the bulk of the process control circuit can exist in the main list of functions as previously described. Sometimes however, a faster response time of several milliseconds is required to one or more particular inputs. To accommodate this, a group of reserved pages within the function list are executed by the interpreter at a programmable rate which may be as frequent as 100Hz or greater. Obviously the number of functions that can be included within these pages has to be limited at high execution rates so that the main function list can be fully executed in the given time interval. Fortunately, in practice, many process control applications do not require excessive quantities of high speed control logic.

Multi-plant control. Often an application demands the control of several similar items of plant. One could simply design the logic for one item and repeat it for the others but this is inefficient for many reasons. Special purpose functions are provided which permit pages of logic to be called as subroutines - this is of little use however if the same data is being processed each time through. Before calling a subroutine to control another similar item of plant, an offset function may be executed which will cause a programmable number to be added to every subsequent junction reference number until a new offset function is executed or the function list re-started. Each run through a subroutine can be organised so that a different block of junctions is accessed.

A practical application would be the control of 20 conveyors; the design can be implemented and tested for one item with the control functions terminated by a "Return" function. Twenty subroutine functions, each preceded by offset functions, will duplicate the control for each conveyor using data unique to each conveyor.

LOGICON PROGRAM DEVELOPMENT

One of the most difficult problems associated with process control software development is how to test the program when incomplete or no hardware input/output exists. The consequences of having little control over inputs and outputs is also very apparent when commissioning or modifying hardware and software on-site and/or on-line.

Junction forcing. The LOGICON system attempts to reduce these debugging problems by building in a forcing ability into every analogue and digital junction. Each junction contains an actual value, a forced value and a flag to indicate which value is to be taken as an input to a function. All function outputs only affect the actual value stored in a junction. Once the building program forces a junction during program development, any function using this junction for its input data will take the forced value rather than the actual value.

A simple example will illustrate this versatility:

> A section of the control program is checking several digital inputs being in an alarm state. If so, then an alarm siren should sound.

To test this program on-line, the output junction corresponding to the siren should be forced to an off state to avoid unnecessary alarms. This junction is also displayed on the logic diagram drawn on the VDU screen. Combinations of the relevant digital inputs may then be forced to alarm states and the effect upon the output observed by watching the actual junction value corresponding to the siren on the screen.

All of this may be done without leaving the keyboard and screen and in confidence that any input can be simulated and checked whilst maintaining outputs driving motors and valves, for example, in a safe condition.

The ability to dynamically observe the contents of selected junctions is a very powerful debugging aid. When a page is displayed on the screen, the contents of one or more junctions can be displayed in real-time in their correct circuit locations; other junctions not on the current page display can also be observed.

SUPPORT ENVIRONMENT

There are several programs which must run concurrently to permit control logic to be entered and tested, as well as providing an operator interface tailored for the specific applications. These may be summarized as:

(a) LOGICON interpreter,
(b) System verification (e.g. ROM/RAM checks),
(c) LOGICON builder and
(d) BASIC interpreter.

The interpreter and system verification programs operate continuously via the controlling multi-tasking executive. The BASIC interpreter is also continuously running and outputs information such as alarm messages to a screen RAM buffer. If the LOGICON builder is currently being used then keyboard input is directed to that program and the visable display is created by it. After leaving the builder program and returning to the BASIC applications program, keyboard input is directed to the BASIC interpreter and the BASIC interpreter screen RAM is displayed. The BASIC screen may now contain information generated whilst the LOGICON builder was active.

The system verification program consists of program code sumchecks, function list sumchecks and simple non-destructive junction data base RAM tests. Any faults detected will be communicated to the junction data-base by loading values into reserved junctions. If a memory failure still permits the interpreter to run, then a safe shutdown and alarm may be generated by testing these junctions.

CONCLUSIONS

The combination of the LOGICON process control language and a standard BASIC interpreter offers the process control engineer powerful on-line design and debug tools. Each language is used for the work for which it is most suited; this avoids the compromises necessary when trying to use any single general purpose language.

The ability of the modern BASIC to produce interesting displays relatively easily and without lengthy compilations can be used to advantage in many applications. The presence of an interpreter also permits enhancements to be included and tested throughout the life of the equipment without resorting to expensive development systems.

LOGICON itself is specifically oriented towards the demands of industrial process control. The logic designer is isolated as far as possible from the complexity of simultaneous real-time control and is presented with a friendly graphics based program input and test environment. This contrasts with the rather stark alphanumeric listings presented by a more conventional language. The ability to display, change and monitor a control program in an easily understood form considerably shortens the time necessary between formulating a control algorithm and successful implementation. If, as is often the case, full input/output hardware is not available, the control program can still be fully tested without resorting to traditional "boxes of switches and lights" and time consuming development system emulation.

REFERENCES

1. EEUA "Guide to the engineering of microprocessor based systems for Instrumentation and control". Engineering Equipment Users Association (1981).

2. IBM "PC Technical Reference". IBM Corporation (1983).