# ENGINEERING IMPLICATIONS OF PROGRAMMABLE ELECTRONIC COMPONENTS

D. Aspinall*

Rapporteur: Dr. P.A. Lee

## 1. Introduction

When the microprocessor was introduced into the electronic engineer's component repertoire in 1972, it heralded a new era of electronic equipment implementation possibilities [1]. Equipment for communications, instrumentation, and process control for the enhancement of performance and to improve the ergonomics of domestic appliances (white goods) had previously been based upon electrical circuits in which components were interconnected by simple conductors, usually copper wire. They could now be implemented by a combination of interconnected programmable components plus the programs and data structures within these components. In designing, commisioning, and maintaining such equipment, the engineer finds himself requiring to adopt the attitude of a computer engineer and to learn from the experiences of twenty-five years of computer system development and use.

Consider, for example, the products of a manufacturer of electronic instruments. These range from basic measuring instruments such as the voltmeter, through multi-range meters to signal generators and signal processors for spectral analysis.

The manufacturer should develop a strategy for specification, design, production, and maintenance which is common to all products in the range. It has been recognised that the key to any such strategy is a structured representation of the equipment requirements which can be used as the basis for the documentation necessary for the subsequent stages in the product production and use [2]. For a typical measuring instrument, the requirement definition may appear as in Figure 1. On the front panel of the instrument will be the controls to enable the user to set the desired range of operation and select the facility required. Activity 1 must interpret these commands and arrange for their display back to the user, and also for them to control the other activities in the instrument. The quantity to be measured will usually be in an analogue or continuous form, and activity 2 is required to convert into digital form. The range and performance of the conversion is controlled from activity 1. The digital data then passes to activity 3, where the measurement process is carried out. The processed data passes to activity 4, where it is further processed to be presented in a suitable form of visual display, acceptable to the user. One important activity which is not shown

* This contribution is based upon a lecture delivered at the CREST-1TG Advanced Course "The Microprocessor and its Application" Sept. 1977 at the University College of Swansea. Published by Cambridge University Press, summer 1978.

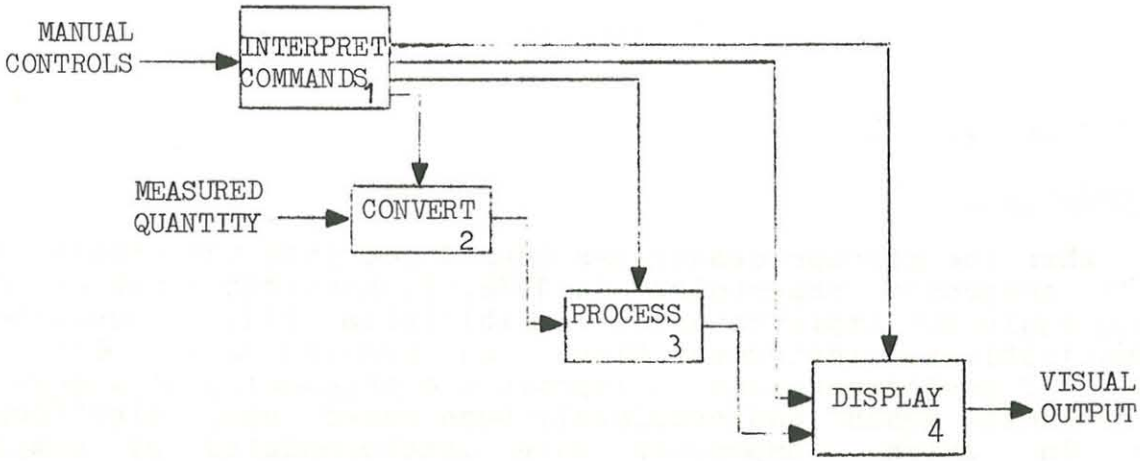



Figure 1 : Activities in a Measuring Instrument

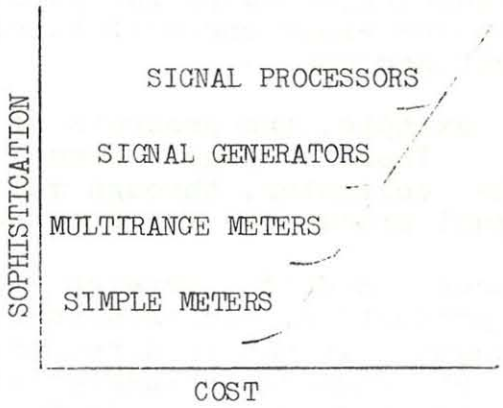



Figure 2
Instrument Range

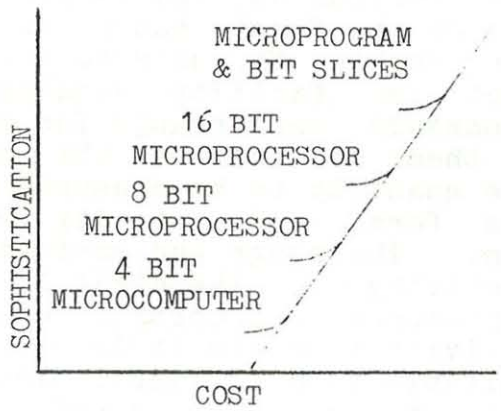



Figure 3 : Implementation by Single Processing Element

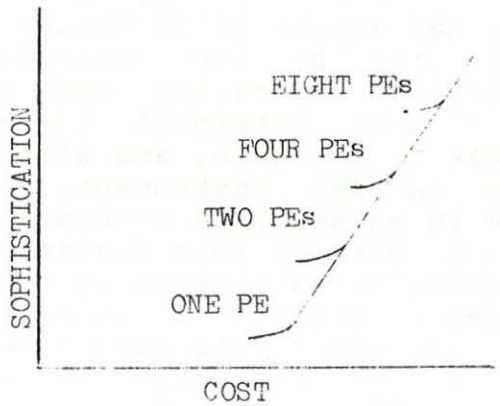



Figure 4 : Implementation by Many Processing Elements

in the diagram is that of self-test. The low capital cost of the instrument cannot justify the expense of a team of on-site maintenance engineers. Facilities for rapid fault diagnosis by the user must be included within the instrument to provide high availability. Such facilities, together with further elaboration of the various activities in Figure 1, should be represented within the structured requirement definition description.

On scanning the range of instruments, it is evident that the sophistication of the equipment increases from the simple meter through to the complex signal processor, and it is not surprising that the cost of a piece of equipment rises with its sophistication, as shown in Figure 2. A graph of this type is not unfamiliar to the manufacturer of computing equipment in attempting to meet the varying requirements of different users. The computer industry realised that a range of computer mainframes, each based upon a single sequential processor, could meet such a demand curve. At the bottom of the curve, the processor is based upon low-cost logic circuits. On moving up the curve, higher performance logic circuits are used, and there is an increase in the parallelism of each processing step. At all points in the range there is a single processor which obeys a sequential program.

An instrument manufacturer could adopt a similar strategy. The activities of Figure 1 could be implemented as one program to be processed in a single processing element. At the bottom of the range, the processing element could be a single micro-computer component comprising a 4-bit word processor plus ROM and RAM (Figure 3). Moving up the range, the processor could become more powerful, as demonstrated by its 8-bit or 16-bit word processing ability. For the highest members of the range, specialised processors could be manufactured from the high performance bi-polar semiconductor 'bit slice' components, supported by microprogram controllers possibly exploiting the Uncommitted Logic Array (UCLA) or Programmable Logic Array (PLA). At all points in the range the activities are implemented as one sequential program in a single sequential processor. The high performance processor may be time-shared amongst several programs, but at any one instant of time one, and only one, program step is being obeyed.

In adopting any strategy, the manufacturer should ensure that the tools, such as language and development systems to fashion the different instruments, should be common, to minimise the cost of retraining the engineers and to maximise the use of portable programs up and down the processing elements of the range.

The cost of these programmable electronic components is directly dependent upon the production volume achieved by the semiconductor manufacturer. The highest demand and, hence, production, will be for the less complicated micro-computers which find application in the consumer market.

Thus, the component for the bottom of the range should be exceedingly low in cost. This economic factor suggests that the instrument manufacturer may have an alternative strategy. Instead of basing the implementation upon a single sequential processing element, the alternative is to use a plurality of low-cost

processing elements interconnected to achieve the required instrument. The number of such components employed depends upon the level of sophistication, as shown in Figure 4.

Before such a strategy can be adopted, the manufacturer needs to be reassured of two significant points; first, that it is possible to identify those activities which can be efficiently assigned to separate processing elements, and secondly, that the physical method of interconnecting the elements is reliable and does not impose formidable overheads which cost too much when compared to the alternative use of single, more powerful, processing elements.

## 2. Logical Design to Achieve Parallelism

The designers of the logic circuits within leading-edge mainframe computers have realised the value of concurrent operation of processing elements to achieve a high processing rate, [3]. Circuit techniques were developed to provide the protocols at the interconnection of separate elements, to ensure correct synchronism and to guard against indeterminancy and deadlock. The analysis of these techniques can be based upon the contribution by Petri [4], Holt [5], and have been well summarised by Noe & Nutt [6] and Dennis [7]. The principle is that the forward flow of control, from element to element, must be associated with the backward flow of signals to acknowledge receipt of the forward command and completion of the required action. A typical example of this technique is the T-module of Dennis (Figure 5). The processing element A is idle, in a dormant state, until it receives a command BEGIN from the T-module. On completing its task, the element sends an END signal to the T-module and returns to its dormant state. The T-module will only give the BEGIN command if it has received a backward ACKNOWLEDGE signal from the next T-module in the sequence and a forward READY signal from the T-module which precedes it in the sequence. On receiving the END signal from the element, the T-module sends both a backward ACKNOWLEDGE signal to the preceding T-module and a forward READY signal to the next T-module in the sequence. Such procedures ensure the reliable operation of concurrent actions. A series of T-modules with their associated processing elements is often termed a 'pipeline'. Within a pipeline, it is assumed that there will be more than one token, indicating more than one action occurring concurrently. Such a pipeline could be implemented by using a number of separate processing elements, each based upon a micro-computer. The T-module procedure and associated action A could be programmed into the element and Ready/Acknowledge circuits provided through input/output ports. A mechanism for passing data from element to element could also be devised.

Having found a technique for managing and implementing concurrency the next problem is to identify those situations which naturally spawn many tokens at once. The question is : How can a designer identify parallelism in a sequential program?

## 2.1 Evolution of Parallelism from Sequential Programs

The logic circuit engineers within the instrument manufacturers have developed sound disciplines for the design of circuits [8]. When they were required to implement their design in the programs and data structures of a micro-computer, they accepted principles of structured programming [9], for they seemed a natural extension of familiar procedures and disciplines. When writing a program, it is natural to think sequentially and produce a solution as a sequence of program steps. It is natural to turn to such programs and attempt to identify parallelism which can exploit more than one processing element to execute the actions, [10]. The parallel construct, par begin ..... par end, can be used where appropriate. A single processing element will obey the main sequential thread of the program whilst an extra processor is added, to be invoked whenever the parallel construct is encountered within the main sequence. The number of extra processors is not equal to the number of parallel constructs used within the sequential program, but equal to the greatest number of extra parallel procedures invoked within a parallel construct.

Techniques to provide concurrency, such as the pipeline or overlap, do not naturally follow from the analysis of a sequential program. The reason is that the single locus of control, or token, in a sequential program cannot be readily replicated to provide the many tokens within such concurrent systems. The parts of a program which may provide an opportunity to replicate the locus of control are the repetitive clauses such as While ... Do or Repeat ... Until. By examining the flow of control through a repetitive clause, such as that shown in Figure 6, it is possible to imagine that, since the locus passes round the loop many times before passing to the next clause, the actions within the loop may be executed in separate processing elements, arranged in a pipeline. Consider what happens if each action $Z_0, Z_1, Z_2, Z_3, Z_4$, is assigned to a separate processing element. The locus of control enters the qualifier action within the main processing element. If the qualifier is true, then a token passes to processing element $Z_0$. On completion of the action $Z_0$, the token passes to processing element $Z_1$ while $Z_0$ pretends that it has received a second token and repeats its action concurrently with the action in $Z_1$. and so on. Many tokens will exist within the pipe until the token which leaves $Z_4$ and, on passing back to the qualifier action finds the qualifier false, causes the clause to be exited. The tokens behind it in the pipeline are invalid. At best, they have to be destroyed, by some mechanism, whilst there is a good chance that actions took place which should not have been allowed, and which may have corrupted data.

It seems that the only chance of finding concurrency within a repetitive clause is if the sequence of actions within the loop can be arranged so that the first one, $Z_0$, is the only one which modifies the qualifier [10]. It is then necessary to provide a new construct Con begin .... Con end and invoke the use of the Dennis T-modules to construct the loop, as shown in Figure 7. The operation is as follows: the locus of control of the main sequential program enters the qualifier action Q. If true, the locus passes to $Z_0$. On completion of $Z_0$, the locus passes to the CONBEGIN

construct. The locus is immediately returned to the qualifier action Q and two tokens are generated; the first passes directly to the CONEND construct, to be counted. This count will equal the number of tokens entering the pipe. The first token will eventually pass through all the remaining actions and finally emerge from the last T-module to enter the CONEND construct. These tokens are counted to determine the number leaving the pipe. Meanwhile, the locus of control will have passed through the action $Z_0$ several times, until the qualifier is false; when the locus passes to the CONEND construct, where it waits until the total number of tokens leaving the pipe is equal to the number which entered. At this instant, the locus passes to the next action within the main program sequence.

If the time to complete one traverse of the loop in Figure 6 is equal to T, and the total number of traverses is equal to n, then the time to execute the clause is nT. If the time to traverse Q and $Z_0$ is t, and the time to pass through the pipe $Z_1$, $Z_2$, $Z_3$, $Z_4$ and the constructs of Figure 7 is T', then the total time to execute the clause is nt+T'.

$$\text{since} \qquad t \ll T$$

$$\text{then} \qquad nt + T' < nT$$

Thus concurrency has enabled a reduction in the time to execute the actions of the repetitive clause.

At present, this seems to be the only procedure for optimising the repetitive clauses within a sequential program. It is a laborious process, and is not guaranteed to produce a significant increase in overall performance. This unsatisfactory situation suggests that the writing of the initial program should have been undertaken with a parallel implementation in mind. Programmers should learn ways to think parallel, before they think sequential.

## 2.2 Evolution of Parallelism by Functional Division

One can attempt the search for parallelism at the requirements definition stage of a product, before any implementation by programming is begun. Inspection of instrument descriptions such as that shown in Figure 1 suggests that it is possible to visualise each of the activities being implemented by program in a separate processing element. Depending upon the required performance, the elements may exist as separate physical entities, operating concurrently, or they may exist within one physical entity, being active one at a time. When the concurrent implementation is required, the solution involves the physical interconnection of the separate elements.

## 3. Design of a Circuit of Processing Elements

All circuits or networks may be considered in the abstract, as node components interconnected by arc components. In electronic circuits, the node components are physical electronic devices, whilst the arc is usually a piece of wire or track on the surface

of a silicon chip or printed circuit board. In logic circuits, the node components are logic gates or flip-flops, and the arc is similar to that in electronic circuits. In circuits of processing elements, the node component is a processing element comprising processor and memory, real or virtual, and the arc component is a data path which may be as real as a few pieces of wire or the complex Post Office data network, or as abstract as a pointer to memory.

The activities within a node component are shown in Figure 8. The input data from the arcs are received and acknowledged by the activity 1, and assembled as input data for the activity 2, which carries out the necessary processing of these data to produce result data to be passed to activity 3, which sends the data over the output arcs and confirms that the transmission has been acknowledged. This total set of activities may be implemented in a processing element, as shown in Figure 9 (a). The element comprises the processor and memory for obeying the programs of the three activities. The input data arrives within a read only image memory, whilst the output data flows from registers in the same image memory (memory mapped input/output) [11]. The structure of the program within the element is shown in Figure 9 (b). The single locus of control enters the Receive and Acknowledge action, which monitors the input ports and accepts input data, which it assembles as a message for the next action. Each transaction through the input port must follow an agreed protocol. Once the message has been assembled, the locus of control passes to the Process action, which carries out the necessary processing of the data and prepares a results message for transmission to the next action. On completion of the process, the locus of control passes to the final action, which sends the messages over the output arcs. Each transaction through the output port must follow a protocol to enable confirmation of correct transmission. When all arcs have been serviced, the locus of control is passed back to the first action, to repeat the cycle.

This sequence is similar to that which used to occur within a large mainframe computer before time-sharing of input/output and processing was introduced. The main reason for introducing time sharing, by interrupt or polling, was to maximise the use of an expensive capital resource. The electronic programmable devices are low in cost, and it is less important to time-share their use. The inner process activity is independent of the interconnection structure of which it is a component. The Receive-Acknowledge and Send-Confirm actions are dependant upon the interconnection structure. The process component corresponds to a logic circuit component, whilst the other two processes correspond to the back wiring or printed circuit board layout of a logic circuit. Thus a standard process program may be used as a standard component within many instruments. The difference between instruments is reflected in the choice of such components and in the programming of the outer activities. Performance is governed by the time taken to execute the inner process and the time to execute the overheads of the outer processes. Just as in a logic circuit, the performance depends upon the delay of the logic components and the delay due to the wiring between components. During the design

| TRANSFER STRATEGY: | | | | |
|---|---|---|---|---|
| DIRECT | | | | |
| | PATH: DEDICATED | | | |
| | | TOPOLOGY: | LOOP | (DDL) |
| | | : | WIRED COMPLETE | (DDWC) |
| | | : | WIRED PARTIAL | (DDWP) |
| | PATH: SHARED | | | |
| | | TOPOLOGY: | MEMORY | (DSM) |
| | | : | BUS | (DSB) |
| TRANSFER STRATEGY: INDIRECT | | | | |
| | ROUTING: CENTRALISED | | | |
| | | PATH: DEDICATED | | |
| | | | TOPOLOGY: STAR | (ICDS) |
| | | | : LOOP | (ICDL) |
| | | PATH: SHARED | | |
| | | | TOPOLOGY: BUS | (ICSB) |
| | ROUTING: DECENTRALISED | | | |
| | | PATH: DEDICATED | | |
| | | | TOPOLOGY: REGULAR | (IDDR) |
| | | | : IRREGULAR | (IDDI) |
| | | PATH: SHARED | | |
| | | | TOPOLOGY: BUS | (IDSB) |

Table 1. A Taxonomy (after Anderson & Jensen, 1975)

stage, the performance of different arrangements and numbers of processing elements must be assessed to arrive at an optimum solution.

## 4. Implementation of a Circuit of Processing Elements

The design process produces an abstract circuit showing the desired interconnections between processing elements. The next step is to man this circuit into a physical interconnection structure. Many factors have to be considered before deciding upon an interconnection structure. These factors, and many structures, are covered in the paper by Anderson & Jensen [12,13]. A variation of their taxonomy of interconnection structures, which is applicable to the present discussion, is given in Table 1. It shows eleven named varieties of interconnection structure.

### 4.1 Summary of Selected Structures

Certain of the structures listed in Table 1 have been selected for a brief description, since they warrant particular attention. They are shown in Figure 10.

#### 4.1.1 Direct Dedicated Wire Complete - DDWC (Figure 10(a))

In this, each element is connected to all others. Provided that the number of elements required is less than, or equal to, the number required, the implementation is straightforward. The addition of one extra element affects all of the others.

#### 4.1.2 Direct Dedicated Wired Partial - DDWP (Figure 10(b))

The interconnections are custom-built to suit the particular requirement. Random wiring of this type was used in the patching of the elements in an analogue computer. The wires to interconnect the digital processing elements are expensive to install and a potential source of unreliability.

#### 4.1.3 Direct Shared Memory - DSM (Figure 10(c))

The registers in the image memory of each processing element are contained within the shared memory. The Send and Confirm action in the source element sets the output data in a register, whilst the Receive and Acknowledge action in the destination element reads the register. The low cost of memory suggests that this is an appropriate structure for use in closely-coupled systems. Furthermore, the shared memory may be used to hold common data, or to extend the local memory by adding to that provided within a processing element.

#### 4.1.4 Direct Shared Bus - DSB (Figure 10(d))

Information transfer takes place by a processor in the source element writing, via the bus, into the memory of the destination element. Thus, the memory access mechanism within the element must allow two routes of entry; one from the local processor, and one from the bus. This interconnection structure is the one most favoured by the semiconductor component manufacturers since it offers a high degree of flexibility.

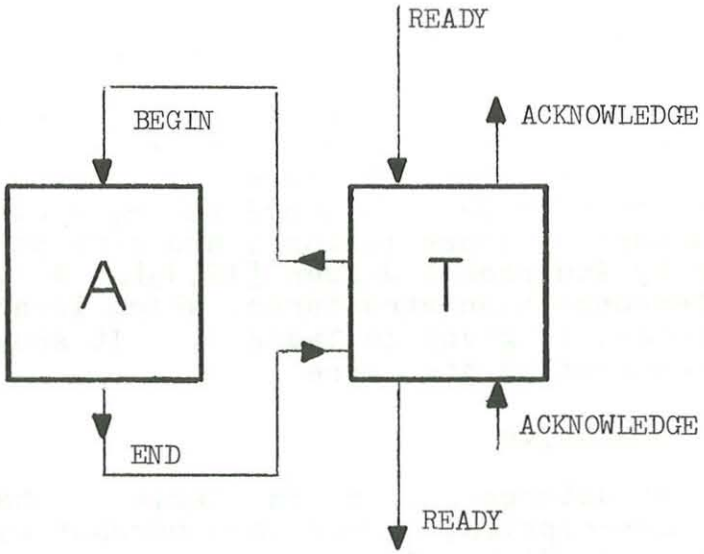



Figure 5 : T-Module and Controlled Action

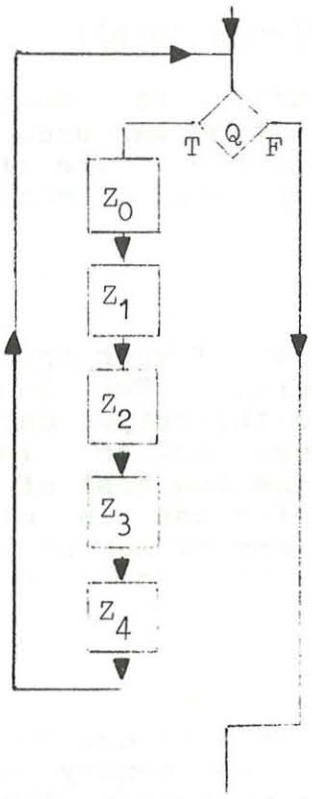



Figure 6 : A Repetitive Clause

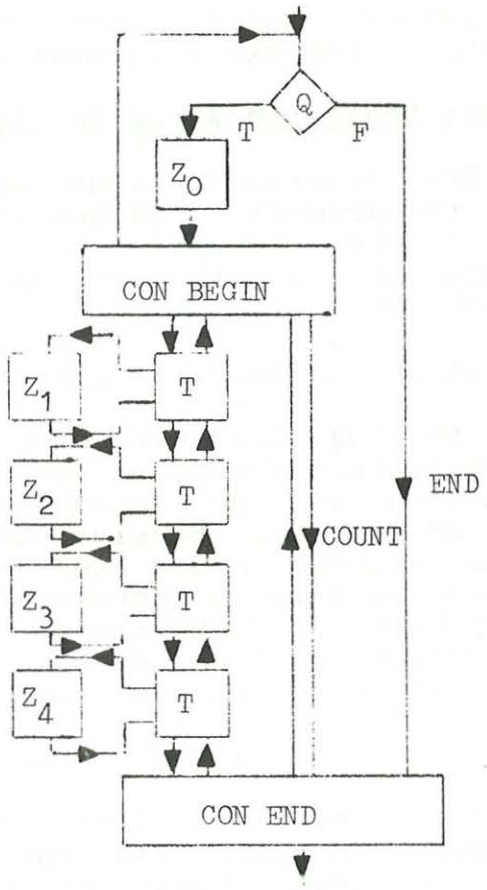



Figure 7 : A Pipeline Structure

4.1.5 The Remainder

All members of the taxonomy are worthy of some consideration [12]. In particular, the I D D R structure may come to the fore as the on-chip structure when very large-scale integration is in production.

## 5. A Research Vehicle

The engineering implications of programmable electronic devices has been a subject for investigation in the Department of Electrical and Electronic Engineering at the University College of Swansea, since 1972. More recently, in collaboration with the Department of Computer Science, and with support from the Science Research Council, the investigation has turned to a study of the design and implementation of equipment, using many processing elements. As part of the investigation, a vehicle is under construction to undertake the complete production of equipment to evaluate the design procedures and to establish design and development support facilities. The research vehicle, *CYBA-M [14] is based upon the DSM interconnection structure, as shown in Figure 11.

There are sixteen processing elements, each based upon the INTEL 8080-A microprocessor, plus 16 k bytes RAM of local memory. There are two shared memories. The Arc Image Memory, of 16 k bytes RAM provides the input/output buffer registers for inter-element communication. The Peripheral Image Memory is a 16 k byte area of memory mapped input/output registers for the communication of input and output data at the periphery of the equipment. Special semaphore facilities have also been included to permit interelement communication in a time-shared situation. The diagram of Figure 11 is presented to the designer at the time when the design is being mapped into the interconnection structure. However, during the program development stage, when the actual implementation is being tested, the engineer has extra monitoring facilities available as shown in Figure 12. The console processing element, CYBA-80, has access into each processing element to enable loading of the program and monitoring of the state of CYBA-80 and all the registers within the processors can be accessed to provide full information to the engineer. The monitoring facilities will also be available for maintenance of CYBA-M, and have already proved invaluable during the commissioning phase of the project. The total research vehicle includes software aids to design and development, and will be used in the implementation of several applications studies in the provision of communications equipment, sophisticated instruments, and in process control. The vehicle also provides an opportunity to investigate the interconnection structures listed in the Anderson & Jensen taxonomy.

*CYBA-M is the abbreviation for Coley Y Brifysgol Abertarwe-MYNYCH.

## 6. Conclusion

The use of many processing elements to provide the circuit of a piece of equipment is a subject for further research. This approach will draw heavily on the accumulated experience of the computer industry in its endeavours to provide multi-programming and multi-computer solutions to the problems posed by its customers. It shares many concepts with the area of distributed processing in which the processing elements are scattered over a wide physical area.

Before the advent of low-cost programmable electronic devices, the designer, or programmer, could only design for a single processing element; just as the early composers wrote for a solo instrument. Now that it is possible to contemplate a plurality of elements operating in concert on a common task, the designer can think of composing symphonies.

## References

[1] ASPINALL, D. Microprocessors: New Components for the Electronics Engineer. Electronics and Power, July 1976, 22.6, p.437.

[2] ROSS, D.T. Structured Analysis (SA) : A Language for Communicating Ideas. IEEE Trans. on Software Engineering, SE-3 No. 1, Jan. 1977. pp.16-34.

[3] THORNTON, J.E. Design of a Computer : The Control Data 6600. Scott Fonesman & Co., Glenview, 1970.

[4] PETRI, C.A. Kommunication mit Automaten, Translated into English in Project MAC-M-212 Report, 1962.

[5] HOLT, A.W. Information System Theory : Project Report. Applied Data Research Inc., 1969.

[6] NOE, J.D. and NUTT G.J. Macro E-Nets for Representation of Parallel Systems. IEEE Trans. on Computers, C-22, 1973, pp.718-727.

[7] DENNIS, J.B. Modular Asynchronous Control Structures for a High Performance Processor. Record of Project MAC Conference on Concurrent Systems and Parallel Computation, 1970, pp. 55-80.

[8] CLARE, C.R. Designing Logic Systems using State Machines. McGraw-Hill, 1973.

[9] DIJKSTRA, E.W. Notes on Structured Programming, in Dahl, O.J., Dijkstra, E.W. and Hoare, C.A.R. Structured Programming Academic Press, 1972, pp.1-82.

[10] ASPINALL, D., DAGLESS, E.L. and DOWSING, R.D. Design Methods for Digital Systems including Parallelism. Electronic Circuits and Systems, Jan. 1972, pp. 49-56.

[11] ASPINALL, D. and DAGLESS, E.L. (Editors) Introduction to Microprocessors. Pitman/Academic Press, 1977.

[12] ANDERSON, G.A. and JENSEN, E.D. Computer Interconnection: Taxonomy, Characteristics and Examples. ACM Computing Surveys, Vol.7, No.4, Dec., 1975.

[13] JENSEN, E.D. et al. A Review of Systematic Methods in Distributed Processor Interconnection. Proc.IEEE Conf. on Communications, Philadelphia, June, 1976.

[14] DAGLESS, E.L. CYBA-M : A Multimicroprocessor. Information Processing, 77. B.Gilchrist (Ed.). North Holland Publishing Co., Amsterdam, 1977, pp.843-848.

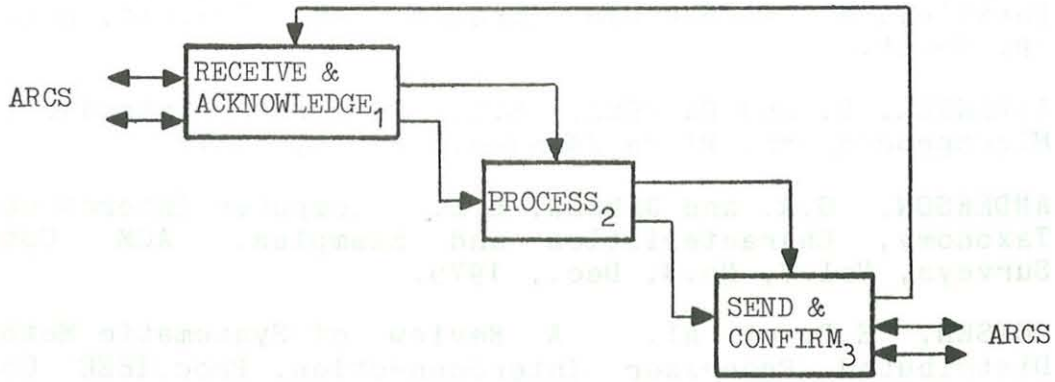
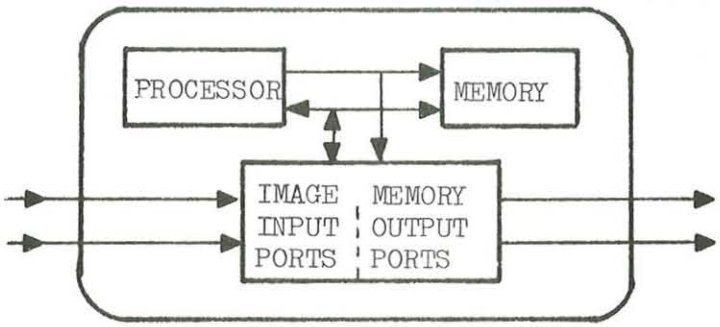
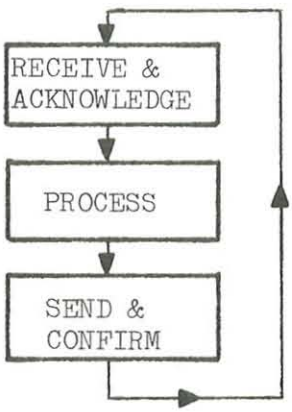



Figure 8 : Activities within Node

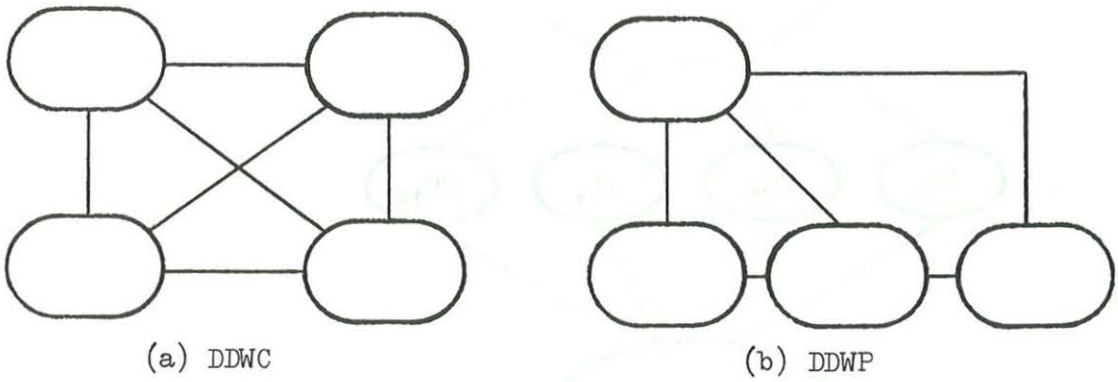



(a) Processing Element




(b) Programme within Element

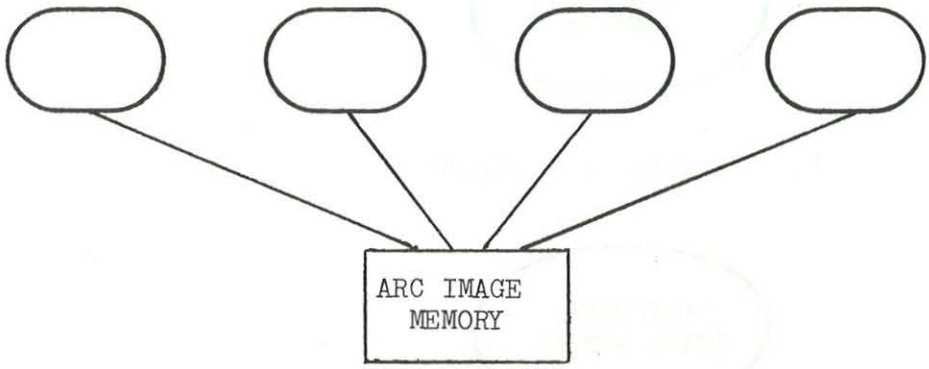
Figure 9 : Implementation of a Node



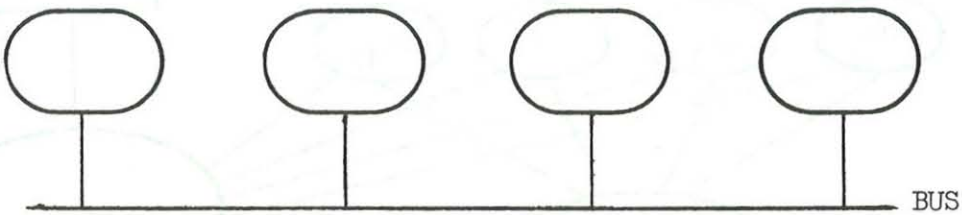
(a) DDWC



(b) DDWP




(c) DSM




(d) DSB

Figure 10 : Selected Interconnection Structures

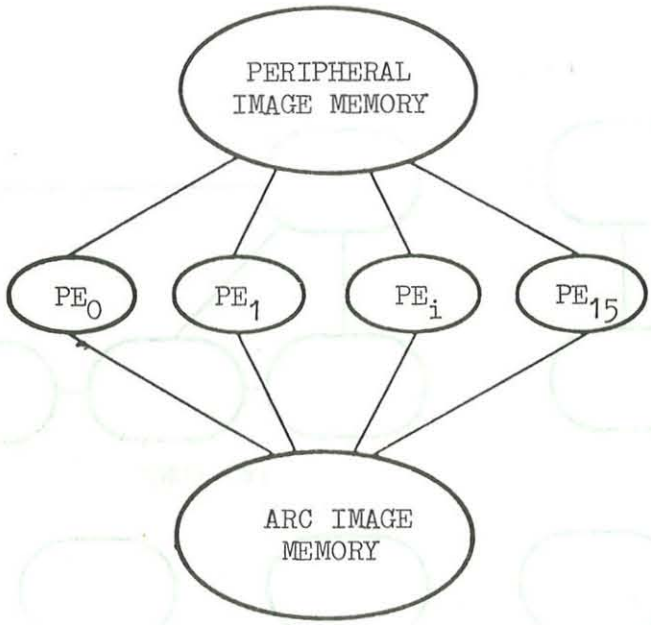



Figure 11 : CYBA-M

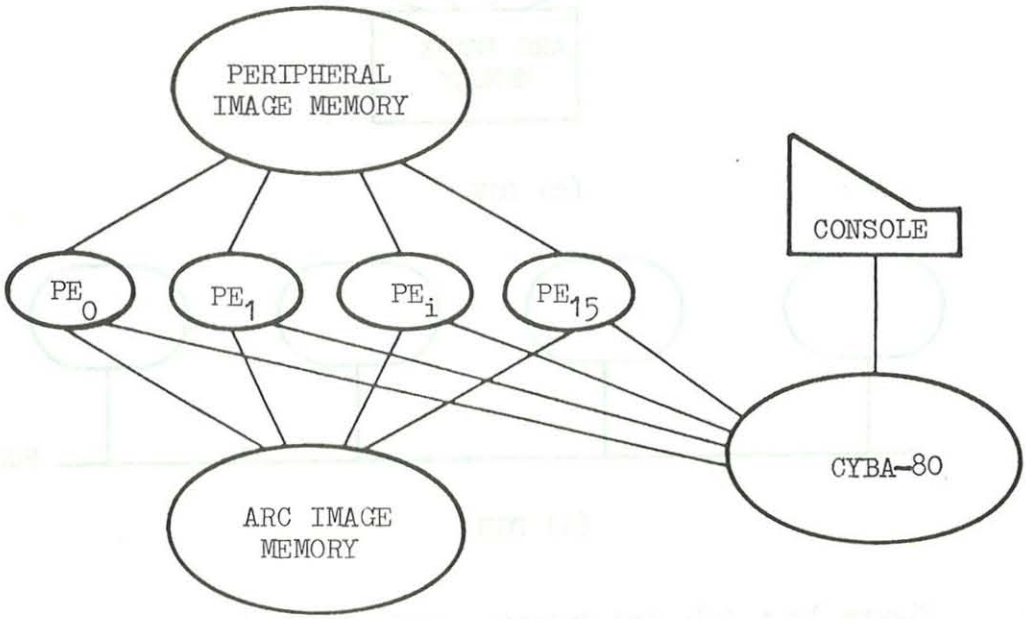



Figure 12 : Development System