Multiprocessor design for real-time embedded systems

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfillment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/7474

Publisher: © W.I. Al-Hasawi

Please cite the published version.
This item is held in Loughborough University’s Institutional Repository (https://dspace.lboro.ac.uk/) and was harvested from the British Library’s EThOS service (http://www.ethos.bl.uk/). It is made available under the following Creative Commons Licence conditions.

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
MULTIPROCESSOR DESIGN FOR REAL-TIME EMBEDDED SYSTEMS

BY

WALED IS A AL-HASAWI, B.Sc., M.Sc.

a doctoral thesis submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University of Technology

Feb. 1987

Supervisor:

J.E. Cooling B.Sc., C.Eng., MIEE.
Department of Electronic and Electrical Engineering

© W.I. Al-Hasawi, 1987
ACKNOWLEDGEMENT

I wish to express my deep sense of gratitude to my supervisor Mr J E Cooling, for his guidance, inspiration and encouragement during the preparation of this thesis.

I would also like to thank the staff, research students, and technicians for their assistance.

The author is indebted to his wife for her endurance and support throughout the project.
SYNOPSIS

Multiprocessor Design for Real-Time Embedded Systems

In the world of engineering real-time embedded systems are many and numerous, including applications such as aircraft flight control systems, robotics and process control. Real-time systems are identified as being embedded using two specific criteria. First there is a time constraint set by the task, response times usually being quite short. Failure to meet such requirements normally leads to a partial or total failure of the system. This means that the performance must be guaranteed under all loading conditions, i.e. the system must be deterministic in operation. Second, such implementations use computers as components, not as computers in the sense of mainframe installations. Most implementations have, up to the present time, used single processor designs. However, where high throughput, safety and security of operation and flexibility and expandability are required a multiprocessor solution is superior to traditional methods. This thesis presents a new type of multiprocessor architecture that has been devised specifically for real-time embedded applications. The structure is designed to achieve:

- High performance with deterministic response times
- High modularity and expandability
- Good fault tolerance performance
- Automatic reconfiguration

The system described here has the following main features:

- It is based on a loosely coupled multiprocessor structure, each processing unit consisting of a single board computer.
- It uses a single shared parallel bus for communication between these computers ('stations').
- The communications and computing tasks are decoupled, each station having a dedicated processor to handle inter-processor communications.
- Bus control is fully distributed, no master station being involved.
- Deterministic response is obtained by using a control and message passing technique called 'Token Passing Bus Access'.
- The computing processor is CPU independent.
- It includes a duplicated parallel bus together with a serial data bus for use in critical systems as a back-up communications facility.

The demonstrator system described here uses an Intel 8031 processor to handle communications activities whilst an Intel 188 performs the main computing task. Software coding is done in ASM51, ASM86 and Pascal.
TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1

1.1 OVERVIEW 1
1.2 THESIS ORGANISATION 1

CHAPTER 2 REAL-TIME SYSTEMS REQUIREMENTS 2

2.1 OVERVIEW 2
2.2 GENERAL REQUIREMENTS 2
  2.2.1 OVERVIEW 2
  2.2.2 CORRECTNESS 2
  2.2.3 RESPONSIVENESS 2
  2.2.4 RELIABILITY AND AVAILABILITY 2
  2.2.5 EXPANDABILITY 2
2.3 REAL-TIME DISTRIBUTED SYSTEMS REQUIREMENTS 2
  2.3.1 OVERVIEW 2
  2.3.2 BOUNDED MESSAGE TRANSFER 2
  2.3.3 HIGH THROUGHPUT 2

CHAPTER 3 MULTIPROCESSOR SYSTEMS 3

3.1 INTRODUCTION 3
3.2 CRITERIA FOR EVALUATION 3
  3.2.1 GENERAL 3
  3.2.2 SYSTEM THROUGHPUT 3
  3.2.3 RELIABILITY 3
  3.2.4 TURN-ROUND TIME 3
  3.2.5 MODULARITY 3
  3.2.6 PERFORMANCE/COST 3
  3.2.7 FEASIBILITY 3
3.3 CLASSIFICATION OF PROCESSING SYSTEMS 3
3.4 SINGLE INSTRUCTION MULTIPLE DATA (SIMD) ARCHITECTURE 3
3.5 MULTIPLE INSTRUCTION MULTIPLE DATA (MIMD) SYSTEMS 3
  3.5.1 GENERAL 3
  3.5.2 CLASSIFICATION OF MIMD SYSTEMS GENERAL 3
3.5.3 CLASSIFICATION BASED ON INTER COMPUTER COMMUNICATION

3.5.4 CLASSIFICATION BASED ON PROCESSOR/DEVICE INTERCONNECTION

3.5.5 DEGREE OF INTERACTION BETWEEN PROCESSORS

3.5.6 INTERCONNECTION TOPOLOGY FOR COMMUNICATION

3.6 SYSTEMS WITH DIRECT CONNECTIONS

3.6.1 RING STRUCTURE

3.6.2 COMPLETELY INTERCONNECTED ARCHITECTURE

3.6.3 CENTRAL MEMORY ARCHITECTURE

3.6.4 GLOBAL BUS ARCHITECTURES

3.7 SYSTEMS WITH INDIRECT CONNECTIONS

3.7.1 STAR STRUCTURE

3.7.2 RING ARCHITECTURE WITH SWITCH

3.7.3 BUS STRUCTURE WITH SWITCH

3.7.4 REGULAR NETWORKS

3.7.5 IRREGULAR NETWORKS

3.7.6 BUS WINDOW SYSTEMS

3.8 CONFLICT RESOLUTION

3.8.1 GENERAL

3.8.2 ARBITRATION

3.8.3 TEST-AND-SET CAPABILITY

3.8.4 INTERRUPTS

3.9 MAILBOXES

REFERENCES

CHAPTER 4 TOKEN PASSING BUS ACCESS METHOD

4.1 MAIN FEATURES OF THE TPBAM

4.2 BASIC CONCEPT

4.3 USE OF THE BUS

4.4 RING INITIALISATION AND MAINTENANCE

4.4.1 RING INITIALISATION

4.4.2 ADDITION OF A STATION

4.4.3 DELETION OF A STATION

4.5 FRAME FORMAT

4.6 TIMERS
CHAPTER 5 IMPLEMENTATION OF THE TPBAM IN A MULTIPROCESSOR SYSTEM

5.1 OVERVIEW
5.2 SYSTEM INTERFACING
5.3 THE NETWORK INTERFACE BLOCK (NIB)
  5.3.1 GENERAL
  5.3.2 SCRATCH PAD MEMORY
  5.3.3 ACCESS CONTROL BLOCK (ACB)
  5.3.4 COMMUNICATION CPU
  5.3.5 STATION SELECT AND ADDRESS RECOGNITION BLOCK
  5.3.6 TIMING AND CONTROL BLOCK
  5.3.7 SYSTEM BUS BUFFERS
5.4 FUNCTIONS OF THE MAIN PROCESSOR BLOCK

CHAPTER 6 HARDWARE DESIGN OF THE NETWORK INTERFACE BLOCK

6.1 GENERAL
6.2 SCRATCH PAD MEMORY
  6.2.1 OVERVIEW
  6.2.2 TMS9650 PORT A INTERFACE
  6.2.3 TMS9650 PORT B INTERFACE
6.3 ACCESS CONTROL BLOCK
6.4 COMMUNICATION CPU
  6.4.1 GENERAL
  6.4.2 CLOCK
  6.4.3 POWER-ON RESET
  6.4.4 ADDRESS/DATA BUFFERS
  6.4.5 CHIP SELECT LOGIC
  6.4.6 PROGRAM MEMORY
  6.4.7 INPUT/OUTPUT
6.5 STATION SELECT AND ADDRESS RECOGNITION
  6.5.1 FUNCTIONS
  6.5.2 SETTING OWN ADDRESS
  6.5.3 ADDRESS RECOGNITION
  6.5.4 SET DESTINATION ADDRESS
6.5.5 SYSTEM BUS ADDRESSING 6-14
6.6 TIMING AND CONTROL BLOCK 6-14
6.6.1 GENERAL 6-14
6.6.2 OPERATION OF THE TC 6-15
6.7 SYSTEM BUFFERS 6-16
6.7.1 DATA BUFFERS 6-16
6.7.2 CONTROL SIGNALS BUFFERS 6-16
6.8 NETWORK INTERFACE BLOCK MODES OF OPERATION 6-16
6.8.1 GENERAL 6-16
6.8.2 GET MESSAGE FROM MAIN PROCESSOR 6-17
6.8.3 TRANSMIT MESSAGE 6-18
6.8.4 RECEIVE MESSAGE 6-18
6.8.5 TRANSFER MESSAGE TO MAIN PROCESSOR 6-18

CHAPTER 7 THE MAIN PROCESSOR 7-1

7.1 GENERAL 7-1
7.2 MAIN PROCESSOR SYSTEM OVERVIEW 7-2
7.2.1 GENERAL 7-2
7.2.2 CPU SECTION 7-2
7.2.3 MEMORY 7-3
7.2.4 SERIAL COMMUNICATIONS 7-3
7.2.5 NIB INTERFACE 7-3
7.3 MAIN PROCESSOR BLOCK-HARDWARE DESIGN 7-4
7.3.1 THE CPU SECTION 7-4
7.3.2 MEMORY 7-9
7.3.3 SERIAL COMMUNICATIONS 7-10
7.3.4 NIB INTERFACE 7-10
7.3.5 ANCILLARY CIRCUITS 7-12

REFERENCES 7-13

CHAPTER 8 PROTOCOL SOFTWARE DESIGN 8-1

8.1 OVERVIEW 8-1
8.2 INITIALLISATION 8-2
8.2.1 GENERAL 8-2
8.2.2 FIRST STATION ROUTE (ROUTE1) 8-4
8.2.3 CLAIM TOKEN RECEIVED ROUTE (ROUTE2) 8-6
8.2.4 SOLICIT SUCCESSOR RECEIVED ROUTE
CHAPTER 9 SYSTEM PERFORMANCE EVALUATION

9.1 OVERVIEW
9.2 EFFECT OF BUS ACCESS METHOD ON DISTRIBUTED COMPUTERS
9.3 PERFORMANCE MEASURES
9.4 MATHEMATICAL MODEL
9.5 RESULTS
  9.5.1 ASSUMPTIONS
  9.5.2 EFFECT OF NUMBER OF STATIONS ON RESPONSE TIME
  9.5.3 EFFECT OF MESSAGE LENGTH ON RESPONSE TIME
  9.5.4 EFFECT OF MESSAGE LENGTH ON THROUGHPUT
9.6 ENHANCED SYSTEM

REFERENCES

CHAPTER 10 CONCLUSION

10.1 OVERVIEW
10.2 SOFTWARE DEVELOPMENT FOR DISTRIBUTED SYSTEMS
10.3 EVALUATING DIFFERENT ACCESS TECHNIQUES
10.4 COMMENTS AND CONCLUSIONS

REFERENCES
APPENDIX A CONTROL AND DATA FRAMES

APPENDIX B HARDWARE DESIGN OF THE NETWORK INTERFACE BLOCK

B.1 TMS 9650
B.2 PAL PROGRAMMING
B.3 TIMING SIGNALS GENERATION
Chapter 1

INTRODUCTION

1.1 OVERVIEW

The advent of the microprocessors had a tremendous impact on both our technology and our society, the device being used in an almost endless number of applications. The electronics/semiconductor industry has succeeded in increasing the speed of microprocessors and related components and at the same time reducing their physical size. Along with this has come a major reduction in costs. The economics of modern processor systems has inspired designers to develop new and varied multiprocessor systems in order to significantly improve computer performance. The ideal objective is to create a system containing \( n \) processors that is \( n \) times faster than a system containing only a single processor. A multiprocessor system possesses the capability to not only improve speed but also to increase reliability and provide for the tolerance of processor failure.

Real-time systems are one possible application area for multiprocessor systems. In these systems the time critical task can be partitioned into independent subtasks which are executed simultaneously on different processors. Therefore a fast response time can be attained by exploiting a multiprocessor system
design.

This thesis presents a new type of multiprocessor system architecture that has been devised specifically for real-time embedded applications. The structure is designed to achieve:

* High performance with deterministic bus response times.
* High modularity and expandability.
* Good fault tolerance performance.
* Automatic reconfiguration.

The system described here has the following main features:

* It is based on loosely coupled multiprocessor structure, each processing unit consisting of a single board computer.

* It uses a single shared parallel bus for communication between these computers (stations).

* The communications and computing tasks are decoupled, each station having a dedicated processor to handle inter-processor communications.

* Bus control is fully distributed, no master station being involved. Deterministic bus response is obtained by using a control and message passing technique called 'Token Passing Bus Access'.

* The computing power is CPU independent.

* It includes a serial data bus for use in critical systems as a back-up communication's facility.

The demonstrator system described here uses an Intel 8031 processor to handle
communication's activities whilst an Intel 80188 performs the main computing task.

1.2 THESIS ORGANISATION

The requirements of real-time systems is presented in Chapter 2 with some emphasis on distributed systems.

Chapter 3 is a general review of the current state of multiprocessor systems. Different classifications are presented together with implementation examples and basic evaluation of each structure.

Chapter 4 introduces the processor access technique used for message transmission between stations. This is defined as the 'Token Passing Bus Access' method.

Chapter 5 describes the implementation of the multiprocessor system developed in this research programme at building block level. The function of each block and its role in the system is demonstrated. It introduces the idea of using two separate processors, one for communication task handling, the other for execution of application programs.

Chapter 6 is devoted for the hardware design of the communication processor sub-system (Network Interface Block). The hardware realisation of each block described in chapter 5 is illustrated here.

Chapter 7 set out the hardware realisation of the processor which is responsible
for executing application programs (the Main Processor).

Chapter 8 describes the software design of the overall communication process, illustrated by the use of flowchart diagrams.

Chapter 9 presents mathematical models used to evaluate the performance of the system and its suitability for real-time applications. The effect of varying individual parameters on system performance is also studied.

Chapter 10 reviews and assesses the experience gained from working in this project. It also points the way forward for further areas of research related to the development of real-time multiprocessor systems.
Chapter 2

REAL-TIME SYSTEMS REQUIREMENTS

2.1 OVERVIEW

A real-time computer system may be defined as one which controls an environment by receiving data, processing them, and taking action or returning results sufficiently quickly to affect the functioning of the environment at that time [1]. In designing real-time systems there are two fundamental concerns [2]:

(a) To produce correct results.
(b) To produce the results within an allotted time.

The above conditions must be met by the real-time system in all circumstances. If the system fails to meet these conditions it must then behave safely to avoid any damages (degrade gracefully).

Many of the real-time requirements are in some way connected to the above two fundamental concerns.
2.2 GENERAL REQUIREMENTS

2.2.1 OVERVIEW

The general requirements of a real-time system are;

(a) Correctness.
(b) Responsiveness.
(c) Reliability and availability.
(d) Expandability.

2.2.2 CORRECTNESS

A real-time system must produce accurate output states only [3]. The output of the system must be correct, in that suitable decisions are made when different situations arise (even if the decision is to do nothing at all).

2.2.3 RESPONSIVENESS

A real-time system must be responsive to changes in the environment it is operating on. Unless the system reacts sufficiently rapidly, it cannot be considered to be operating in real-time. This property is usually quantified as the system's response time [4]. The response time of a system is defined as the time that the system will take to react to a change in, or stimulus from, its environment. It is useless (and dangerous) for a real-time system to deliver an output if some deadline has been missed. It must be able to either deliver correct outputs within strict time margins or remain exceptionally silent.
2.2.4 RELIABILITY AND AVAILABILITY

A real-time system must be reliable. It must be able to provide a service that can be closely defined in terms of guaranteed mean time between failure (MTBF) and mean time to repair (MTTR). Reliability and availability can be enhanced by the use of fault prevention and fault tolerance techniques. The use of fault tolerance techniques is based on the generally accepted belief that a complex system, no matter how carefully designed and validated, is likely to contain residual design faults [4]. In real-time systems, erroneous computation and erroneous timing are the enemy.

2.2.5 EXPANDABILITY

This is a characteristic which has always been desirable in the construction of real-time systems. It is unquestionably a great advantage for a system to be complete for the purpose of one particular application, and at the same time to form the basis for a system of greater proportions.

2.3 REAL-TIME DISTRIBUTED SYSTEMS REQUIREMENTS

2.3.1 OVERVIEW

The concept of a distributed/multiprocessor system refers to the use of multiple, quasi-independent processing modules, whose action are coordinated to accomplish a large task or to implement a large system [5]. Microprocessors now
offer high computational power, high reliability and low power consumption at a low cost. They are finding widespread use in instrumentation and control systems where the microprocessor provides a centralised computing resource. Increasingly, microprocessors are being used in the construction of decentralised and distributed systems [6], in which a number of processors are physically distributed about the application plant and interact, or exchange information, with each other by passing messages over interprocessor communication channels. The individual processors in these systems form part of an overall system which must be co-ordinated to give a global response.

In addition to the requirements mentioned in the previous sections distributed real-time systems have their own requirements. The reason for this is the use of interprocessor communications to exchange information between processors. These additional requirements are;

(a) Bounded message transfer.
(b) High throughput.

2.3.2 BOUNDED MESSAGE TRANSFER

One of the most obvious constraints on a distributed real-time system is that message transfer times (and thus medium access) be “bounded” [7]. This means that the maximum time a processor has to wait before being able to transmit a message (gain access to the medium) can be predicted.

2.3.3 HIGH THROUGHPUT

System throughput is defined as the total number of bits (or bytes) received at
the destination per second [8]. Every distributed system has a maximum throughput value. If a system has high throughput then the amount of information exchanged between processors is also high.
REFERENCES FOR CHAPTER 2


8. Chien, J. 'Performance analysis of the 802.4 token bus media access control protocol' Workshop on Analytic and Simulation Modeling of IEEE
802.4 Token Bus Local Area networks held at Gaithersburg, Maryland, USA, April 1985 (Final report), pp113-152.
Chapter 3

MULTIPROCESSOR SYSTEMS

3.1 INTRODUCTION

Why move to a multiprocessor configuration from a single (uni) processor one and suffer the cost and technical complexity involved in such a design? The short answer is 'necessity', the driving forces mainly being:

* Throughput
* safety/security
* Flexibility and expandibility

Earliest applications of multiprocessing were those where speed of operation was paramount, e.g. the use of array processors for data handling in Radar, Sonar and signal processing applications [1]. However with the emergence of 32 bit general purpose processors (e.g. Motorola 68030, National NCS32000, Intel 80386) and the availability of specialist signal processing chips (e.g. Texas TMS32020) much of this type of work can be handled by uniprocessor designs.

The second point, that of security and safety of operation, is quite different in its requirements. Safety critical multiprocessing applications include aircraft fly-by-wire systems, automatic train control and plant monitoring functions.
Generally each processor performs an identical task, all outputs being cross-checked by a 'majority voter'. Basically the voter is required to detect malfunctions within the system and force it into a safe condition. In some applications it is essential that the system 'degrades gracefully'. For instance a quadruplex fly-by-wire controller should, in the event of a single channel failure, fall back to triplex operation.

Finally, flexibility and expandibility are very desirable in certain applications. At one end of the spectrum there are defence projects which have, by industrial standards, exceptionally long lives. For instance the mid-life refit of a warship usually takes place after 10 years. To this should be added the development lead-time of the project, typically 5 to 10 years. System requirements may change substantially within this period, far exceeding the capability of the original designs. Multiprocessor solutions, at the very least, offer the ability to significantly improve the effectiveness of computer based systems. At the other end of the spectrum many commercial manufacturers provide standard electronic hardware 'building blocks'; multiprocessor designs within such systems provide great flexibility in meeting many and varied customer requirements. It is no accident that all modern standard microprocessor bus structures [2] support multiprocessing. This is essentially a response to technical and commercial pressures.

At present, there are no standard methods for connecting several processors together. There is, however, a wide range of possible architectures, each having its own advantages which make it suited for certain applications.
3.2 CRITERIA FOR EVALUATION

3.2.1 GENERAL

One very important factor in developing multiprocessor systems is the evaluation of one structure in comparison with others. The major criteria are:

(a) System throughput.
(b) Reliability.
(c) Turn-round time.
(d) Modularity.
(e) Performance/cost.
(f) Feasibility.

3.2.2 SYSTEM THROUGHPUT

Throughput is defined as the reciprocal of the time employed to perform a benchmark program and is measured in operations per time unit. In this sense it is important to consider that, when a program originally designed for single processor system is subdivided into tasks for a multiprocessor system, it is necessary to provide a certain "overhead" of time. This time is necessary to allow communications to take place between processors; however it must be minimized to attain high throughputs.

3.2.3 RELIABILITY

In general, the breakdown of one of the components of a system must allow operations to be continued, albeit at a reduced level of throughput. This is
true when the existing redundancy permits redistribution of tasks. However, if this is not possible, the system must do without the task assigned to the processor that has broken down. In the first case, evaluation of reliability consists in determining the reduced functionality of the system, in the second it is the average repair time.

3.2.4 TURN-ROUND TIME

When a processor requires data from another processor in order to perform its task it may well have to wait for a certain period of time before being able to complete the computation. The length of waiting time is defined as the turn-round time.

3.2.5 MODULARITY

A system is said to be modular if it consists of a small set of basic modules which may be connected in almost any number [3].

This is a characteristic which has always been desirable in the construction of processing systems because of the inherent expansion facility. In addition, a modular structure gives the system the flexibility required in many applications. It is unquestionably a great advantage for a processing system to be complete for the purpose of one particular application, and at the same time to form the basis for a system of greater proportions.

3.2.6 PERFORMANCE/COST

In the design of multiprocessor systems it is very important to consider the
cost of implementing such an architecture with respect to the performance gained from this implementation. This is defined as the base performance/cost ratio. The incremental performance/cost ratio is defined as the ratio of the performance gained versus hardware costs as more processors are added to the system.

3.2.7 FEASIBILITY

Feasibility is defined as the possibility of development which includes the availability of all the hardware and software needed to implement the system, the affordability of the costs, and the ease of development.

Based on the above mentioned points, one can evaluate different architectures of multiprocessor systems keeping in mind that they vary in importance depending on different applications and situations.

3.3 CLASSIFICATION OF PROCESSING SYSTEMS

There is no standard classification of processing systems. In attempting a first description the concept of streams is used, i.e. a sequence of codes which can be executed or processed by a processor according to whether they represent instructions or data [4]. This way of processing leads to four types of architecture.

(a) SISD (Single-Instruction/Single-Data-stream): This is simply the conventional single-processor system.
(b) SIMD (Single-Instruction/Multiple-Data-stream): The SIMD architecture has also been called a "parallel processor" because the same instruction is carried out simultaneously by a number of processors on a number of data streams.

(c) MIMD (Multiple-instruction/Multiple-Data-stream): This includes computer networks and multiprocessor systems, a structure having the widest field of architecture.

(d) MISD (Multiple-Instruction/Single-Data-stream): These are structures which are rarely used at present.

Only the SIMD and MIMD structures are described here, examples being given where applicable.

3.4 SINGLE INSTRUCTION MULTIPLE DATA (SIMD) ARCHITECTURES

The SIMD type of computer usually consists of a single control unit and a number of processing elements (PROC) connected through an interconnection network (Fig. 3.1). PROCs may be general or special processors, each having the ability to execute specific tasks at high speeds. For instance, PROC might be a processor designed specially to perform mathematical functions for the solution of differential equations. The control processor handles overall co-ordination and control of computation process. It places (or is responsible for placing) data and instructions in the memory store of each PROC and then commanding the PROCs to execute their instructions based on this information.

SIMD computers are generally found in applications where a single task is to be
executed using different data sets, and where speed is the prime requirement. These include weather prediction, air-traffic control, processing of radar signals, high speed vectorial calculations and high speed signal processing in general [5].

The major advantages of SIMD systems are [6]:

(a) High throughputs on problems which have an inherent parallel structure.

(b) High performance/cost ratio, resulting from the modularity of processing elements and the use of single common controller.

The main problems of SIMD systems are [6]:

(a) The communications between the processors is fixed.

(b) The efficient use of hardware resources is significantly affected by the fact that, at any given moment, all of the processing units are executing the same instruction. When branches in the source program occur, the control unit can follow only one of the instructions for it, and so the other processors remain inactive.

(e) There are difficulties involved in constructing operating systems for them.

one example of a practical SIMD microprocessor implementation is ANMA (A Novel MultiProcessor Array [5]).
3.5 MULTIPLE INSTRUCTION MULTIPLE DATA (MIMD) SYSTEMS

3.5.1 GENERAL

In MIMD structures the parallel (concurrent) processing is achieved by executing independent tasks during the same time slot. The various processors use different groups of data; where appropriate the data itself can be exchanged between processors in order to carry out the system processing task. In general, the architecture of the MIMD computer is as shown in Fig. 3.2. Two very important types of MIMD processing systems, computer networks and multiprocessor systems, fall into this category.

The distinction between the two is based on both hardware and software. Computer networks are formed of computers (Fig. 3.3), each complete in itself, i.e. constituted of CPU, primary memory storage, mass memory storage, peripheral I/O devices and network interface channel. These computers communicate with each other through the network interface channel, often serially. This makes it possible to implement geographically distributed systems. Each computer works under the control of its own operating system, the network supervisor being responsible only for handling information exchange between the various computers. The degree of interaction between the processors is very low, as each processor is connected to the others by messages sent through I/O channels [4,13,18]. In these systems transmission speeds are generally low (when compared with a parallel bus) and the time taken for a processor to access data held by another one is long.
However, when several CPUs interact at the central storage level, we can speak of actual, true multiprocessor systems (Fig. 3.4). The degree of interaction between processors can be very high, to the point of having an interrupt, via hardware, of one processor by another. The operating system of a multiprocessor system is single from the logical point of view, even though its routines can be implemented on different processors [7].

The classification definitions now given are not absolute as the border-line between the two organisation is certainly not clearly established [4].

3.5.2 CLASSIFICATION OF MIMD SYSTEMS - GENERAL

There is no standard classification of the MIMD system; instead sub-groups can be established defined by the attributes of the multiprocessor implementation. The four most important attributes are:

(a) Inter-computer communication.

(b) Processor/device interconnection.

(c) Degree of interaction between processors.

(d) Interconnection topology for communication.

3.5.3 CLASSIFICATION BASED ON INTER-COMPUTER COMMUNICATION

The methods used to communicate between computers form one basis for the classification of MIMD systems. Assume we have two computers, each consisting of a microprocessor, memory and I/O devices (Fig. 3.5). The following methods
can be used to enable the two to communicate with each other;

* Shared bus communication (Fig. 3.6).

* Multiport memory communication (Fig. 3.7).

* Input-output linked communication (Fig. 3.8).

(a) Shared bus communication: In this type of structure all the processors are connected to the same system bus. Inter-computer communication is achieved through the use of common (shared) memory techniques. One example of a multiprocessor system that uses a shared bus is the TOMP80 [8].

(b) Multiport memory communication: In this configuration inter-computer communication is achieved via the use of a multiport memory. Here all computers can access one specific memory block but each one use a different port. As a result there is no direct electrical connection between the computers. A good example of this type can be found in RSM (A multiprocessor with replicated shared memory [9]).

(c) Input-output linked communication: In this arrangement, typified by computer networks [10], communication is made via I/O links.

Note that when using (a) or (b) each computer can in theory access and execute common program code. This isn't possible with the I/O linked communication.

3.5.4 CLASSIFICATION BASED ON PROCESSOR/DEVICE INTERCONNECTION

This is based on the methods used to interconnect processors with memory and I/O devices. In general the most widely used schemes are;
* Shared bus (Fig. 3.6).

* Multibus multiport (Fig. 3.9).

* Crossbar switch (Fig. 3.10).

(a) Shared bus scheme: This has already been covered in section 3.5.3.

(b) Multibus multiport scheme: In this scheme every processor has a dedicated connection to all the memory and I/O devices (Fig. 3.9). Hence, each memory and I/O device must have a number of ports equal to the number of processors used in the system. Since these devices are shared then a technique must be used to control access to them. An implementation of this type of structure can be found in [11].

(c) Crossbar switch scheme: In this scheme a crossbar switch provides the interconnection paths between processors, memory and I/O devices, very much as in the telephone switch. Thus a number of processors can simultaneously access memory and I/O devices as long as they are not accessing the same device (no access conflict). With the crossbar switch access conflicts are resolved by the crossbar matrix control logic. Note that memory and I/O devices only need one access port. An implementation of the crossbar switch can be found in [12].

3.5.5 DEGREE OF INTERACTION BETWEEN PROCESSORS

The degree of interaction between processors forms yet another method by which MIMD systems may be classified. If the interaction is high (to the extent of sharing code) then the system is said to be tightly coupled (Fig. 3.11). If,
however, each processor has its local resources and communicate to other processors by passing messages then the system is said to be loosely coupled (Fig. 3.12).

(a) Tightly coupled systems: When a number of processors share a common memory block the result is a tightly coupled multiprocessor system. This is one of the most commonly used configurations, allowing systems of limited size (generally up to 16 processors) to be built [13]. The shared memory provides a very fast data transfer medium as well as making it possible to share common code. A tightly coupled system has the following major characteristics;

(i) Processors share the same physical enclosure.

(ii) The system has a shared memory.

(iii) Processors may share program code.

(iv) The message transfer rate between processors is very high (depends on the speed of the processor but it could reach up to 10 Mbyte/s e.g. VME, Multibus [2]).

(b) Loosely coupled systems: Loosely coupled systems are constructed when a number of processors with their local memories are connected via I/O devices. Thus message transfer is slower than that of the tightly coupled systems. However, the interconnection medium is more flexible in the sense of length and type of cable used to interconnect the processors. Also the interconnection medium may be connected as a parallel or serial bus structure.
3.5.6 INTERCONNECTION TOPOLOGY FOR COMMUNICATION

This classification is based on the hardware structures used to support the transfer of messages from one processor to another [4]. This includes the communication paths and the elements controlling the data transfers (switches).

A communication path is a physical means by which information is transferred from one processor to another.

The control elements act on messages, modifying the destination address or sending the message through one path rather than another (routing).

The options open to the designer are shown in Fig. 3.13. The first level consideration is to establish the transfer strategy, i.e. whether to allow processors to communicate directly with each other or to use a switched connection. In the latter case a further consideration is in the choice of a central or a distributed switch. Now decisions must be made concerning the communication paths, whether they are to be shared or dedicated. The final (bottom) level in Fig. 3.13 refers to specific implementation methods.

(a) Systems with direct connections:

* Ring structure.

* Completely interconnected architecture.

* Central memory architecture.

* Global bus architecture.
(b) Systems with indirect connections:

* Star structure.

* Ring architecture with switch.

* Bus structure with switch.

* Regular networks.

* Irregular networks.

* Bus window systems.

3.6 SYSTEMS WITH DIRECT CONNECTIONS

3.6.1 RING STRUCTURE

The ring structure is shown in Fig. 3.14, operating as follows; data is passed from processor to processor until the specified destination contained in the message is reached. Thus each processor inspects the message received and checks to see if the destination matches its own address. If so, the message is removed; if not the message is passed on to the next processor along the ring.

Features:

* Offers excellent modularity. Insertion of a new processor is a simple task but it will introduce an extra delay into the message transmission operation.

* Fault tolerance is low because a breakdown in any one communication path can
interrupt the flow of information in the ring.

* Very high transmission rate is possible (e.g. 10 to 100 Mbit/sec).

* Mixed transmission media can be used e.g. cable + fiber optics.

A variety of methods are used with the ring structure to control message handling operation between processors. These were originally designed for Local Area Networks (LAN) systems but may be adopted for multiprocessor systems. Some examples are; token passing technique, empty slot technique, and register insertion technique [14].

The best known LAN example of this type is that designed and installed at the University of Cambridge Computer Laboratory [15]. One implementation within a multiprocessor system is described in [16].

3.6.2 COMPLETELY INTERCONNECTED ARCHITECTURE

Here, (Fig. 3.15), each processor is connected directly with all others in the system by a dedicated connection. Therefore, if one processor needs to send a message to another processor it specifies the location of the destination processor and selects the appropriate path.

Features:

* The system is hardly modular, since when a new processor is added to the system it must somehow be connected to all other processors.

* Fault tolerance is good, since the failure of one processor will not cause a breakdown of the system.
* Communication between processors is simple because communication paths are dedicated, not shared.

One example of this type of architecture is the IBM Attached Support Processor System in which a maximum of four 360 or 370 systems may be connected together through I/O channels [4].

3.6.3 CENTRAL MEMORY ARCHITECTURE

This is the most widely used mode of connecting processors together [9] is that to which certain authors give exclusively the name "multiprocessor". Here the processors communicate through one or more storage units accessible to several processors (Fig. 3.16). The storage unit is used as a means of connection rather than merely for retaining data.

Features:

* Modularity of the system is very good.

* The capacity for exchanging messages can be increased simply by increasing the dimensions of the storage unit.

* Communication between processors is simple.

* The effects of a failure in the processors are felt in a limited manner by the system but a failure in the shared memory stops the system completely.

* The performance level increases more slowly than the number of processors added.
Example of this structure are the Multi-Maren multiprocessor [17], and the MCS multiprocessor system [18].

3.6.4 GLOBAL BUS ARCHITECTURES

The architecture examined here, illustrated in Fig. 3.17, includes a number of processors connected to the same global bus, through which they communicate with each other directly. The bus may be serial or parallel but the system must include a bus access control mechanism.

Features:

* Modularity is high, but, as the number of processors increases, the incremental performance gain attained by the addition of a single processor falls off.

* Fault tolerance and facility of reconfiguration are good in the event of damage to the processors, but are very low as regards damage to the bus.

A number of methods are used to control message transmission between processors for this type of structure. Many were originally designed for Local Area Networks (LAN) systems but have been adopted for multiprocessor systems. Some examples are; token passing bus, pure ALOHA, slotted ALOHA, carrier-sense Multiple access (CSMA), and carrier-sense multiple access with collision detection (CSMA/CD) [14].

The Honeywell Experimental Distributed Processor (HXDP) is an example of the global bus structure [19], examples also being found in [8] and [20].
3.7 SYSTEMS WITH INDIRECT CONNECTIONS

3.7.1 STAR STRUCTURE

In the star structure, shown in Fig. 3.18, the processors are connected to a central switch via dedicated paths. This switch contains a microcomputer where received messages are processed then transmitted to their destinations.

Features:

* Modularity depends on the ability of the central switch to handle more processors.

* Fault tolerance is high as regards the processors but low as regards the central switch.

* The complexity of the communication logic is moderate.

The star configuration is quite common in computer networks; e.g. the IBM Network 440 system which used 360-series computers connected to a 369/91 central control. These have also been implemented using microprocessors, typified by that developed by Pimental and Loeffler to build a real-time engine simulator [21].

3.7.2 RING ARCHITECTURE WITH SWITCH

In the ring structure with switch shown in Fig. 3.19, the switch has the overall
control in deciding which processor can use the shared transmission media. This can be achieved by sending polls out to each processor in turn. This type of structure is best suited to handling low-speed devices such as terminals [14]. In these the switch is connected to a remote computer and is responsible for a number of terminals.

Features:

* Modularity is high because it is simple to add new processors.

* Fault tolerance is low.

3.7.3 BUS STRUCTURE WITH SWITCH

The bus architecture with a central switch is shown in Fig. 3.20. The processors must acquire the bus before sending out messages towards the switch, which then transmits the messages to their destination. Features:

* The modularity of the bus system is greater than that of the star structure.

* Fault tolerance is high as regards the processor but low as regards the switch and the system bus.

The SMS machine is an example of such type of architecture [22]. The Military Standard 1553A developed by the Department of Defense (USA) is one of the early implementations of such system structure [23].

3.7.4 REGULAR NETWORKS

The regular networks structure, an example of which is shown in Fig. 3.21,
consists of an array of processors connected by dedicated connections. Each processor has four communication paths, one for each neighbouring processor like tetra-valent bonding connection. Each row and column can be visualized as a ring of processors.

Features:

* The modularity of this type of system is low.

* Fault tolerance is good.

* Communications complexity is not very high.

* The possibility of reconfiguration is very low.

This type of architectures have applications in special areas such as image processing [4]. The invention of the transputer by INMOS made the realisation of such an architecture a straightforward task [24]. The reason for this is the provision of four fast (1.5 Mbytes/s) independent serial links within each transputer.

3.7.5 IRREGULAR NETWORKS

This differs from the previously described architecture in that the processors can be inserted at any point (Fig. 3.22). Each processor can be connected to any number of neighbouring ones through dedicated connections.

Features:

* Modularity is excellent as both processors and connections can be inserted at
any point.

* Flexibility is high as the structure follow the geographical distribution of processors.

* Communications control is complex.

The most common applications of irregular networks architecture are to be found in geographically distributed computer networks [13].

The u* project is an example of a microprocessor based implementation of the irregular network structure [25].

3.7.6 BUS WINDOW SYSTEMS

In this type of structure, the connections with the switch devices are shared by several processors (Fig. 3.23). Switching is carried out by more than one unit, and the messages can be re-transmitted on the connections by which they have arrived at the switch or on a different one.

Features:

* Modularity is good.

* Fault tolerance and reconfiguration possibilities are less than that of a global bus.

* The complexity of the communications control grows rapidly with an increase in the number of switches.

Other structures of interconnection are possible, such as combinations of those
described earlier [4,9].

3.8 CONFLICT RESOLUTION

3.8.1 GENERAL

Sharing of resources leads to the possibility that deadlocks (permanent unresolved access) may occur and that interference (two or more processors attempting to access the same resource) between the processors may cause a reduction in the throughput of the system. The three most widely used methods to cope with simultaneous access requests to the same resource by several processors are: arbitration, flag test-and-set, and interrupt.

3.8.2 ARBITRATION

The arbiter is a hardware structure which accepts requests made by the processors, resolves conflicts and advises all processors of its decision. Daisychain, fixed priority or dynamic priority [13] decision methods are used, the choice depending on the characteristics of the system. The AN7 single chip multiprocessor arbiter designed by Signetics is an example of hardware arbitration [26].

3.8.3 TEST-AND-SET CAPABILITY

When the test and set arbitration method is used, the shared resource possesses a flag which indicates the state of the resource. When a processor requires
that resource, it examines the state of the flag. If it is "busy", the processor
waits; if "ready", the processor access the shared resource. The user
processor first sets the indicator flag to "busy", uses the resource, and on
completion resets the flag to "ready".

3.8.4 INTERRUPTS

In single processor systems interrupts are used for signalling errors, event
counting, control loop timing, management of slow peripherals, etc.. In a
multiprocessor system the interrupts can also be used for synchronization of
communications between the individual processors [20,27].

3.9 MAILBOXES

All the three conflict resolution methods explained in the previous section can
be used for communications between processors. Most of these communications
take place through a memory known as a "mail box". The mail box memory is a
resource shared by the processors. A sending processor puts information into
the mail box, then the receiving processor looks to see if there is 'a letter for
it'. Alternatively the sender may advise the addressee that information is
waiting to be collected.
REFERENCES FOR CHAPTER 3


9. Lillevik, S., et al 'A multiprocessor with replicated shared memory' National


18. Mazare, G., 'MCS - A symmetric multi-micro-processor system' EUROMICRO

20. Balph, T., 'Multiple processor control systems provide higher performance and improved diagnostics' Control Engineering (USA), Vol. 30, No. 11, pp76-84 (October 83).


23. 'Military standard - aircraft internal time division command/response multiplex data bus' Department of the air force (USA), MIL-STD-1553A, April 1975.


Fig. 3.1 General Structure of SIMD Processor.
FIG. 3.2 GENERAL STRUCTURE OF MIMD PROCESSOR.
FIG. 3.3 GENERAL STRUCTURE OF COMPUTER NETWORKS.
FIG. 3.4 GENERAL STRUCTURE OF MULTIPROCESSOR SYSTEMS
FIG. 3.5 A TYPICAL MICROCOMPUTER.
FIG. 3.6 SHARED BUS COMMUNICATION.
FIG 3.9  Multibus multiport structure
FIG. 3.11 TIGHTLY COUPLED SYSTEM.
FIG. 3.12 LOOSELY COUPLED SYSTEM.
FIG. 3.13 CLASSIFICATION OF INTERCONNECTION TOPOLOGY.
FIG. 3.14 RING STRUCTURE.
FIG. 3.15 COMPLETELY INTERCONNECTED STRUCTURE.
FIG. 3.16 CENTRAL MEMORY ARCHITECTURE.
FIG. 3.18 STAR STRUCTURE.
FIG. 3.19 RING STRUCTURE WITH SWITCH.
FIG 3.21  Regular Network Structure
FIG. 3.22 IRREGULAR NETWORK STRUCTURE.
Chapter 4

TOKEN PASSING BUS ACCESS METHOD

4.1 MAIN FEATURES OF THE TPBAM

The token passing bus access method (TPBAM) has some very important features which can be summarised in the following three points:

(a) Fair access to the bus: The method is fair in the sense that it offers each station an equal share of the bus.

(b) Reconfigurable: The method handles addition and deletion of stations very easily and without any modification to the hardware or the software (protocol).

(c) Deterministic: The method provides computable deterministic, worst case bounds on access delay for any given network. This feature is essential in real time systems where system response time must be guaranteed.

4.2 BASIC CONCEPT

Basically the Token Passing Bus Access Method (TPBAM) allows a series of bus connected units ('stations') to communicate as a ring structure [1,2]. Such a
situation is shown in Fig. 4.1 where a number of stations, each one having a unique address, are coupled to a shared bus. The right to use the bus is transferred from station to station, thus forming a logical ring (dotted line Fig. 4.1). When a station has this right it is said to hold the 'token'.

Each station knows the addresses of the preceding station (Previous Station (PS)), the station following it (Next Station (NS)) and finally its own address (This Address (TS)).

4.3 USE OF THE BUS

At any given time one station, and only one station, holds the token, and is obligated to pass it on when finished with it.

Each station can only hold the token for a limited period of time. This means that maximum time taken by the token to traverse the network is defined, i.e., access to the system bus is deterministic.

Bus operation proceeds as follows (Fig. 4.1); for explanatory purposes assume that the network has already been initialised, meaning that the logical ring has been established. At this instant the token is held by station 4 (the station whose address = 4).

When station 4 (This Station, TS) is ready to pass on its bus access right (the token) it sends the token to the station number known to it as NS (Next Station), in this case 7. Once this is received by station 7 it can transmit its message(s). These can be sent directly to any station within the ring, i.e.,
they don't propagate through the system station by station. When station 7 is ready to pass the token, it sends a message to its NS (station 9) and so the cycle continues in a circular fashion, from station 9 to 14 to 4.... In this way the token is passed from one station to the next in a logical ring.

Notice that station numbers need not be contiguous. As shown here the relatively arbitrary station numbering poses no inefficiency to the access method. The value of this is the ability to add and remove stations to the network without re-arranging established addresses.

4.4 RING INITIALISATION AND MAINTENANCE

The following functions, at a minimum, must be performed by one or more stations on the bus:

* Ring initialisation.
* Ring reconfiguration: Addition of a station.
* Ring reconfiguration: Deletion of a station.

4.4.1 RING INITIALISATION

On power-up each station within the ring can be supplied with its own address, either from software or hardware. Unfortunately it has no knowledge of the address of its previous or next stations and so cannot form the logical ring. Hence it is necessary to incorporate an active initialisation process (Fig. 4.2) to get the ring going in the first place. This is accomplished through the use of a hardware timer called the Response timer, the corresponding time-out being
the Response Time (RT). RT is directly proportional to the station address, i.e., the lowest address has the shortest time-out period. In the example of Fig. 4.1 this would be station 4.

On power up all timers are activated simultaneously. The first one to time-out (lowest address, 4) sends out a bus message called 'claim token'; as a result all other stations reset waiting for further bus messages. At this point station 4 has the token; however it still has no information concerning its predecessor (PS) or its successor (NS).

It now solicits its successor using a message (control frame) called 'who follows'. Included in this is the address of the sending (token holding) station. All other stations respond by activating their response timers. The first one to time-out is that with the next highest address in the system (7); it reacts by sending a control frame called 'set successor' to the sender station (4), informing the sender that it is next in sequence. Station 4 then passes the token to 7 (its successor) and waits for a confirmation of reception. 7 confirms reception by sending a control frame called 'token ack' to 4. At this stage the link between 4 and 7 is patched in.

Station 7 now goes through the same procedure to link up to 9. Note that when it issues 'who follows' only stations which haven't established a PS value respond, which includes the first station in the link, 4. If the time-out of 4 is left at its initial value the 7 would patch a link to 4 and the ring would appear to be formed correctly (although only two stations are involved). To prevent this happening the first station to get the token resets its response timer to a much longer value, longer, in fact, than that of the highest numbered station in
the system. Therefore, when 7 transmits the 'who follows' message 9 will have the shortest response time and so will link up to 7.

This process is repeated at each station, the logical ring being completely formed when the first station receives the token from the last one in the ring. It can be seen that when the last station (19) issues 'who follows' only the first one (4) can respond, patching in the final link in the ring. It then sends a 'set last' control frame to all stations defining the address of the last station in the ring. This is followed by a message which specifies the total number of stations in the system.

Initialisation is completed when the first station finally sends out 'init done' message, the system now entering the steady state condition. At this stage each station has established a value for the following:

* This station's own address (TS).

* The next station's address (NS).

* The previous station's address (PS).

* The first station's address (FS).

* The last station's address (LS).

* Number of stations in the ring (CS).

The above information is essential for every station in the ring in order to function successfully. Therefore, it must be updated whenever a change, such as addition or deletion of a station, has taken place in the ring.
4.4.2 ADDITION OF A STATION

Each station in the ring has the responsibility of periodically allowing new stations to enter the ring. The entry procedure is as follows. During normal operation a station holding the 'token' periodically issues a 'solicit successor' frame; this invites stations with an address between itself and the next station in logical sequence to demand entrance. The transmitting station then waits for a time relative to the next station address (because the address of any station between TS and NS cannot exceed the NS address). Two events can occur:

- No response: Nobody wants to get in the ring. The token holder transfers the token to its successor as usual.

- One response: One station issues a 'set successor' frame. The token holder sets its NS to be the new station and transmits the token to it.

When a number of stations numbered between TS and NS are waiting to join the ring that with the lowest address responds first and gains entry as described above. The others have to wait for another invitation to enter the ring.

The addition of the new station to the ring leads to the following changes in the information each station has about the ring (Fig. 4.3):

* All stations in the ring increase their record of the number of stations in the ring by one.

* The station that has sent the solicit successor frame sets its NS to be the new station.
* The station which was next to the station that has sent the solicit successor frame sets its PS to be the new station.

* If the new station's address is greater than the last station then all stations update their LS to be the new station (Fig. 4.4).

* If the new station's address is less than the first station then all stations update their FS to be the new station (Fig. 4.5).

4.4.3 DELETION OF A STATION

A station may exit from the ring either as a result of a station fault or else as part of the normal operating procedures.

(a) Station failure: Failure may be detected in one of two ways. In the first case the fault is detected when a token has been sent to the defective station by the previous station (PS) and no response has been detected. The sending station assumes a failure and starts reconfiguring the ring by sending a 'who follows' frame. The next operational station in the logical ring responds by sending a 'set successor' frame; the sending station then sends the token and its own address to its new (NS).

In the second situation a station may fail whilst actually holding the token. This condition is detected by one of the (good) stations waiting to receive the token when its 'token rotation time' timer times out. Should this occur the loss of the token is recognised and a 'claim token' frame is sent out.

(b) Station drop out: If a station wishes to drop out it waits until it receives the token, then it sends a 'set successor' frame to its predecessor. However
this differs from all previous cases in that the NS address is not that of the
station which wants to drop out. It is in fact the NS address which had been
held by the drop-out station.

The deletion of the new station to the ring leads to the following changes in the
information each station has about the ring (Fig. 4.6):

* All stations in the ring decrease their record of the number of stations in the
ring by one.

* The station that has sent the 'who follows' frame sets its NS to be the station
next to the failed station.

* The station which was next in succession to the failed station sets its PS to
be the station that has sent the 'who follows' frame.

* If the failed station is the last station then all stations update their LS to be
the station that has sent the 'who follows' frame (Fig. 4.7).

* If the failed station is the first station then all stations update their FS to be
the station next to the sender of the 'who follows' frame (Fig. 4.8).

4.5 FRAME FORMAT

This section describes the frame formats used in sending messages between
stations. All frames sent or received by stations conform to the following
general format:
Where:

SD: Start delimiter  
DML: Data message length  
FT: Frame type  
DA: Destination address  
SA: Source address  
DATA: Data  
ED: End delimiter

SD: Start delimiter (one byte)
DML: Data message length (one byte)
FT: Frame type (one byte)
DA: Destination address (one byte)
SA: Source address (one byte)
DATA: Data (one to 120 bytes)
ED: End delimiter (one byte)

The Data Message Length field contains the number of bytes in the data field.

The Frame Type is the most important field, simply because it determines what type of response is needed when a message has been received. The following is a list of the frame types:

"Claim Token"
"Token"
"Token Ack"
"Who Follows"
"Solicit Successor"
"Set Successor1"
"Set Successor2"
"Set Previous"
"New Member"
"Delete Member"
"Member Request"
"Member Count"
"Who Last"
"Set Last"
"Who First"
"Set First"
"Init Done"
"Data"

Refer to Appendix A for the full description of the control frames.
4.6 TIMERS

A number of logical timers are used in applying the token passing bus access method, as follows;

(a) Token Hold Time (TH)
(b) Token Lost Time (TL)
(c) Response Time (RT)
(d) Token Acknowledge Time (TA)
(e) Who Follows Time (WF)
(f) Solicit Successor Time (SS)

(a) Token Hold Time

This time controls how long a station can hold the token. The station may send data and control frames as long as the Token Hold Time has not expired.

(b) Token Lost Time

This time controls how long a station should wait for the token before assuming that the token is lost and issuing a 'claim token' frame. This time is equal to:

\[ TL = (TH \times N) + SM \]

Where TL is the Token Lost time, TH is the Token Hold time, N is the number of stations in the ring and SM is safety margin time.

(c) Response Time
Without taking design precautions it would be possible for simultaneous bus accesses to be made by a number of stations under the following conditions:

* After power-up.

* After receiving a 'who follows' frame.

* After receiving a 'solicit successor' frame.

This would result in a collision situation, leading to either temporary or permanent bus failure. To prevent such collisions a set of response timers are used, one in each station.

The response timer time is unique for every station, being related to its own unique address. It prevents stations making simultaneous access to the bus in the following circumstances:

(i) After power-up: In this case the station with the lowest address sends a 'claim token' frame since its Response Time expires first.

(ii) After receiving a 'who follows' frame: In this case the first station after the sender of the control frame responds before the other stations by sending a 'set successor' frame.

(iii) After receiving a 'solicit successor' frame: In this case if there are more than one station who are wishing to enter the ring the station with the lowest Response Time enters the ring by sending a 'set successor' frame. The other stations continue waiting for another invitation.

(d) Token Acknowledge Time
It controls how long a station waits for a 'token ack' frame after passing the token to its successor.

(e) Who Follows Time

It controls how long a station waits for a 'set successor' frame after sending a 'who follows' frame. It is equal to:

$$RT(LS) + SM$$

Where $RT(LS)$ is the Response time of the last station in the ring and SM is safety margin time.

(f) Solicit Successor Time

It controls how long a station waits for a 'set successor' frame after sending a 'solicit successor' frame. It is equal to:

$$RT(NS) + SM$$

Where $RT(NS)$ is the Response time of the next station in the ring and SM is safety margin time.
REFERENCES FOR CHAPTER 4


THIS STATION'S ADDRESS: TS
NEXT STATION'S ADDRESS: NS
PREVIOUS STATION'S ADDRESS: PS

FIG. 4.1 TOKEN PASSING ON A LOGICAL RING.
<table>
<thead>
<tr>
<th>Time Slot</th>
<th>Station 4</th>
<th>Station 7</th>
<th>Station 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power On</td>
<td>Read Own Address</td>
<td>Read Own Address</td>
<td>Read Own Address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1</th>
<th>Load and Start</th>
<th>Response Timer</th>
<th>Load and Start</th>
<th>Response Timer</th>
<th>Load and Start</th>
<th>Response Timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>Time Out Occurs</td>
<td>Still Running</td>
<td>Time Out Occurs</td>
<td>Still Running</td>
<td>Time Out Occurs</td>
<td>Still Running</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T3</th>
<th>Send Out 'Claim Token'</th>
<th>Stop Timer</th>
<th>Stop Timer</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>T4</th>
<th>Send Out 'Who Follows'</th>
<th>Wait for Response</th>
<th>Start Timer</th>
<th>Start Timer</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>T5</th>
<th>Send 'Set Successor'</th>
<th>Stop Timer</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>T6</th>
<th>Send 'Token' To 7</th>
<th>Receive 'Token Ack'</th>
<th>Send 'Token Ack'</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>T7</th>
<th>Send 'Who Follows'</th>
<th>Start Timer</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>T8</th>
<th>Still Running</th>
<th>Timeout Occurs</th>
<th>Send 'Set Successor'</th>
<th>Set PS</th>
</tr>
</thead>
</table>

| T9 | Wait | Send 'Token' To 14 | Receive 'Token Ack' | Receive 'Token' | Send 'Token Ack' |

**Fig. 4.2 Ring Initialisation Process.**
FIG. 4.4 ADDITION OF A STATION (CASE 2).
FIG. 4.5 ADDITION OF A STATION (CASE 3).

TS = THIS STATION
NS = NEXT STATION
FS = FIRST STATION
LS = LAST STATION
CS = NUMBER OF STATIONS AFTER RECONFIGURATION

THE SENDER
THE INVITED STATION
LOGICAL RING

TS = 10
NS = 7
FS = 4
LS = 10
CS = 5.6

TS = 7
NS = 8
FS = 3.2
LS = 10
CS = 5.6

TS = 4
NS = 3
FS = 3.2
LS = 10
CS = 5.6

TS = 2
NS = 3
FS = 3.2
LS = 10
CS = 1.6

4.18
FIG. 4.6 DELETION OF A STATION (CASE 1).

TS = THIS STATION
NS = NEXT STATION
PS = PREVIOUS STATION
FS = FIRST STATION
LS = LAST STATION
CS = NUMBER OF STATIONS

, AFTER RECONFIGURATION

THE DELETED STATION

THE WHO FOLLOWS SENDER
FIG. 4.7 DELETION OF A STATION CASE(2).
Fig. 4.8 Deletion of a Station (Case 3).
Chapter 5

IMPLEMENTING THE TPBAM IN A MULTIPROCESSOR SYSTEM

5.1 OVERVIEW

When processing units within a multiprocessor system contain their own memory and I/O the resultant structure resembles that of a distributed computer network. Therefore most aspects of computer networks can be applied to the design and evaluations of multiprocessor systems of this type.

The aim here is to implement a powerful reconfigureable multiprocessor system using a series of single board computers (SBCs), linked together via a fast simple backplane bus, interboard communication being handled by the Token Passing Bus Access Method.

Three basic design criteria were established;

(a) A standard bus communications method would be used irrespective of board functions.

(b) From the outset the system would be designed to support high speed data transfer on the bus.

(c) The computing elements would be processor independent.
One processor in a station may not be fast enough to handle both computation and communication tasks concurrently. For this reason a processor is dedicated to each task. Thus each station in the ring has two main blocks (Fig. 5.1), a Network Interface Block (NIB) and a main Processor Block (MPB). The functions of each block are described in the following sections.

5.2 SYSTEM INTERFACING

There are two distinct interfacing tasks involved here. Firstly each station has to manage the flow of information sent over the system bus; secondly, within each station, data exchange between the MPB and the NIB must be supervised and controlled.

The main function of the NIB is to handle all communications activities including ring initialisation and maintenance, so isolating the main processor from the system bus. It thus removes a considerable burden, both in terms of software and time, from this processor. From the main processor's point of view the NIB is a device to which it sends messages for onward transmission and from which it receives incoming data. Its functional block diagram is shown in Fig. 5.2, consisting of the following subfunctions:

- **Network control**: Performs all network control functions.

- **Parallel bus data handling**: Transmits and receives information over the parallel backplane bus.

- **Main Processor data handling**: Stores data written by the main processor
for transmission to other stations and transfers received data to the main processor.

Interfacing and data transfer is designed to be fast and simple, being organised as follows (Fig. 5.3);

(a) system bus interfacing: A total of 15 lines (excluding power supplies) are used on the backplane bus. These consist of;

* Address - four  
* Data - eight  
* Control - four

Control functions are

* General startup command - START*  
* Address validator - SSS*  
* Data strobe - WRT*  
* Transfer control - BSY*

A typical message transfer sequence is shown in Fig. 5.4.

For critical systems where fault degradation must be gradual single point failures need to be eliminated. In a bus based processor system the bus itself (and its associated drivers / receivers) give rise to such a situation. Hence in such application the bus must, at the very least, be duplicated. By using a simple structure such as the one devised here the backplane bus may be replicated at a relatively low cost for use with standard Eurocard size backplanes.

(b) NIB-Main processor interfacing: All data is exchanged between the network interface block and the main processor using direct memory access (DMA)
techniques. The DMA controller is located in the main processor section and generates the required control signals (read, write, and chip select). Authority for the control of all data transfers resides with the NIB; however the main processor may request service through the use of interrupt signals. Thus any CPU (e.g. 8086, 186, 8085, 6800, etc.) may be used in the main processor section provided the interface meets the NIB requirements.

5.3 THE NETWORK INTERFACE BLOCK (NIB)

5.3.1 GENERAL

Fig. 5.5 shows a more detailed functional diagram of the NIB, the main subsystems being:

* Scratch Pad Memory (SPM).

* Access Control Block (ACB).

* Communication CPU (Com.CPU).

* Station Select and Address Recognition (SSAR).

* Timing and Control (TC).

* System Bus Buffers (SBB).

The NIB has five modes of operation, as follows:

* Transmission mode: in this mode the NIB transfers data from the local station
to the system bus.

* Reception mode: Here the NIB receives data from the system bus.

* Transmission to MP mode: in this mode the NIB transfers data to the main processor.

* Reception from MP mode: The NIB receives data from the main processor.

* Idle mode: in this mode the NIB is waiting, ready to respond to requests to perform data transfer functions.

5.3.2 SCRATCH PAD MEMORY

The scratch pad memory is used as a temporary shared storage area for bus messages (Fig. 5.6), incoming messages being stored in a reception area while outgoing ones are deposited in a transmission region. The actual location and size of these are set by the Com.CPU.

It has two ports, port A and port B, each one giving access to the RAM storage area. Port A is shared between the Com. and the main processor while port B is connected to the system bus. Accesses are mutually exclusive, controlled by the Access control Block (ACB). Normally access to port A is given over to the Com.CPU but is transferred to the main processor when it wishes to read or write to the SPM. Control of port B is given over to the local (own station) TC during transmission mode. However it is transferred to the remote (sending) TC in reception mode.
5.3.3 ACCESS CONTROL BLOCK (ACB)

The function of this block is to prevent simultaneous access to the SPM from different devices. Highest priority is given to the Com.CPU because it is responsible for preparing the SPM for any of the NIB modes. The second priority is given to the remote TC in message reception mode. Third priority is given to the main processor for writing and reading messages. Lowest priority is given to the local TC in message transmission period. The ACB has the ability to suspend message reception when the SPM is being accessed by the Com. or the main processor.

5.3.4 COMMUNICATION CPU

The block diagram of the Com.CPU is shown in Fig. 5.7, its functions which are implemented in software being to;

* Establish the station's own address by reading it from the station select and address recognition block (SSAR).

* Perform memory management of the SPM. The Com.CPU divides the SPM into reception and transmission area(s). It also sets the SPM before any transmit or receive operation.

* Resolve access contention problems for the SPM.

* Detect the arrival of messages sent over the system bus to the station. The SSAR block notifies the Com.CPU of the existence of a message received in the SPM.
* Set up station address for outgoing messages.

* Initiate transmission to other stations in the system once:

  (i) the destination address has been set up.

  (ii) the data is set in the RAM transmission area.

* Recognise type of message received. The received message is either a data or control frame. If it is a control frame then the Com.CPU performs some analysis to determine the type of control frame received.

* Respond to control frames. This response depends on the type of control frame received.

* Control data exchange with the main processor.

* Handle serial communications to a VDU terminal.

5.3.5 STATION SELECT AND ADDRESS RECOGNITION BLOCK

This block is responsible for the following tasks(See Fig. 5.8):

* Putting out the address of the destination station written to it by the Com.CPU.

* Detecting the presence of broadcast (all stations) address.

* Detecting its own address on the system bus.

When the NIB intends to transmit a message to a station in the bus, the address of the destination station is written to the SSAR. The SSAR saves this address
until the NIB enters the transmission mode where the saved address is put out onto the system bus. The other stations on the bus then compare this address with their own and the broadcast address. The station(s) which has an address equal to that on the bus enters the reception mode.

5.3.6 TIMING AND CONTROL BLOCK

The function of this block is to generate control signals for the local and remote SPMs in transmission mode (See Fig. 5.9). In transmission mode data is read from the local SPM and written to the remote SPM. The TC generates the read and write signals with the correct timing requirements by the SPM. It has the ability to detect end of message transmission. When the end of transmission frame is detected a signal is generated to inform the Com.CPU of this condition; the processor then stops the transmission mode.

5.3.7 SYSTEM BUS BUFFERS

This block includes the data and control buffers of the system bus. Their function is to ensure that all system bus signals have the ability to drive the system bus and all devices connected onto it. These buffers are enabled only in reception or transmission modes, their direction being determined by the mode of operation (reception or transmission).

5.4 FUNCTIONS OF THE MAIN PROCESSOR BLOCK

The main processor has its own memory and I/O devices. Computations needed
for certain applications are carried out in this processor. It is completely isolated from the system bus and has nothing to do with communication protocols (See Fig. 5.1). When it has a message to be transmitted to other stations it writes it to the NIB; when the NIB receives a data frame from a remote station it informs the main processor to read it. In both cases access to the SPM must be granted by the ACB to the main processor prior to the start of message transfer between the NIB and the main processor. Details of the design of the main processor is explained in chapter 7.
FIG. 5.2 FUNCTIONAL BLOCK DIAGRAM OF THE NETWORK INTERFACE BLOCK.
FIG. 5.3 SYSTEM INTERFACING.
FIG. 5.4 TYPICAL MESSAGE TRANSFER SEQUENCE.
FIG. 5.5 NIB MAJOR SUB-SYSTEMS.
FIG. 5.7 THE COM. COMPUTER.
FIG. 5.8 THE STATION SELECT AND ADDRESS RECOGNITION.
FIG. 5.9 TIMING CONTROL BLOCK.
Chapter 6

HARDWARE DESIGN OF THE NETWORK INTERFACE BLOCK

6.1 GENERAL

The NIB is based on an Intel 8031 single chip processor together with an 8K Byte (2764) EPROM. As the processor on-chip RAM is sufficient for the task in hand no external RAM chips are used. Address latching and decoding are provided as necessary. Central to the operation of the NIB is the scratch-pad memory, this consisting of a Texas Instruments TMS9650 dual-port RAM and a set of bus switches. Standard bus tracheivers (74LS245) are used for bus switching, similar techniques being used on the backplane. Programmable array logic (PAL) devices are used to implement access control (ACB) station select/address recognition (SSAR) functions whilst LS TTL is used for timing and control block (TC).

Serial I/O communications are supported by the on-chip UART of the 8031. Its primary purpose is to provide a back up communications link between processors in the event of bus faults. However for development work it has been used as a system development tool in conjunction with a display/keyboard unit.
6.2 SCRATCH PAD MEMORY

6.2.1 OVERVIEW

The SPM is based on the use of the TMS9650 dual port RAM. Data may be accessed by the Com.CPU, main processor, or the parallel bus. However, programming (setting internal registers) of the TMS9650 can only be done by the Com.CPU. The TMS9650 has two ports; port A and port B (details in Appendix B). Data from a 256 Byte RAM can be accessed from either port.

6.2.2 TMS9650 PORT A INTERFACE

Port A interface is shared by the Com.CPU and the main processor (Fig. 6.1). Each processor accesses this port of the RAM chip via control and data signals, the ACB determining which processor may use the TMS9650. The main processor is not allowed to select any of the RAM internal registers, these being under the control of the Com.CPU.

The data and control signals of both CPUs are buffered before connection to port A, using 74LS245 and 74LS125 bus buffers. Both devices are controlled by the ACB to prevent simultaneous accesses to port A. The select lines of port A are set directly by the Com.CPU. The signals type and function are described in the following table.
Table 6.1 Port A interface signals description

<table>
<thead>
<tr>
<th>Signal</th>
<th>Source type</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS0-AS2</td>
<td>Com.CPU</td>
<td>Select TMS9650 internal registers</td>
</tr>
<tr>
<td>CD7-CD0</td>
<td>Com.CPU</td>
<td>I/O  Data signals</td>
</tr>
<tr>
<td>CRD*</td>
<td>Com.CPU</td>
<td>I  Data output enable</td>
</tr>
<tr>
<td>CWRT*</td>
<td>Com.CPU</td>
<td>I  Data write enable</td>
</tr>
<tr>
<td>CommEn*</td>
<td>ACB</td>
<td>I  Enable Com.CPU buffers to the TMS</td>
</tr>
<tr>
<td>CSA*</td>
<td>ACB</td>
<td>I  Chip select signal to port A of the TMS</td>
</tr>
<tr>
<td>MD7-MD0</td>
<td>Main CPU</td>
<td>I/O  Data signals</td>
</tr>
<tr>
<td>MRD*</td>
<td>Main CPU</td>
<td>I  Data output enable</td>
</tr>
<tr>
<td>MWRT*</td>
<td>Main CPU</td>
<td>I  Data write enable</td>
</tr>
<tr>
<td>MainEn*</td>
<td>ACB</td>
<td>I  Enable Main CPU buffers to the TMS</td>
</tr>
<tr>
<td>EINT*</td>
<td>SPM</td>
<td>O  End of message transmission interrupt</td>
</tr>
</tbody>
</table>

Notes:

1. The * symbol means that the signal is active low.
2. Set means that the signal is at positive logic "1".
3. Clear means that the signal is at positive logic "0".
4. O means output.
5. I means input.

The INT* signal in the TMS9650 is used to detect end-of-message transfer.

6.2.3 TMS9650 PORT B INTERFACE

Port B is connected to the system backplane bus buffers (Fig. 6.2). It is used to handle the transfer of data into and out of the RAM. Information is transmitted and received in byte serial form, the RAM being used as a temporary store for this information. Port B interface has three modes of operation:
(a) Deselect: The NIB is not in receive or transmit mode.

(b) Transmit: The NIB is in transmission mode.

(c) Receive: The NIB is in reception mode.

The signals type and function are described in the following table.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Source</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD0-SD7</td>
<td>System bus</td>
<td>I/O</td>
<td>Data signals</td>
</tr>
<tr>
<td>CSB*</td>
<td>ACB</td>
<td>I</td>
<td>Chip select to port B</td>
</tr>
<tr>
<td>LRD*</td>
<td>TC</td>
<td>I</td>
<td>Data output enable</td>
</tr>
<tr>
<td>RWRT</td>
<td>System bus</td>
<td>I</td>
<td>Data write enable</td>
</tr>
</tbody>
</table>

In transmission mode the local TC generates a LRD* signal to enable data from port B in the local TMS9650 followed by a RWRT* signal to write the data in port B of the remote TMS9650. This way of transferring data is similar, in some way, to a DMA transfer performed by a DMA controller.

In reception mode data coming from the backplane is written to the TMS9650 via port B.

6.3 ACCESS CONTROL BLOCK

The function of this block is to prevent simultaneous accesses being made to the SPM by the main processor, the Com.CPU, local TC, and the remote TC. This is implemented through the use of a Programmable Logic Array (PAL).
device (refer to Appendix B for a full description of the device and the Boolean equations used to program it for the required function). Its input signals are;

<table>
<thead>
<tr>
<th>Input signal</th>
<th>Source</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>READY</td>
<td>Com.CPU</td>
<td>The SPM is ready for reception</td>
</tr>
<tr>
<td>SELECT</td>
<td>Com.CPU</td>
<td>Select access to the SPM</td>
</tr>
<tr>
<td>CS1*</td>
<td>Com.CPU</td>
<td>Chip select signal from Com.CPU</td>
</tr>
<tr>
<td>CSM*</td>
<td>Main CPU</td>
<td>Chip select signal from Main</td>
</tr>
<tr>
<td>CS4*</td>
<td>Com.CPU</td>
<td>Chip select signal from Com.CPU</td>
</tr>
<tr>
<td>CS5*</td>
<td>Com.CPU</td>
<td>Chip select signal from Com.CPU</td>
</tr>
<tr>
<td>CS6*</td>
<td>Com.CPU</td>
<td>Chip select signal from Com.CPU</td>
</tr>
<tr>
<td>WRT*</td>
<td>Com.CPU</td>
<td>Write enable signal from Com.CPU</td>
</tr>
<tr>
<td>RXEN*</td>
<td>SSAR</td>
<td>Receive enable</td>
</tr>
<tr>
<td>TXEN*</td>
<td>TC</td>
<td>Transmit enable</td>
</tr>
</tbody>
</table>

The output signals from this block are as follows:

<table>
<thead>
<tr>
<th>Output signal</th>
<th>Destination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Com.En*</td>
<td>Com.CPU buffers</td>
<td>Enable buffers to the SPM</td>
</tr>
<tr>
<td>MainEn*</td>
<td>Main CPU buffers</td>
<td>Enable buffers to the SPM</td>
</tr>
<tr>
<td>CSA*</td>
<td>Port A of SPM</td>
<td>Chip select port A</td>
</tr>
<tr>
<td>CSB*</td>
<td>Port B of SPM</td>
<td>Chip select port B</td>
</tr>
<tr>
<td>BUSY</td>
<td>System bus buffer</td>
<td>SPM is busy</td>
</tr>
<tr>
<td>DMA0</td>
<td>Main CPU</td>
<td>Channel 0 DMA request</td>
</tr>
<tr>
<td>DMA1</td>
<td>Main CPU</td>
<td>Channel 1 DMA request</td>
</tr>
<tr>
<td>INTA*</td>
<td>Main CPU</td>
<td>Clear interrupt flag</td>
</tr>
</tbody>
</table>

The Com.CPU has the first priority in accessing the SPM. Whenever the SPM is being accessed by either the Com.CPU or main processor the Com.CPU pulls the READY signal low. In this case, if a remote station wishes to transmit a
message to this station, the READY signal causes the BUSY* signal on the system bus to be active. The BUSY* signal suspends message transmission in the remote station. When the Com.CPU returns the READY signal to a "1" state the BUSY* signal goes inactive. Now the SPM is ready to receive a message from the remote station.

The DMA0, DMA1 and INTA* signals are used for message transfer between the NIB and the Main Processor. They are explained in detail in section 6.8.

6.4 COMMUNICATION CPU

6.4.1 GENERAL

The single chip computer 8031 is used as the Com.CPU. Its main specifications are as follows;

* Four 8-bit ports; P0,P1,P2 and P3.

* Two 16-bit timers ;T0 and T1.

* Internal USART.

* 256 byte internal RAM.

* Interrupt controller.

The 8031 is configured as shown in Fig. 6.3.
6.4.2 CLOCK

The 8031 contains an on-chip oscillator and clock circuit but needs an external crystal and capacitor. In this design a 6.144 MHz clock is used, though a 12 MHz clock could be used provided other components have sufficiently fast access time.

6.4.3 POWER-ON RESET

A low to high transition on the RST/VPD pin of the 8031 resets the processor. A small internal resistor permits power-on reset using only a capacitor connected to the +5V rail. A push-button has also been provided to reset the processor manually.

6.4.4 ADDRESS/DATA BUFFERS

The lower 8 address bits (A0-A7) are obtained from the 8031 multiplexed address/data bus which is available on port P0. The 8031 also generates an address latch enable (ALE) signal which indicates when the address/data bus carries address information. An 8 bit latch may be used in conjunction with ALE to hold the address for use by the EPROM. The device selected for this purpose is the 74LS573.

The 8 data bits (D0-D7) are fed to a bidirectional buffer (74LS245) the direction of which is controlled by the RD* and PSEN* (program store enable).

The higher 8 address bits (A8-A15) are left unbuffered.
6.4.5 CHIP SELECT LOGIC

The higher three bits of the address bus (A13-A15) are connected to a 3 to 8 decoder (Fig. 6.4). This produces 8 chip select signals, each signal accessing 8K bytes of memory address space. The memory map of the system is shown in Table 6.5.

Table 6.5 Memory map of the 8031 CPU

<table>
<thead>
<tr>
<th>Memory location</th>
<th>Active pin</th>
<th>Size</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-1FFFH</td>
<td>CS0*</td>
<td>8K</td>
<td>Select EPROM</td>
</tr>
<tr>
<td>2000-3FFFH</td>
<td>CS1*</td>
<td>8K</td>
<td>Select SPM</td>
</tr>
<tr>
<td>4000-5FFFH</td>
<td>CS2*</td>
<td>8K</td>
<td>Select SSAR</td>
</tr>
<tr>
<td>6000-7FFFH</td>
<td>CS3*</td>
<td>8K</td>
<td>Reserved</td>
</tr>
<tr>
<td>8000-9FFFH</td>
<td>CS4*</td>
<td>8K</td>
<td>Select interrupt flag</td>
</tr>
<tr>
<td>A000-BFFFH</td>
<td>CS5*</td>
<td>8K</td>
<td>Select DMA0</td>
</tr>
<tr>
<td>C000-DFFFH</td>
<td>CS6*</td>
<td>8K</td>
<td>Select DMA1</td>
</tr>
<tr>
<td>E000-FFFFH</td>
<td>CS7*</td>
<td>8K</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

6.4.6 PROGRAM MEMORY

An 8K bytes of EPROM program memory is provided using the 2764 device (Fig. 6.4). This device has a chip enable input for address decoding. This input is connected to CS0* of the chip select logic block. The EPROM has an output enable (OE*) pin to control its tristate output buffers. The OE* pin may be controlled by the 8031 program store enable signal (PSEN*), so that the EPROM is effectively only connected to the address bus during the intervals when the processor is fetching from memory.
6.4.7 INPUT/OUTPUT

The 8031 contains four I/O parts called P0, P1, P2, and P3. Each port is 8 bits wide and may be addressed as individual bits or as a collective byte. Some of the I/O lines have alternative special uses. These are:

P0: 8 bit multiplexed low order address/data for memory expansion.

P1: No special use. General purpose I/O only.

P2: 8 bit high order address bus for memory expansion.

P3: Bit 0 - Serial port receiver input (RXD)
    Bit 1 - Serial port transmitter output (TXD)
    Bit 2 - External interrupt request input (INT0*)
    Bit 3 - External interrupt request input (INT1*)
    Bit 4 - Timer/counter input (T0)
    Bit 5 - Timer/counter input (T1)
    Bit 6 - Write strobe for external data memory output (WR*)
    Bit 7 - Read strobe for external data memory output (RD*)

If any of these functions are not required, then the appropriate pins may be used for general purpose I/O.

In this application the ports are configured as follows:

Port 0 and Port 2 are used for accessing external program and data, while Port 1 and Port 3 are used as shown below:
Table 6.6 Port 1 and Port 3 configuration of the 8031

<table>
<thead>
<tr>
<th>Port No.</th>
<th>Bit No.</th>
<th>Type</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.0 - P1.2</td>
<td>0</td>
<td>AS0-AS2</td>
<td></td>
</tr>
<tr>
<td>P1.3</td>
<td>0</td>
<td>READY</td>
<td></td>
</tr>
<tr>
<td>P1.4</td>
<td>0</td>
<td>STX*</td>
<td></td>
</tr>
<tr>
<td>P1.5</td>
<td>0</td>
<td>SELECT</td>
<td></td>
</tr>
<tr>
<td>P1.6</td>
<td>I</td>
<td>START*</td>
<td></td>
</tr>
<tr>
<td>P1.7</td>
<td>0</td>
<td>WAIT</td>
<td></td>
</tr>
<tr>
<td>P3.0</td>
<td>I</td>
<td>RXD</td>
<td></td>
</tr>
<tr>
<td>P3.1</td>
<td>0</td>
<td>TXD</td>
<td></td>
</tr>
<tr>
<td>P3.2</td>
<td>I</td>
<td>INTO*</td>
<td></td>
</tr>
<tr>
<td>P3.3</td>
<td>I</td>
<td>INT1*</td>
<td></td>
</tr>
<tr>
<td>P3.4</td>
<td>0</td>
<td>CLR</td>
<td></td>
</tr>
<tr>
<td>P3.5</td>
<td>0</td>
<td>SAEN*</td>
<td></td>
</tr>
<tr>
<td>P3.6</td>
<td>0</td>
<td>WR*</td>
<td></td>
</tr>
<tr>
<td>P3.7</td>
<td>0</td>
<td>RD*</td>
<td></td>
</tr>
</tbody>
</table>

AS0-AS2 These bits select the internal registers of the TMS9650. The instructions SETB and CLR are used for setting the proper values for selecting each register.

READY This bit holds reception from other stations when the SPM is busy.

STX* This bit controls data transfer down the backplane.

SELECT This bit is used to select either the Com.CPU or the main processor to gain access to port A of the TMS9650.

START*, WAIT These signals are used for two functions; (a) guarantee that all the Com.CPUs in different stations start their initialisation programs at the same time. (b) Hold any newly added station from transmitting messages until a station sends a solicit successor frame. The relation between the two signals is shown in Fig. 6.5. The START* signal is active only when all the WAIT signals of the
stations in the ring are low. Each station first pulls its own WAIT signal low when it is ready and then tests its START* bit; when it goes low the station starts its initialisation program. In steady state operation all stations pull their WAIT signal low except the station that holds the "TOKEN", this implies that control over the START* signal transfers with the "Token". When a new station is added to the ring it can't start since its START* signal is not active. When the "Token" holder station wishes to invite new stations to the ring it pulls its WAIT signal low therefore causing the START* signal to become low. At this time the Com.CPU of the new station can starts its initialisation program.

RXD, TXD

These two pins are connected to an RS232 receiver and driver respectively.

INT0*

This pin is the first external interrupt signal to the Com.CPU, the interrupt signal being generated by the main processor. It represents a request from the main processor to the Com.CPU to transfer access to the main processor. It may also be used to indicate end of access to the SPM.

INT1*

This pin is the second external interrupt signal to the Com.CPU. The input signal to this pin is either from the SSAR (where it represents a request to set the NIB in receive mode) or it is from the TC block (where it
represents end of message transfer).

CLR* This signal is used to clear end of transmission interrupt flag (INT0*) which is set by the TC.

SAEN* This signal is cleared to enable the destination address stored in the SSAR onto the system bus prior to activating the STX* signal.

WR* RD* These are control signals used to access various devices in the system.

6.5 STATION SELECT AND ADDRESS RECOGNITION

6.5.1 FUNCTIONS

The functions of this block are to;

(a) Set own address.
(b) Perform address recognition.
(c) Set destination address.
(d) Control system bus addressing.

These tasks are explained in the next sections. A programmable logic array I.C.(PAL16P8) and a latch device are selected to perform all the tasks required. Refer to Appendix B for the complete description and programming of the PAL.
6.5.2 SETTING OWN ADDRESS

The address of each station is set manually using a 4 bit switch, the selected value being fed to the PAL. This device is programmed to enable a 4 bit data signals in the data bus (D0-D3) when the SSAR is selected and the RD* signal is active (Fig. 6.6). This enables the Com.CPU to read its own address on power-up.

6.5.3 ADDRESS RECOGNITION

The address selected by the 4 bit switch is fed to the PAL together with the address from the system bus. Comparison takes place when signal SSS* is active (Fig. 6.6); An RXEN* signal is generated when either of the following is true:

- The address in the system bus equals that set by the on board switch and the SSS* of the system bus is active.

- The address on the system bus equals 1111 (broadcast address) and SSS* is active.

The RXEN* signal can't be active when the NIB is in transmission mode.

6.5.4 SET DESTINATION ADDRESS

When sending out data onto the backplane bus the Com.CPU stores a 4 bit number in the SSAR. This defines the address of the station for which the message is intended. The Com.CPU uses its lower 4 bit data bus in writing this
address, qualified by the CS2* and WRT* signals. An 8 bit latch is used to do this job (74LS573). Only 4 bits are used of this latch device, the other 4 bits being reserved for future expansion.

6.5.5 SYSTEM BUS ADDRESSING

The address latch mentioned earlier is kept in a tristate condition until the Com.CPU activates the SAEN* signal, enabling the stored address onto the system bus together with the SSS* signal. These two, when active together, enable the destination station to recognise its own address and enter the reception mode.

6.6 TIMING AND CONTROL BLOCK

6.6.1 GENERAL

This block is responsible for generating the LRD* and RWRT* signals used for enabling data from the local SPM and writing it to the remote SPM. These signals comply with the timing requirements of the TMS9650.

The signals used in this block with their functions are described in the following table.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXEN*</td>
<td>0</td>
<td>Transmit enable</td>
</tr>
</tbody>
</table>

6-14
6.6.2 OPERATION OF THE TC

The transmission cycle starts when the STX* signal is set low by the Com.CPU. Provided the bus is free (BUSY* inactive) TXEN* now goes active to start message transfer under the control of signals LRD* and RWRT*. If BUSY* is active TXEN* is held off in the inactive state, preventing any data transmission until the bus is clear.

The end of message transmission is detected by the TC when it receives an interrupt signal from the SPM. This interrupt signal sets a flag to the Com.CPU. This flag causes the Com.CPU to execute a subroutine that clears this flag and sets the STX* signal. The timing diagram that explains the TC operation is shown in Fig. 6.7.

The control signals LRD*, LWRT*, and RWRT* are generated by a 32x8 PROM. The address input to this PROM is generated by a 4-bit counter which is driven by 16MHZ clock (Fig. 6.8). Byte transfer cycle starts at a low to high transition at Q2 of the counter and ends at the second low to high transition of the same signal. Appendix B shows the data stored in the PROM which generates the control signals with the desired timing.

The SPM is set to pull low an interrupt signal (EINT*) at end of message transfer. This signal sets a flip flop which its output is fed to the Com.CPU's interrupt input (INT1*). The subroutine of this interrupt clears this flip flop by
the CLR* signal.

6.7 SYSTEM BUFFERS

6.7.1 DATA BUFFER

An 8 bit bidirectional buffer is used to drive and receive data from the system bus (Fig. 6.9). This buffer is enabled when CSB* is active, in other words in transmission and reception modes. The direction of this buffer is controlled by the LRD* signal which implies that the buffer is always pointing to the SPM except the periods where data is being read from the local SPM.

6.7.2 CONTROL SIGNALS BUFFER

The control signals (Fig. 6.9) in the system bus are as follows:

- **RWRT*** Remote write
- **BUSY*** Station busy
- **START** Start initialisation program
- **SSS*** Station select strobe

6.8 NETWORK INTERFACE BLOCK MODES OF OPERATION

6.8.1 GENERAL

The NIB has four modes of operation; get message from main processor, transmit message to remote station, receive message from remote station and transfer
message to main processor. The selected mode of operation is determined by
the Com.CPU which receives requests for each mode. Each mode is explained in
the following four sections.

6.8.2 GET MESSAGE FROM MAIN PROCESSOR

This mode is requested by the Main processor when it has a message to
transmit to a remote station. The main processor sets an interrupt flag which
is fed to the INTO* input of the 8031 processor. The subroutine of this
interrupt does the following:

* Clears the interrupt flag by activating the INTA* signal.

* Clears the READY signal to hold any message reception from other stations.

* Points to the beginning of transmission area in the TMS9650.

* Selects the Data/Increment register in the TMS9650 via AS2-AS0.

* Clears the SELECT signal to enable the Main processor buffers to the
TMS9650.

* Activates DMA0 signal which initialise DMA transfer from the Main processor'
memory to the TMS9650 dual port RAM.

* Waits for another interrupt from the main processor which indicates end of
message transfer.

* Clears the interrupt flag by activating INTA*.

* Sets the SELECT signal to enable the Com.CPU buffers to the TMS9650.
6.8.3 TRANSMIT MESSAGE

This mode is entered when there is a message to be sent to a remote station by the station which holds the "Token". In this case the Com.CPU sets TMS9650 to point to the start of the message to be sent and programs the TMS9650 through its control register to activate an interrupt when last byte of the message has been transmitted. The NIB enters the transmit mode when the Com.CPU activates the SRX* signal. End of message transmission is detected when the signal EINT* from the TMS9650 is active. The activation of this signal then sets a flag which is fed to the INT1* input of the 8031 CPU. When the 8031 is interrupted by this flag it deactivates the STX* signal and clears the interrupt flag by the CLR* signal.

6.8.4 RECEIVE MESSAGE

Receive mode is requested when RXEN* signal is active. RXEN* signal causes an interrupt to the 8031 via the input INT1*. If the NIB is in the idle mode then the message coming from a remote station can be handled immediately. However if the NIB is in other modes then reception is suspended (due to the Com.CPU resetting the READY signal) from the Com.CPU. As soon as this mode is done then the Com.CPU sets the READY signal to enable reception. End of message reception is detected when the RXEN* and hence the INT1* goes inactive.

6.8.5 TRANSFER MESSAGE TO MAIN PROCESSOR

This mode is initiated by the Com.CPU when a Data Frame has just been
received in the SPM. A subroutine is then executed to do the following:

* Clears the READY signal to hold any message reception from other stations.

* Sets the TMS9650 to point to the beginning of the reception area in the TMS9650.

* Selects the Data/Increment register in the TMS9650 via AS2-AS0.

* Clears the SELECT signal to enable the Main processor buffers to the TMS9650.

* Activates DMA1 signal which initialise DMA transfer from the TMS9650 dual port RAM to the Main's memory.

* Waits for an interrupt from the main processor which indicates end of message transfer.

* Clears the interrupt flag by activating INTA*.

* Sets the SELECT signal to enable the Com.CPU buffers to the TMS9650.
FIG. 6.1 TMS9650 PORT A INTERFACE.
FIG. 6.3 8031 CONFIGURATION.
FIG. 6.4 CHIP SELECT AND PROGRAM MEMORY FOR THE 8031 CPU.
Fig. 6.5 START* and WAIT control signals.
FIG. 6.7 TIMING DIAGRAM OF MESSAGE TRANSFER SEQUENCE.
FIG. 6.9 SYSTEM DATA AND CONTROL BUS.
7.1 GENERAL

In the multiprocessor concept described here no assumptions are made about the structure of the main processor block. For some applications a separate processor may not even be used (e.g. display subsystems). However, where the design is used to support a distributed computing system each main processor will have its own memory and I/O devices. All application software runs in these processors. They are completely isolated from the system bus, having nothing to do with the communications activities. The main processor merely interchanges information with the communication processor; moreover the transfer protocol is kept simple by using a combination of interrupt and DMA interfacing between the two processors.

7.2 MAIN_PROCESSOR_SYSTEM_OVERVIEW
7.2.1 GENERAL

The main processor (Fig. 7.1) consists of the following main blocks:

(a) CPU section.
(b) Memory.
(c) Serial communication.
(d) NIB interface.

7.2.2 CPU SECTION

The CPU section is based on the use of an Intel 80188 processor together with an Intel 8087 numeric processor extension. An advanced bus controller (82188) is included to provide 188/8087 interfacing. The complete CPU section is composed of;

(a) Microprocessor: Processing power is provided by an Intel 80188 high integration 8-bit microprocessor [1], which includes the following internal units:

(i) Clock generator.
(ii) Programmable interrupt controller.
(iii) Programmable DMA controller.
(vi) Programmable chip select unit.
(v) Programmable timers.

(b) Hardware math unit: Support for fast maths operation is provided by an 8087 numeric co-processor [2].

(c) 82188 advanced bus controller: This controller is included to support 8087
interfacing with the 80188 [3].

(d) Address/data buffers: Buffers are included to increase the driving capability of the address and data signals.

(e) Power-on reset circuitry: This circuitry provides a reset signal to the 80188 after power-on and in response to a manual reset command.

(f) Single step control: A single step circuit is provided to allow for initial hardware testing and de-bugging.

(g) Watchdog timer: A watchdog timer is included to provide a means of escape should the processor crash.

7.2.3 MEMORY

The memory for the processor consists of EPROM, static RAM (SRAM) and optional dynamic RAM (DRAM) mounted on a piggy back board. Various sizes of EPROM (from 16K to 64K Byte) and SRAM (from 2K to 8K Byte) can be used in this design.

7.2.4 SERIAL COMMUNICATIONS

Two RS-232 serial communication channels are implemented using a Dual Universal Asynchronous Receiver/Transmitter (DUART) (Signetics 2681 [4]) and appropriate line interface circuits.

7.2.5 NIB INTERFACE
The function of the NIB interface is to facilitate data transfer between the main processor and the communication processor, this being carried out under the supervision of the 80188 DMA controller.

7.3 MAIN PROCESSOR BLOCK - HARDWARE DESIGN

7.3.1 THE CPU SECTION

(a) THE 80188 PROCESSOR The 80188 is a highly integrated microprocessor which combines a large number of the most common 8088 system components on a single chip. A block diagram of the 80188 is shown in Fig. 7.2. As shown here it consists of the a DMA unit, timers, interrupt controller, clock generator, and a chip select unit. All are housed in a 64 pin package, external circuit connections being shown in Fig. 7.3.

(i) CLOCK GENERATOR The 80188 includes a crystal oscillator. The inputs X1 and X2 provide an external connection for a fundamental mode parallel resonant crystal for the oscillator. The crystal frequency selected is double the CPU clock frequency. An 8MHZ crystal is used to generate a 4MHZ clock for the 80188.

(ii) INTERRUPT CONTROLLER The 80188 programmable interrupt controller (PIC) can handle interrupts which are generated by either software or hardware. A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0-31 are reserved for predefined interrupts which may be activated either by software or hardware. The software
interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the result of conditions specified by instructions (DIV, IDIV, etc.). The hardware interrupts are divided into two groups; internal and external. The internal interrupts are:

DMA 0 interrupt
DMA 1 interrupt
TIMER 0 interrupt
TIMER 1 interrupt
TIMER 2 interrupt

The external interrupts are:

INTO : connected to the 8087 numeric processor
INT1 : connected to the 2681 DUART
INT2 : not used
INT3 : not used
NMI : watchdog timer

All these interrupts are maskable except the NMI interrupt which is connected the watchdog timer.

The internal interrupts DMA 0 and DMA 1 are used in this design to detect DMA transfer termination. The external interrupt INTO is connected to the 8087 which uses it to indicate that unmasked exceptions have occurred during numeric instruction execution. INT1 is connected to the 2681 DUART which allows it to use interrupt handling of communication process instead of polled operation. For more details on programming modes of the PIC refer to [1].
(iii) DMA UNIT The DMA unit provides two high speed DMA channels. Data transfer can be performed to or from any combination of memory and I/O space in byte form. A transfer count is also maintained in order to allow termination of DMA transfers after a pre-programmed number of transfers. Each data transfer consumes 2 bus cycles (a minimum of 8 clock periods), one cycle to fetch data and the other to store data.

The two external DMA request inputs, DRQ0 and DRQ1, are connected to the NIB. DRQ0 is activated when data is to be transferred from the main processor to the network interface block (NIB) while DRQ1 is activated when data is to be transferred from the NIB to the main processor. The controller has the option of producing an internal interrupt when the transfer count reaches zero. This interrupt is used to inform the main processor that message transfer has been completed.

(vi) CHIP SELECT UNIT The integrated chip select unit provides programmable chip-select logic which can be used to select memory or peripherals (6 memory and 7 peripherals are provided) during processor controlled read or write operation. Note that these become inactive if the processor is forced into the "Hold" state.

The memory chip select are split into three groups for separately addressing the major memory areas in the system:

1 Upper memory (UCS*) - for reset EPROM
1 Lower memory (LCS*) - for lower RAM area
4 Mid-range memory (MCS0* - MCS3*) - for program and data memory
The size of each of these areas, and the starting location of the mid-range memory, are user programmable, with some restrictions.

Each of the peripheral chip select lines (PCS0* to PCS6*) address one of seven adjacent 128 byte blocks whose base address is programmable. This block can be programmed to be part of the memory or in a separate I/O block.

The chip select lines are connected to the following:

- UCS0*: EPROM
- LCS0*: RAM
- MCS0*: reserved for piggy back board
- MCS1*: reserved for piggy back board
- MCS2*: reserved for piggy back board
- MCS3*: reserved for optional memory chip socket (SK3)
- PCS0*: 2681 DUART
- PCS1*: NIB scratch pad memory
- PCS2*: Set INT0* flag to the NIB
- PCS3*: watchdog timer
- PCS4*: reserved
- PCS5*: reserved
- PCS6*: reserved

Each of the programmed chip select areas has a set of programmable ready bits associated with it. These ready bits control an integrated wait state generator which is programmable to provide 0 to 3 wait states for all accesses to the area of memory associated with a chip select signal.

(v) PROGRAMMABLE TIMERS The timer unit provides three independent 16-bit timers/counters. Two of these timers are available for use external to the CPU whilst the third timer is only available for internal use. All three timers operate independently of the CPU. In this design all the timers signals are left unconnected.
(b) NUMERIC PROCESSOR EXTENSION (8087) The 8087 is a numeric processor extension that provides arithmetic and logical instruction support for a variety of numeric data types. It executes numerous built-in functions such as tangent, log, exponential, square root and so on.

The 8087 can execute numeric instructions somewhere in the order of 100 times faster than a 80188 operating at the same speed [5].

As a coprocessor to the 80188, the 8087 is wired in parallel with the CPU as shown in Fig. 7.4. The CPU’s status (S0-S2) and queue status lines (QS0-QS1) enable the 8087 to monitor and decode instructions in synchronisation with the CPU and without any CPU overhead. All 8087 instructions appear as ESCAPE instructions to the 80188. Both the 80188 and 8087 decode and execute the ESC instruction together. The start of the numeric operation is accomplished when the CPU executes the ESC instruction. The 8087 can interrupt the CPU when it detects an error or exception, its interrupt being connected to INTO of the 80188.

(c) ADVANCED BUS CONTROLLER (82188) The Intel 82188 generates system command and control timing signals which are determined by the bus status lines signals (Fig. 7.4). It also provides HOLD/HLDA - RQ/GT bus protocol exchanger; this allows it to be used where bus control mechanisms between devices differ, such as between the 80188 and the 8087. In this design some of the control signals are buffered to increase the drive capability (Fig. 7.4).

(d) POWER-ON RESET The 80188 has a RES* input pin and a synchronised RESET output pin (Fig. 7.3). The RES* input is provided with a Schmitt-trigger to allow power-on reset via an R-C network, the corresponding RESET output lasting an
integer number of clock periods determined by the length of the RES* signal.

(e) ADDRESS/DATA BUS BUFFERS The 80188's has a time multiplexed address/data bus consisting of 8 lines (A/D0-A/D7) together with various control and status signals. The multiplexed lines are connected to latches (74LS573) which provide a demultiplexing function for the address bus signals (Fig. 7.5). These are controlled by the advanced bus controller (82188) which generates the demultiplexing signal.

The high address bus (A8-A19) is also buffered (74LS645); this, together with the address latches, ensures that the address bus has a high drive capability on all the 20 bit address bus.

7.3.2 MEMORY

Three 28 pin memory sockets are provided to host EPROM or static RAM (SRAM) devices (Fig. 7.6). The EPROM devices can be up to 32K Byte (SK1) while the SRAM can be up to 8K (SK2). SK3 can be configured for either an EPROM or a SRAM device.

On reset the 80188 begins execution at address FFFFOH, thus a jump instruction must be inserted at this location to transfer execution to the start of program. Consequently an EPROM chip must be mapped into the top of the memory. It contains the initialisation and main program software, its chip select pin being connected to UCS*.

The 80188 uses locations 0H-3FFH (1K Byte) for its interrupt vector table. This vector table allows different interrupt types to be serviced. The 80188 also
needs RAM for the storage of data variables, flags and stack. In this design an 8K SRAM is placed in location 0H to 1FFFH, its chip select pin being connected to LCS*.

The dynamic RAM store (DRAM) is located on a separate piggy back board as an option. It consists of sixteen 256K x 1bit DRAM I.C.s (1/2 mega-byte total), controlled by an Intel DRAM controller (8208), a set of data bus buffers (74LS245's) and associated control circuitry.

7.3.3 SERIAL COMMUNICATIONS

The Signetics 2681 DUART provides two independent full-duplex channels in a single chip (channel A and B). The DUART has a software programmable transmission format (number of data bits, stop bits, parity, etc.), programmable baud rate, error detection, multifunction counter/timer, 7-bit input port, 8-bit output port, interrupt system and on-chip oscillator.

The circuit diagram of the serial communication system is shown in Fig. 7.7.

To provide signals to meet the RS 232C specifications [6] a MC1488 line driver and a MC1489 receiver are used.

To ensure that the output slew rate of the line driver conforms to the RS232C specifications (30V/us) 390pF capacitor are connected across the outputs of the line drivers and 0V (analogue).

7.3.4 NIB INTERFACE

The aim of this interface (Fig. 7.8) is to accomplish data transfer between the
main processor and the network interface block by using the DMA controller in
the 80188.

The following table lists the interface signals with their functions;

<table>
<thead>
<tr>
<th>Signal</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCS2*</td>
<td>O</td>
<td>sets INTO flag to the NIB</td>
</tr>
<tr>
<td>DRQ0</td>
<td>I</td>
<td>channel 0 DMA request</td>
</tr>
<tr>
<td>DRQ1</td>
<td>I</td>
<td>channel 1 DMA request</td>
</tr>
<tr>
<td>PCS1*</td>
<td>O</td>
<td>chip select signal to the NIB</td>
</tr>
<tr>
<td>MWR*</td>
<td>O</td>
<td>data write enable</td>
</tr>
<tr>
<td>MRD*</td>
<td>O</td>
<td>data output enable</td>
</tr>
<tr>
<td>MDO-MD7</td>
<td>I/O</td>
<td>data signals</td>
</tr>
</tbody>
</table>

The NIB has the responsibility of requesting DMA transfers by activating one of
the DMA request signals (DRQ0 and DRQ1). Therefore there are two types of
transfer; transfer data from the main processor to the NIB, or transfer data
from the NIB to the main processor. The following is a description of each
transfer operation.

(a) Transfer data, NIB to main processor: If the NIB wants data to be
transferred from its memory to the main processor's memory it activates DRQ1.
The DMA controller then starts the transfer of data from the source (NIB
memory) to the destination (main processor memory). When data transfer has
been completed the main processor informs the NIB of its status by setting the
INT0* flag (74LS74) using PCS2* line. The NIB then clears this flag (INT0*) by
activating its INTA signal.
(b) Transfer data, main processor to NIB: The transfer operation starts when the main processor activates PCS2*, setting interrupt flag INTO* to get the attention of the NIB. The NIB responds to the INTO* signal by first clearing it and then transferring data as described above. In this instance, however, it uses the DRQ0 signal to activate data transfer in the opposite direction.

7.3.5 ANCILLARY CIRCUITS

(a) SINGLE STEP CONTROL The single step circuit (Fig. 7.9) is included to aid de-bugging and testing of both hardware and software. Using this a program can be executed one step at a time, making examination/testing of on-board devices more convenient. When switched in, on the push of a button the CPU asynchronous ready line (ARDY) is made active for one clock cycle only, thus only allowing the processor to complete one bus cycle. At all other times the ARDY line is held inactive, thus causing the processor to continuously insert wait states in the bus cycle.

The single step circuit is switched into the ARDY line by the toggle switch. Two NAND gates are used to debounce the push-button switch, so that when it is pressed and then released a single positive pulse is produced. When flip-flop 1 receives a positive going edge from the de-bounce circuit it's Q1 output is latched up high. This takes the D2 input of flip flop 2 high. On the next positive going edge of the CPU clock the Q2 output of 2 goes high, sending ARDY high, and the Q2* output of 2 goes low, clearing flip-flop 1. Since flip-flop 1 has now been cleared the D2 input to flip-flop 2 is now low. On the next positive going edge of the CPU clock the Q2 output of 2 (ARDY) goes back low. Thus the ARDY line has gone high for one CPU clock cycle. When the
push-button is released flip-flop I receives a negative going edge from the debounce circuit, but this has no effect.

(a) WATCHDOG TIMER The watchdog timer (Fig. 7.10) comprises a retriggerable monostable (74LS123) which is triggered by writing to a specific address. This is done repeatedly under program control so that, in normal circumstances, the monostable is always retriggered before its period expires. If the program crashes, the timer expires and so causes a Non-Maskable Interrupt (NMI).

In normal operation, when the monostable receives a negative going edge on its A1 input (from decoded address and PCS3*) the output Q* goes low. This assumes that OP5 from the 2681 is low, otherwise Q* remains high. It stays low for a time determined by the resistor / capacitor combination. provided the timer is selected (addressed) before the end of its time-out period Q* stays, the timer being re-triggered (normal operation). If the timer is not re-selected before the end of the time-out period Q* goes low, causing a NMI. This forces the processor to go through a pre-programmed interrupt service routine to recover from the fault condition. The values chosen for the timing components (R3 and C2) are selected to give a 1 second time-out period. The primary purpose of the control signal from the 2681 DUART (OP5) is to allow power-on initialisation to be completed without having to cope with an instantaneous NMI request. It also ensures that the watchdog timer doesn’t cause accidental interrupts when not in use.
REFERENCES FOR CHAPTER 7


4. SCN2681 Dual asynchronous receiver/transmitter (DUART), Mullard, Signetics 1985, No. 9397 093 66422.


FIG. 7.2 80188 BLOCK DIAGRAM.
FIG 7.3  80188 Configuration
FIG 7.4  82188 CONTROLLER AND 8087 NUMERICAL PROCESSOR
FIG. 7.5 ADDRESS/DATA BUFFERS AND LATCHES.
FIG. 7.6 MEMORY SYSTEM.
FIG. 7.9 SINGLE STEP CIRCUIT.
Chapter 8

PROTOCOL SOFTWARE DESIGN

8.1 OVERVIEW

The function of the software described here is to implement the Token Passing Protocol which has been described in chapter Four. The program designed to support interprocessor communications is written in ASM51 assembly language, software being developed on a Future FX20 personal computer (PC) using the AD X8051 cross-assembly package. The resulting ROMable code occupies approximately 2 kBytes of program store. Notice that the description given here is for the program from the station's point of view and not the ring as a whole.

The protocol program can be divided into three main sections (Fig. 8.1):

(a) Initialisation.
(b) Steady state.
(c) Ring maintenance.

The initialisation section starts after power-on, its aim being to construct the logical ring.
The steady state section starts after the initialisation process has ended. In this condition only 'token' and 'data' frames are received and transmitted.

The ring maintenance section takes care of addition and deletion of stations in the ring.

8.2 INITIALISATION

8.2.1 GENERAL

Construction of the logical ring is complete when each station in the ring has loaded the following registers (Fig. 8.2):

<table>
<thead>
<tr>
<th>REGISTER</th>
<th>DESCRIPTION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWN_ADD</td>
<td>The station's own address</td>
<td>TS</td>
</tr>
<tr>
<td>NEXT_ADD</td>
<td>The next station's address</td>
<td>NS</td>
</tr>
<tr>
<td>PREV_ADD</td>
<td>The previous station's address</td>
<td>PS</td>
</tr>
<tr>
<td>FIRST_ADD</td>
<td>The first station's address</td>
<td>FS</td>
</tr>
<tr>
<td>LAST_ADD</td>
<td>The last station's address</td>
<td>LS</td>
</tr>
<tr>
<td>MEMB_NO</td>
<td>The number of stations in the ring</td>
<td>CS</td>
</tr>
</tbody>
</table>

The above are called the ring construction registers. These registers must always be updated when any change in the ring takes place, this being the function of the maintenance section (section 8.4).

All stations start with disabling all flags used in the program then every station sets its own address register by doing a read cycle of the SSAR (follow the flowchart shown in Fig. 8.3). They then set their response timers (the time out
period being proportional to the station address) and set them running. Every station then goes to a subroutine which monitors message reception and timer overflow. If a message is received before timer time-out then this subroutine sets the (REC_MESS) flag. However if the opposite happens then the (TIME-OUT) flag is set. In the latter case this station must be the first station in the ring and responds by sending a 'claim token' frame to all other stations in the ring. In the former case the station tests the received message to recognise the type of control frame received. If the received message is 'claim token' then this station sets the FIRST_ADD to be the sender of the 'claim token' frame. However, if the control frame is 'solicit successor' then the station knows that it has been added to an already established ring. If the received control frame is not of the above two types then the station repeats the initialisation process.

The above discussion implies that there are three main routes for the initialisation process:

(a) First station route (Route1).

(b) Claim Token received route (Route2).

(c) Solicit successor received route (Route3).

Each route ends up with loading the ring construction registers with the appropriate values. However, each route takes a different sequence in loading these registers. Note that OWN_ADD register is set in all stations before they follow any route.
8.2.2 FIRST STATION ROUTE (ROUTE1)

This route is followed by the first station (i.e. that with the lowest address) in the ring. Here the station sets its ring construction registers according to the following sequence:

1. Own address.
2. First address.
3. Next address.
4. Previous address.
5. Last address.
6. Number of stations.

The overall procedures carried out during Routel activities are shown in Fig. 8.4a, b, c, d; setting of the above mentioned registers is clearly shown on these diagrams and explained below.

**Own address:** TS sets this register by reading the SSAR location in which the station's address is stored.

**First address:** Since TS is the 'claim token' sender then it loads this register with its own address.

**Next address:** In order to elicit the address of the next station in the sequence TS sends a 'who follows' frame and waits for a response (Fig. 8.4a). If a
message is received it is tested. If it is a 'set successor' frame TS reads the sender's address (included in the received frame) and sets the NEXT_ADD register. If no response is received after a preset wait time or if the received message is not 'set successor' the station assumes a failure condition; the initialisation process is then restarted. Once TS has identified its successor it sends the 'token' to NS and listens for a response. In normal circumstances it receives a 'token ack' frame.

From now on we will refer to the subroutine which sets the next address as the 'who follows subroutine' (WF_SUB).

Previous address: After passing the 'token' to NS, TS then waits for a 'who follows' frame. Every time it receives a 'who follows' it sets its response timer (here TS sets its response timer longer than the highest possible address) and wait for TIME-OUT or REC_MESS flag. If a 'set successor' frame is received the number of stations register (MEMB_NO) is incremented and the station waits for another 'who follows' frame. When the TS response timer times-out before any message is received it sends a 'set successor' frame. Now TS considers the station to whom it has sent the 'set successor' as its previous address and sets the PREV_ADD register.

Last address: Since TS is the first then the number now in the PREV_ADD register must be the address of the last station in the ring. Therefore the LAST_ADD register is now loaded with a value equal to that of the PREV_ADD register.

Number of stations: The first station has the responsibility of counting the number of stations in the ring. This is accomplished by counting the number
of 'set successor' frames sent generated in response to 'who follows' frames, until TS knows PS then the value in MEMB_NO register represents the number of stations in the ring.

TS then sends the following frames to all the stations; 'set last', 'memb count', 'init done', and finally it sends the 'token' to NS. At this stage all ring construction registers in TS are set and the station now enters the steady state condition.

8.2.3 CLAIM TOKEN RECEIVED ROUTE(ROUTE2)

This route is taken by all stations other than the first in the ring. The construction registers are set in the following sequence:

1. Own address.
2. First address.
3. Previous address.
4. Next address.
5. Last address.
6. Number of stations.

The overall procedures carried out during Routel activities are shown in Fig. 8.5a,b,c,d; setting of the above mentioned registers is clearly shown on these diagrams and explained below.

Own address: Same as Routel.
First address: TS sets the FIRST_ADD register from the information contained in
the 'claim token' frame. Note that this originates from the first station to come
on line.

Previous address: Same as Routel (but without counting the number of
stations).

Next address: NEXT_ADD register is set by using the WF_SUB as described in
the Routel sequence.

Last address: The LAST_ADD register is set with the value extracted from the
'set last' control frame issued by the first station at the end of initialisation.

Number of stations: The MEMB_NO register is set with the value extracted from
the 'memb count' control frame issued by the first station at the end of
initialisation.

All stations following this route go into the steady state on receipt of the 'init
done' frame from the first station.

8.2.4 SOLICIT SUCCESSOR RECEIVED ROUTE(ROUTE3)

In this route the TS on receipt of 'solicit successor' checks to see if it lies in
the station range specified by the control frame. If it doesn't lie then it goes
back to repeats the initialisation process. If it does it then sets its ring
construction registers according to the following sequence:

1. Own address.

2. Previous address.
3. Next address.

4. First address.

5. Last address.

6. Number of stations.

The overall procedures carried out during Routel activities are shown in Fig. 8.6a,b,c,d; setting of the above mentioned registers is clearly shown on these diagrams and explained below.

Own address: Same as Routel.

Previous address: The PREV_ADD register is set with the value extracted from the 'solicit successor' control frame issued by the station which has sent the invitation to TS.

Next address: The NEXT_ADD register is set with the value extracted from the 'solicit successor' control frame issued by the station which has sent the invitation to TS.

First address: TS tests the address of the previous and next station. If NS is greater than PS, then TS sends a 'who first' frame and waits for a 'set first' frame. If, however, NS is less than PS, then TS assumes that its location is between the last and first station. TS is then either the new last or first address. In this case it tests its own and first address. If its address is less than the first then it considers itself as the new first address. If it is greater then it considers itself as the new last address.
last address: TS tests the address of the previous and next station. If NS is
greater than PS, then TS sends a 'who last' frame and waits for a 'set last'
frame. If, however, NS is less than PS, then TS assumes that its location is
between the last and first station. In this case it tests its own and last
address. If its address is greater than the last then it considers itself as the
new last address.

Number of stations: The number of stations is set by sending a 'memb req'
frame and waiting for a 'memb count' frame which includes the number of
stations in the ring.

At this moment the station goes to the steady state operation.

8.3 STEADY STATE OPERATION

8.3.1 GENERAL

When a station is in the steady state operation it monitors three events (Fig.
8.7a):

(a) Message reception (REC_MESS),
(b) Interrupt from the main processor (MAIN_REQ),
(c) Token rotation timer time-out (TIME-OUT).

In normal operation the received message is either 'token' or 'data' frame and
the token rotation timer does not time-out before a 'token' is received. If
however event (c) occurred or the received message is not 'token' or 'data' then a fault condition is assumed. Dealing with such faults is left to the ring maintenance section.

8.3.2 'TOKEN' OR 'DATA' FRAME RECEPTION

The received message is either 'token', 'data' or any other control frame as shown in Fig. 8.7a,b,c. In this section only 'token' and 'data' frames are mentioned, the other control frames are left to the section which deals with ring maintenance.

If the received message is a 'token' frame then TS responds by sending a 'token ack' frame to its predecessor. TS then goes to a subroutine called ACCESS (Fig. 8.8). This subroutine starts by testing the 'token counter', if the value stored in the 'token counter is 10 then the station sends a 'solicit successor' frame to invite a new station to enter the ring. The station waits for a preset time for a 'set successor' response. On receipt of this, new NS is set and then TS goes to steady state. If, however, the timer times-out TS assumes that there is no pending request for entry to the ring and so passes the 'token' to NS in the usual way and goes back to the steady state condition.

For a 'token counter' value less than 10 the 'message flag' is tested; if set then there is a message in the SPM ready for transmission. This message is then transmitted to the destination address and the 'message flag' cleared. When message transmission is complete or when the transmission area of the SPM is empty the station then passes the 'token' to its successor in the usual way.
If the received message is a 'data' frame then the station activates the DMA1 request to the main processor to transfer the received data to the main processor's local memory.

8.3.3 INTERRUPT FROM MAIN PROCESSOR

When the station detects an interrupt from the main processor it activates the DMA0 request line to the main processor. This causes data to be transferred from the local memory of the main processor to the transmission area in the SPM. The 'message flag' is then set to indicate that the SPM has a message ready for transmission.

8.4 RING MAINTENANCE

8.4.1 GENERAL

After having entered the steady state condition ring maintenance is required to cope with fault conditions in the following circumstances;

(a) The 'token rotation timer' times-out: This fault occur when the 'token' is lost, the loss being caused by failure of the station currently holding the 'token'.

(b) A 'token' is sent and no 'token ack' frame is received due to failure of the next station in sequence.

(c) A control frame is received which isn't either a 'token' or 'data' frame.
Note that in case (a) and (b) the fault is detected by this station (TS) and therefore TS handles the fault recovery process. In case (c) however the fault condition has occurred elsewhere in the ring and has been detected and recovered from another station. Hence the control frames received by TS are part of the recovery process; in this case the function of TS is to respond to these frames.

8.4.2 TOKEN LOST

The token rotation timer is calculated by multiplying a constant, which represents the token hold time, by the number of stations in the ring. Should the 'token rotation timer' time-out the station assumes that the 'token' has been lost and consequently it reconfigures the ring. TS therefore assumes that it is holding the 'token' and informs all stations in the ring by sending a 'claim token' frame. It then goes to the ACCESS subroutine as if it had just received the 'token'.

8.4.3 NEXT STATION FAILURE

This fault is detected when NS fails to respond to a 'token' frame sent by TS. In this case TS assumes that its successor has failed. It therefore attempts to patch into the next station in the ring sequence through use of the WF_SUB subroutine.

8.4.4 RING MAINTENANCE FRAME RECEIPTION

In this case the fault recovery is carried out by another station. The response
of TS depends on the type of control frame received (Fig. 8.7b,c). The type of control frame and its response are explained in the following;

'who follows': When TS receives this control frame it goes to a subroutine called 'who follows response' (Fig. 8.9). This subroutine checks the address of the failed station, which is included in the received frame. If the failed station address is that of the previous station (PS) then TS responds by sending a 'set successor' frame and sets the sender of the 'who follows' frame as its new PS. If the failed station address is not that of the previous station then TS sets its response timer. If the timer time-out it sends a 'set successor' frame. If a message is received before timing-out TS goes back to steady state.

'solicit successor': When this frame is received the station goes to a subroutine called 'solicit successor response' (Fig. 8.10) where the 'solicit successor timer' is set. The station then waits for a response or time-out flag. In both cases the station goes back to steady state.

'del memb': When this frame is received the 'memb no' register is decremented.

'new memb': When this frame is received the 'memb no' register is incremented.

'who first': When this frame is received the station responds by sending a 'set first' frame.

'who last': When this frame is received the station responds by sending a 'set last' frame.

'memb req': When this frame is received the station responds by sending a 'memb count' frame.
'set last': When received the station sets the last address register according to the address value included in the received frame.

'first': When received the station sets the first address register according to the address value included in the received frame.

'claim token': When received the station goes back to steady state.

'set successor2': When received the station sets the next address register according to the address value included in the received frame. This is different from the 'set successor' frame used to respond to a 'who follows' or 'solicit successor' frames. It is used by any station wishing to drop out of the ring to patch its PS with NS stations.

'set previous': When received the station sets the previous address register according to the address value included in the received frame.

After responding to these control frames TS goes back to steady state operation. If the received message is not one of the above frames then the station goes back to repeat the initialisation process.
FIG. 8.1 PROTOCOL MAIN SECTIONS.
FIG. 8.2 RING CONSTRUCTION REGISTERS.
FIG. 8.3 RECOGNISING INITIALISATION ROOT.
ROUTE 1

2
SET FIRST ADDRESS

INC MEMB NO

SEND CLAIM TOKEN

SEND WHO FOLLOWS

SET WHO FOL TIMER

RUN TIMER

START

TIME-OUT

TIME-OUT OR RECEIVE MESSAGE

REC_MESS

IS IT SUCCESSOR

YES

3
SET NEXT ADDRESS

WF_SUB

FIG. 8.4a ROUTE 1 (FIRST STATION ROUTE).

8.18
A

SEND TOKEN

START

SET TOKEN ACK TIMER

RUN TIMER

TIME-OUT

TIME-OUT OR RECEIVE MESSAGE

REC_MESS

NO

IS IT TOKEN ACK

YES

SET WAIT TIMER

C

TIME-OUT

TIME-OUT OR RECEIVE MESSAGE

REC_MESS

NO

IS IT WHO follows

YES

B

FIG. 8.4b ROUTE 1.
FIG. 8.4c ROUTE 1.
FIG. 8.4d ROUTE 1.
FIG. 8.5a ROUTE 2.
FIG. 8.5b ROUTE 2.
FIG. 8.5c ROUTE 2.
FIG. 8.5d ROUTE 2.
ROUTE 3

START

TEST LOCATION

IS IT WITHIN INVITATION AREA

YES

SET RESPONSE TIMER

RUN TIMER

TIME OUT OR RECEIVE MESSAGE

TIME OUT

SEND SET SUCCESSOR

2

SET PREVIOUS ADDRESS

3

SET NEXT ADDRESS

SET WAIT TIMER

RUN TIMER

A

FIG. 8.6a ROUTE 3.
FIG 8.6b ROUTE 3.
FIG. 8.6c ROUTE 3.
FIG. 8.6d ROUTE 3.
FIG. 3.7b STEADY STATE OPERATION.
C

IS IT MEMB_REQ

YES

SEND MEMB_COUNT

NO

IS IT SET LAST

YES

SET LAST ADDRESS

NO

IS IT SET FIRST

YES

SET FIRST ADDRESS

NO

IS IT CLAIM TOKEN

YES

SET SUCCESSOR2

NO

IS IT SET SUCCESSOR2

YES

SET NEXT ADDRESS

NO

IS IT SET PREVIOUS

YES

SET PREVIOUS ADDRESS

START

GO TO STEADY STATE

FIG. 8.3c STEADY STATE OPERATION.
FIG. 8.8 ACCESS ROUTINE.
FIG. 8.9 WHO Follows RESPONSE ROUTINE.
FIG. 8.10 SOLICIT SUCCESSOR RESPONSE.
Chapter 9

SYSTEM PERFORMANCE EVALUATION

9.1 OVERVIEW

The aim of this chapter is to evaluate and simulate the theoretical performance of the multiprocessor system described in this thesis. Overall system behaviour depends on a number of parameters, the major ones being:

* Number of stations (N).
* Message length (ML).
* System bus transmission rate (VB).
* DMA transfer rate (VM).
* Bus access control.

By understanding the characteristics and behavior of the system its suitability application to real-time problems can be more easily evaluated. Moreover, such a study identifies the importance of the parameters under consideration; using this information system behaviour can be modified to meet specific performance requirements.

Performance can be evaluated in two ways [1]:
Modelling: By generating a mathematical model of the system it is possible, by means of analytic methods or simulation tools, to evaluate the effects of individual parameters on total system performance. These considerations are purely theoretical, requiring no hardware, and giving results even during the design phase. Their accuracy depends on how well the model reflects the actual system.

Measurement: Measuring produces exact results with regard to special cases, but cannot be done until the hardware is fully operational. In general, additional software (and sometimes hardware) is needed in order to be able to carry out these measurements. This leads to a complex problem, that of conserving the unmeasured system state. Note that when a measuring program is inserted the original program should not change its mode of operation, otherwise the results are invalid. Further, the measuring program should not affect system timing and throughput.

In view of these difficulties the mathematical modelling approach has been used for system performance evaluation. It also has the advantages of simplicity for this type of system, as all mathematical computations are deterministic, not statistical measures. The other reason is that there are some parameters which one does not have the ability to vary (such as the number of stations which is limited by the available three stations).

9.2 EFFECT OF BUS ACCESS METHOD ON DISTRIBUTED COMPUTERS

Decentralized or distributed access methods for a shared medium are more
reliable as they do not have a single-point failure [2]. Two major classes of distributed access methods are in use today, contention schemes and non-contention schemes [3].

Contention schemes are simply those which allow collision in the shared medium (two or more stations may transmit at the same time). When collision is detected by the transmitting stations they wait for a random back-off period before retransmitting; this minimise the likelihood of repeated bus collisions. Hence the performance characteristics of the contention schemes are nondeterministic. Access to the bus cannot be predicted, i.e. it is a statistical process. For this reason they are not suitable for use in fast, critical real-time applications [2].

Non-contention schemes eliminate bus collisions by preallocating the bus or by passing the right to use the bus (the 'token') among stations. A common characteristic of all non-contention schemes is that they are deterministic; hence they are very suitable for real-time applications [4].

9.3 PERFORMANCE MEASURES

There are two type of measures that affect the performance of a distributed computer system [5]: system throughput and response time. System throughput is defined as the total number of data bits (bytes here) received at the destination per second [6]. Response time is the delay experienced by a station in acquiring the bus to transmit a message [6]. Note that these definitions are applied to performance evaluation of LAN systems.
In a system that uses token passing techniques to control access to the bus the response time is equal to the token rotation time (TRT).

9.4 MATHEMATICAL MODEL

The mathematical model which represents system throughput and response time of the system is derived here. The parameters that influence these performance measures are the following:

* Number of active stations in the system (N).
* Maximum message length (ML).
* System bus transmission rate (VB).
* DMA transfer rate (VM).

Fig. 9.1 illustrates how the general formula for the TRT is derived:

\[
TRT = N(TS) + (N-1)THT \quad (1)
\]

**TS**

TS is the switch time, defined as being the time from which a station sends the 'token' until 'token ack' is received.

**THT**

THT is the token hold time which is defined as the longest time a station can hold the token for. Here it is assumed that a station can hold the 'token' for a time which is enough for transmitting one message only.
(a) Calculation of TS:

\[ TS = 2 \times \left( \frac{\text{control frame length}}{\text{VB}} \right) + \frac{\text{ML}}{\text{VM}} + \text{set-up time} \quad (2) \]

\( [2(\text{control frame length}/\text{VB})] \) is the transmission time of the 'token' and 'token ack' control frames.

(ML/VM) is a variable response delay within the receiving station; this is caused by its main processor accessing the scratch pad memory and so locking out bus communication.

The set-up time is the time taken by the communication processor (Intel 8031) to set messages and prepare them for transmission.

(b) Calculation of THT:

\[ \text{THT} = \frac{\text{ML}}{\text{VB}} + \frac{\text{ML}}{\text{VM}} + \text{set-up time} \quad (3) \]

(ML/VB) is the time needed to transmit one message of length ML.

(ML/VM) As above.

The set-up time As above. (c) Calculation of throughput: The maximum throughput is equal to;

\[ \text{TRP} = \frac{\text{ML}}{\text{THT}+\text{TS}} \quad (4) \]

9.5 RESULTS
9.5.1 ASSUMPTIONS

The following model assumptions are used in these computations:

* System bus transmission rate ($V_B$) = 2 M Byte/sec.

* DMA transfer rate ($V_M$) = 0.5 M Byte/sec.

* Set-up time = 20 usec.

* The token hold time is long enough to allow for transmission of one message frame of the selected message length.

* Every time a station receives the 'token' it transmits a message of maximum length.

* Every time a station wants to transmit a message it is delayed by the destination station being busy exchanging data with its main processor. The maximum length of this delay is equal the time needed to make a DMA transfer of a maximum message length.

* The system is always in steady state operation (no ring maintenance).

The above assumptions are derived from the actual system. They account for worst case conditions which makes it possible to predict the behavior of the system in such circumstances. This approach is called "worst case system performance" [7].
9.5.2 EFFECT OF NUMBER OF STATIONS ON RESPONSE TIME

The effect on response time (TRT) produced by varying the number of stations is plotted in Fig. 9.2; note that a third parameter is included, that of varying the message length. It is observed that an increase in the number of stations causes a corresponding proportional increase in TRT.

9.5.3 EFFECT OF MESSAGE LENGTH ON RESPONSE TIME

The effect on response time (TRT) produced by varying the message length is plotted in Fig. 9.3; note that a third parameter is included, that of varying the number of stations. It is observed that an increase in the message length causes a corresponding proportional increase in TRT.

9.5.4 EFFECT OF MESSAGE LENGTH ON THROUGHPUT

The effect of system throughput (TRP) produced by varying the message length is plotted in Fig. 9.4. It can be seen that as message length is increased the system throughput initially increases in direct proportion to ML. However the incremental gain gradually reduces until a point is reached where an increase in message length has little effect on the throughput. The reason for this is that, as messages get longer, the control frames overhead becomes relatively less significant. Throughput then becomes approximately constant, because it is bounded only by the bus transmission rate.
9.6 ENHANCED SYSTEM

Here an enhanced system is proposed to improve performance. Three possible hardware modifications that could significantly affect the system performance are:

(a) Increasing DMA transfer rate.
(b) Increasing system bus transmission rate.
(c) Using transparent dual port RAM.
(d) Increasing the 8031 CPU clock speed.

The DMA transfer rate can be increased by using a 16 MHZ clock for the 80188 CPU. This increases the DMA transfer rate to 1 M Byte/s.

In the current design a slight modification to the hardware can increase the bus transmission rate to 4 M Byte/s; this modification should therefore be implemented.

Another hardware modification which improves system performance is the replacement of the current dual port RAM (TMS9650) by one which allows simultaneous access from the two ports. This will reduce the delay experienced when a station is transmitting a message to a station which is busy exchanging information with its main processor.

The 8031 CPU speed can be increased to 12 MHz. This modification reduces the set-up time, needed to prepare a message for transmission, to 10 usec.
The effect of each modification on TRT and TRP is plotted in Figs. 9.5, 9.6 respectively. It is noticed that using a transparent dual port RAM has the greatest influence on system performance then comes the DMA transfer rate, 8031 CPU speed and finally system bus transmission rate.

When all the above modifications are implemented the equations which represent response time and system throughput remain unchanged, i.e.

\[
\text{TRT} = N(TS) + (N-1)THT \quad (1)
\]

\[
\text{TRP} = \frac{ML}{(THT+TS)} \quad (4)
\]

However both TS and THT are modified, as follows;

\[
THT = \frac{ML}{VB} + \text{set-up time} \quad (5)
\]

\[
TS = 2(\text{control frame length}/VB) + \text{set-up time} \quad (6)
\]

The results of the computations are then plotted in Figs. 9.7, 9.8, 9.9. Table 9.1 is constructed just to see the amount of improvement which can be gained by doing these hardware modifications.

Table 9.1 Performance comparison between current and an enhanced system.

<table>
<thead>
<tr>
<th></th>
<th>Current system</th>
<th>Enhanced system</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAR</td>
<td>ML=8, N=5</td>
<td>ML=128, N=50</td>
</tr>
<tr>
<td>TRT</td>
<td>375 us</td>
<td>30.8 ms</td>
</tr>
<tr>
<td>TRP</td>
<td>96 Kbyte/s</td>
<td>200 Kbyte/s</td>
</tr>
<tr>
<td></td>
<td>115.5 us</td>
<td>2.73 ms</td>
</tr>
<tr>
<td></td>
<td>313 Kbyte/s</td>
<td>2.3 Mbyte/s</td>
</tr>
</tbody>
</table>
PAR = Parameter

By careful study of the performance measures one can see that a decision has to be made on selecting the proper values for various parameters (i.e. number of stations, message length, bus transmission rate and DMA transfer rate). These selected values must then influence the performance of the system and make it suitable for the required application. The suitability of the system for a certain application requirements then can be judged in an early stage before implementation.
REFERENCES FOR CHAPTER 9


6. Chien, J. 'Performance analysis of the 802.4 token bus media access control protocol' Workshop on Analytic and Simulation Modeling of IEEE 802.4 Token Bus Local Area networks held at Gaithersburg, Maryland, USA, April 1985 (Final report), pp113-152.

**SECTION 9.1**

**ABBRIVIATIONS**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRT</td>
<td>Token Rotation Time</td>
</tr>
<tr>
<td>THT</td>
<td>Token Hold Time</td>
</tr>
<tr>
<td>TS</td>
<td>Switching Time</td>
</tr>
<tr>
<td>ML</td>
<td>Message Length</td>
</tr>
<tr>
<td>ML/VB</td>
<td>Message Length/Bus Transmission Rate</td>
</tr>
<tr>
<td>ML/VM</td>
<td>Message Length/DMA Transfer Rate</td>
</tr>
</tbody>
</table>

**FIG. 9.1** MATHEMATICAL MODEL DERIVATION.

TRT = \( (N)TS + (N - 1)THT \)

THT = \( (ML/VB) + (ML/VM) + \text{SET-UP TIME} \)

TS = \( 2(\text{CONTROL FRAME LENGTH/VB}) + (ML/VM) + \text{SET-UP TIME} \)
FIG. 9.3 PERFORMANCE EVALUATION - EFFECT OF MESSAGE LENGTH ON TRT.
Fig. 9.5 Performance Evaluation - Effect of Each Hardware Modification on TRT.
FIG. 9.6 PERFORMANCE EVALUATION – EFFECT OF EACH HARDWARE MODIFICATION ON TRP.
FIG. 9.7 PERFORMANCE EVALUATION - EFFECT OF NUMBER OF STATIONS ON TRT (ENHANCED SYSTEM).
FIG. 9.9 PERFORMANCE EVALUATION - EFFECT OF MESSAGE LENGTH ON TRP (ENHANCED SYSTEM).
Chapter 10

CONCLUSION

10.1 OVERVIEW

During this research programme a multiprocessor system suitable for real-time applications was designed, three stations being built to prove the concept. Each station consists of two processors, an Intel 8031 for handling communications and an Intel 80188 for executing application programs. A token passing bus access protocol program has been developed, software being generated in ASM51 assembly language. The program size is 2 Kbytes (approximately) and is run from EPROM on the processor board. A demonstration version of the protocol program has also been developed, its function being to display all the messages received and transmitted by a station on a VDU. This program was an essential aid in development and debugging work, in monitoring station behaviour in normal circumstances, and in observing ring reconfiguration activities due to station failures or addition/deletion of stations.

Simple programs for the 80188 have been designed for system testing, their purpose being to perform message exchange with the Network Interface Block.
An analytical model has been derived for the system in order to study its performance and suitability for real-time applications. This study established the relative importance of system parameters on overall performance; furthermore it identified clearly the way in which significant improvements could be made in system performance within the constraints of the existing design.

The designed system as designed meets the requirements stated in Chapter 2 for distributed real-time systems, having the following features;

* High response time.
* High throughput.
* High reliability.
* High flexibility.
* Bounded message transfer time.

The above features can be improved by adopting the enhancements mentioned in Chapter 9. To increase the reliability the bus may be duplicated to eliminate a single point failure. By using a simple structure such as the one devised here the backplane bus may be replicated at a relatively low cost for use with standard Eurocard size backplane. Additionally to that the serial I/O communications supported by the on-chip UART of the 8031 can be used to provide a back-up communication link between stations in the event of bus faults. Note also that the 8031 has an in-built facility within its serial communications hardware supporting multiprocessor communications [1].

10.2 SOFTWARE DEVELOPMENT FOR DISTRIBUTED SYSTEMS

The usefulness of any multiprocessor system is determined by the ease of
software development for that system. The design and synthesis of software for distributed systems requires the use of a design methodology and programming language which builds on the inherent parallel nature of such systems [2]. Formal methods applied to the design of software for distributed processors lead to the identification of processes, capable of asynchronous execution, interacting with other processes by communications.

Unfortunately, hardware developments to-date have far exceeded software capabilities [3]. There is not yet available standard software tools such as language compilers specifically designed for multiprocessor environment [4]. A survey carried out [5] to evaluate the suitability of modern languages for real-time distributed systems. These languages are: Modula, Modula2, Ada, Edison, Concurrent Pascal, Pearl, and DP. One of the observations of the survey is that most languages intended for real-time systems are not suitable for distributed applications (i.e. the language implementation requires shared memory). Recently INMOS has released the transputer (IMS T424) with a language called Occam. Occam has been designed specifically to operate on the principles of concurrency and parallelism [6].

Intense research activity in recent years is now yielding a more mature understanding of the problems of a distributed environment [3]. One promising approach to the solution of this problem is the use of expert systems, which accept uniprocessor programs and convert them into program partitions suitable for concurrent execution. These are optimised to produce a minimum of interprocessor communication and balanced utilization of each processor.
10.3 EVALUATING DIFFERENT ACCESS TECHNIQUES

It is interesting to realise that different non-contention access techniques can be implemented on the same hardware. Some examples of these techniques are; Vector-Driven Proportional Access (VDPA) [7] and the Implicit Token Passing (ITP) [8].

The VDPA bus allocation scheme defines a set of bus "slots". A slot is an opportunity to transmit a message. Slots are assigned to nodes in a predetermined sequence. As each successive slot is encountered, the node assigned to that slot either sends exactly one message, or it immediately advances to the next slot by transmitting the end-of-transaction (EOT) signal. This approach allows the system configurator to determine the distribution of bus slots among nodes, as well as the maximum delay between successive bus slots for each node.

Fig. 10.1 shows the implementation of the VDPA scheme. Each node maintains a binary allocation vector (or control schedule memory) with one position for each slot. The allocation mechanism is initialised so that all nodes access the same bit position (or address) in their respective allocation vector. Each time the bus is to be reallocated, the node currently having bus access transmits an EOT on the bus, and each node advances to the next position in its allocation vector. As indicated in Fig. 10.1, the vectors are configured so that exactly one node has a "1" in any given allocation vector bit position; this allocates the bus for the corresponding slot to the node with the "1".
In the Implicit Token Passing (ITP) protocol, nodes are assigned a predetermined sequence number. This defines the order in which nodes are permitted to transmit data. Thus in normal operation, nodes transmit individual information packet in sequence. A packet consists of a header and an optional data section. The header contains the sending node’s address (sequence number), the destination address, message word count and a CRC checksum. The data section consists of message data and CRC checksum. If no data is to be sent, the destination and word count fields are set to the source address and zero respectively. By putting the source address in the header, nodes remain aware of message slot boundaries and can determine who is transmitting without using a central timekeeper. Also, nodes are guaranteed a successful bus access within a (relatively short) finite time. This is because the longest amount of time a node must wait between successive transmissions is the time it takes to send one header and one data packet for each other node on the network. A node’s relative priority or frequency of transmission can be adjusted by assigning more than one sequence number to certain nodes.

The ITP protocol was implemented on NASA Langley Research Center’s Data Distribution Evaluation System [9]; a comparison of the implicit and explicit token passing protocols is given in the same reference [9].

10.4 COMMENTS AND CONCLUSIONS

The following points are based on the experience gained in designing the multiprocessor system.
* Multiprocessor systems are ideal structures for use in flight simulators, robotics, process control, i.e. for fast real-time applications.

* Software development for distributed systems is still an open area of research.

* The token passing concept is clearly well suited for use where system reliability is important.

* Adding and deleting stations is simple and easily accomplished. This allows the designer to specify certain stations to enter the ring only when needed only (i.e. at peak times, emergency conditions, etc.). When not needed they can withdraw from the ring.

* Significant flexibility is achieved by allowing the designer to specify the main processor structure and its components. This enables the engineer to accommodate special tasks (i.e. mathematical computations, graphics generation, I/O signals handling) through specific design solutions.

* Some stations may be listeners only (the 'token' is not passed to them). This feature might be useful for display systems.

* The system can be used for geographically distributed processing; this is facilitated through the use of its in-built serial communications feature.

* The performance of the system can be notably improved by implementing the modifications mentioned in chapter 9.

* The development of the protocol program is time consuming since every time a modification is done three EPROMs has to be blown. It would be useful to
provide a method of downloading program directly into the target system (i.e., EPROM emulation).

* Testing message transmission between stations is a difficult task since there were six processors running at the same time.

* There are a lot of new dual port memories coming into the market if they were present when this project started, the hardware design might be different. However, since the hardware is implemented in functional blocks, it will be easy to modify the hardware to cope with the rapid progress of VLSI chips in the industry.
REFERENCES FOR CHAPTER 10

1. Katausky, J. 'Interprocessor communications with single-chip microcomputers' MIDCON/82 Conference Record, USA, pp1-3.


FIG 10.1 THE VDPA BUS ALLOCATION SCHEME
Appendix A

CONTROL AND DATA FRAMES

All frames sent or received by the NIB are described here. The general frame format is:

```
--------------------------------------
| SD ; DML ; FT ; DA ; SA ; DATA ; ED ; |
--------------------------------------
```

Where:

- **SD**: Start delimiter
- **DML**: Data message length
- **FT**: Frame type
- **DA**: Destination address
- **SA**: Source address
- **DATA**: Data field
- **ED**: End delimiter

CLAIM TOKEN:

**When transmitted:**

(a) In the initialisation process: This frame is sent by TS if it has the lowest address in the ring.

(b) In the ring maintenance process: This frame is sent by TS when it discovers that the 'token' is lost.
**Action on receipt:**

(a) In the initialisation process: When TS receives this frame it stops its response timer and wait for a 'who follows' control frame.

(b) In the ring maintenance process: TS re-initialise its token rotation timers.

**TOKEN:**

**When transmitted:**

(a) In the initialisation process: When TS is holding the 'token' and has identified its successor it passes the 'token' to it.

(b) In the ring maintenance: Not used.

(C) In steady state operation: When TS is is holding the 'token' and has finished transmitting a 'data' frame, it passes the 'token' to its successor. Note that if no data is to be transmitted the 'token' is passed straight on to the successor.

**Action on receipt:**

(a) In the initialisation process: TS sends a 'token ack' frame to the sender of the 'token' frame (PS).

(b) In the ring maintenance process: Not used.

**TOKEN ACK:**

**When transmitted:**
(a) In the initialisation process: When TS receives a 'token' frame.

(b) In the ring maintenance process: Not used.

(c) In steady state operation: Same as (a).

**Action on receipt:**

(a) In the initialisation process: TS assumes that its successor has received the 'token'.

(b) In the ring maintenance process: Not used.

(c) In steady state operation: Same as (a).

**WHO FOLLOWS:**

**When transmitted:**

(a) In the initialisation process: When TS is holding the 'token' it sends this control frame in order to identify its successor.

(b) In the ring maintenance process: When the NS fails to send a 'token ack' frame response.

**Action on receipt:**

(a) In the initialisation process: TS starts its response timer; should this time-out before any other station in the system it transmits a 'set successor' frame.

(b) In the ring maintenance process: When TS receives a 'who follows' frame it
compares its PS with the address of the failed station (included in the received frame). If the failed station is PS a 'set successor' frame is transmitted; otherwise TS behaves as described in (a).

SOLICIT SUCCESSOR

When transmitted:

(a) In the initialisation process: Not used.

(b) In the ring maintenance: After TS handles the 'token' for a preset number of times (S) a 'solicit successor' frame is transmitted. The purpose of this is to invite new stations to enter the ring. S is programmable.

Action on receipt:

(a) In the initialisation process: If TS is a new station waiting for an invitation to enter the ring the reception of this frame means the start of the initialisation process to enter the ring. This process starts by checking if its own address lies in the invited region. If yes it starts its response timer if not it waits for another invitation. If the response timer time-out it sends a 'set successor frame'.

(b) In the ring maintenance process: Existing stations do not respond on receipt of this control frame since they are already in the ring.

SET SUCCESSOR 1

When transmitted:

(a) In the initialisation process: When the response timer of TS time-out; being
started as a response to a 'who follows' frame or a 'solicit successor' if TS is invited to enter an already established ring.

(b) In the ring maintenance process: Same as (a) for the reception of a 'who follows' frame only.

Action on receipt:

(a) In the initialisation process: If TS is responsible for eliciting this response then it sets its NS to be the address of the sender of the 'set successor 1' frame. If TS is not responsible it doesn’t do anything.

(b) In the ring maintenance process: same as (a).

SET SUCCESSOR 2:

When transmitted:

(a) In the initialisation process: Not used.

(b) In the ring maintenance process: When TS wishes to drop out from the ring it sends a 'set successor 2' to its predecessor (PS).

Action on receipt:

(a) In the initialisation process: Not used.

(b) In the ring maintenance process: When TS receives this frame it sets its NS value to be the NS of the station which is exiting from the ring.

SET PREVIOUS
When transmitted:

(a) In the initialisation process: Not used.

(b) In the ring maintenance process: When TS wishes to drop out from the ring it sends a 'set previous' frame to its successor (NS).

Action on receipt:

(a) In the initialisation process: Not used.

(b) In the ring maintenance process: TS resets its PS value to be the PS of the station exiting from the ring.

NEW MEMBER:

When transmitted:

(a) In the initialisation process: If TS has just been granted entrance to the ring it sends this frame to all stations.

(b) In the ring maintenance process: Not used.

Action on receipt:

(a) In the initialisation process: Not used.

(b) In the ring maintenance process: TS increments its count record of the number of stations in the ring.

DELETE MEMBER:

When transmitted:
(a) In the initialisation process: Not used.

(b) In the ring maintenance process: When NS fails to respond to a 'token' TS sends a 'delete member' frame to all stations in the ring.

Action on receipt:

(a) In the initialisation process: Not used.

(b) In the ring maintenance process: TS decrements its count record of the number of stations in the ring.

MEMBER REQUEST

When transmitted:

(a) In the initialisation process: If TS has just entered an existing (established) ring it sends a 'member request' frame to identify the number of stations in the ring.

(b) In the ring maintenance process: Not used.

Action on receipt:

(a) In the initialisation process: Not used.

(b) In the ring maintenance process: When TS receives this frame it responds by transmitting a 'member count' frame to the sender.

MEMBER COUNT

When transmitted:
(a) In the initialisation process: If TS is the first station it transmits the 'member count' frame after the ring is completed (formed). This has the function of defining the number of stations in the ring.

(b) In the ring maintenance process: When TS receives the 'member request' frame it responds by transmitting a 'member count' frame to the requester, thus defining the number of stations in the ring.

**Action on receipt:**

(a) In the initialisation process: If TS is a new station, on joining an established ring, it transmits a 'member request' frame and receives a 'member count' frame. It uses the frame information to set up its number of stations record.

(b) In the ring maintenance process: Not used.

**WHO LAST**

**When transmitted:**

(a) In the initialisation process: If TS is a new station, on joining an established ring, it transmits a 'who last' frame.

(b) In the ring maintenance process: Not used.

**Action on receipt:**

(a) In the initialisation process: Not used.

(b) In the ring maintenance process: When TS receives a 'who last' frame it responds by sending 'set last'.
SET LAST

When transmitted:

(a) In the initialisation process: If TS is a new station, on joining an established ring, and its address is greater than that of the old last station address it transmits a 'set last' frame to all stations in the ring informing them that it is the new last station. However, if TS is the first station it transmits a 'set last' frame when it identifies the address of the previous station which is definitely the address of the last station in the ring.

(b) In the ring maintenance process: If TS is the station before the last it transmits a 'set last' frame when the last station fails to respond to a 'token', so TS transmits a 'set last' frame to all stations informing them that it is now the last station of the ring.

Action on receipt:

(a) In the initialisation process: When TS receives a 'set last' frame it sets the last station number using information included in this frame.

(b) In the ring maintenance process: Same as (a).

WHO FIRST

When transmitted:

(a) In the initialisation process: If TS is a new station, on joining an established ring, it transmits a 'who first' frame.

(b) In the ring maintenance process: Not used.
Action on receipt:

(a) In the initialisation process: Not used.

(b) In the ring maintenance process: When TS receives a 'who first' frame it responds by sending 'set first'.

SET FIRST

When transmitted: -

(a) In the initialisation process: If TS is a new station, on joining an established ring, and its address is less than that of the old first station address it transmits a 'set first' frame to all stations in the ring informing them that it is now the first station.

(b) In the ring maintenance process: If TS is the station next to the first it transmits a 'set first' frame when the first station fails.

Action on receipt:

(a) In the initialisation process: When TS receives a 'set first' frame it sets the first station number using information included in this frame.

(b) In the ring maintenance process: Same as (a).

INIT DONE

When transmitted:

(a) In the initialisation process: If TS is the first station it generates a 'init done' frame as the final stage of the ring initialisation process. 'init done' is
sent to all other stations allowing them to enter steady state operation.

(b) In the ring maintenance process: Not used.

**Action on receipt:**

(a) In the initialisation process: TS enters steady state operation upon receipt of this control frame.

(b) In the ring maintenance process: Not used.

**DATA**

*When transmitted:* 'data' is transmitted by TS soon as it receives the 'token'. Note that this frame is sent directly to the destination address, i.e. it doesn't have to propagate around the ring.

**Action on receipt:**

When TS receives a 'data' frame it requests a DMA transfer to the main processor. Note that data frames are sent and received during steady state operation only.
Appendix B

HARDWARE DESIGN OF THE NETWORK INTERFACE BLOCK

B.1 TMS 9650

(a) GENERAL DESCRIPTION

The information presented here is derived from the TMS9650 Multiprocessor Interface (MPIF) Data Manual. Only the most relevant aspects have been selected being based on the information needed to understand the architecture and the operation of the TMS9650.

The TMS 9650 Multiprocessor Interface device (MPIF) provides a bit-parallel, asynchronous communications interface for passing messages and data between two processors or processor system. It behaves as a standard peripheral interface, consisting of eight programmable registers at each of its two ports and furnishes access to 256 bytes of random-access memory (RAM) used to buffer data transmitted between ports. The MPIF supplies arbitration logic to resolve RAM-access conflicts between the two processor systems. The MPIF can be used to connect virtually any 8-bit or 16-bit microprocessor to any 8-bit or 16-bit microprocessor having the capability of interfacing to standard memory or peripheral devices.
(b) ARCHITECTURE

The architecture of the MPIF is shown in block diagram form in Fig. B.1. It is built around a 256 x 8-bit static RAM and includes two complete data paths. Each data path connects the microprocessor interface to the appropriate on-chip register under control of the external interface control signals, register select lines, and an arbitration latch.

Both interfaces have access to the RAM via data registers. These are simple bidirectional buffers between the RAM and the data paths, and involve no storage.

Each data register has associated with it an address pointer register that supplies the address to the RAM when the corresponding data register is used. The address registers can be written to and read from both microprocessor interfaces.

Two message registers are provided, one assigned to each port. Each interface can read and write its own message register but only read that of the other interface.

The control register provides for the configuration and control of the MPIF. It includes the enabling control for the various MPIF interrupt requests.

The status register shows the condition of the interrupt sources.

The RAM can only be used by one host microprocessor, via its data register, at any one time. The two outputs of the arbitration latch, ACTA and ACTB, control access to the RAM. These outputs select the RAM data and address buses from
either the DATA A and ADDR A registers or the DATA B and ADDR B registers respectively. ACTA becomes true when the data register of port A is addressed, but only if ACTB is not already true and port B has not asserted a lockout. A corresponding definition applies for ACTB. Hence, the two signals are mutually exclusive, which ensures that both interfaces cannot use the RAM simultaneously. The MPIF provides its host microprocessors with continuous access to all other registers.

(c) REGISTER DESCRIPTION

The MPIF occupies eight locations in the memory map of each host system. It is so arranged that the registers accessible at the same location of each port serve the same function. The ports of the MPIF are therefore completely identical and can be reversed without software or hardware changes. For the purpose of naming the locations in the memory map, the port under consideration is referred to as the local port and the other is referred to as the remote port. The location and function of each register is shown in Table B.1.

DATA REGISTER

Each port can read and write its own DATA register at the two memory locations designed as DATA and DATA/INCREMENT. If a memory operation is performed to the RAM via the DATA/INCREMENT location, the corresponding address pointer register will be incremented on completion of the memory cycle. This will not happen if the DATA location is used.

ADDRESS POINTER REGISTER
Each port can read and write its own address pointer register at the location designated as the LOCAL ADDRESS POINTER (LAP) and can read and write the other port's pointer at the REMOTE ADDRESS POINTER (RAP). This enables each port to determine where in the RAM it will operate by setting up its own LAP. Alternatively, the management of the RAM can be under the control of only one port, which sets up both address pointers. Each pointer register will cycle through the value FF Hex (i.e. increment to 00 Hex).

MESSAGE REGISTER

Each port can read and write its own message register at the location designated as MESSAGE OUT. In addition, it only reads that of the other port at MESSAGE IN location.

CONTROL REGISTER

Control registers can be written to and read by their respective hosts at any time. The bit assignment is shown in Table B.2:

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|-----------------------------------------------|
| LEA | IEN1 | IEN2 | IEN3 | IEN4 | IEN5 | SLOC | X |

Where;

INE1-INE5 Interrupt enable bits: When set to 1, these allow their respective interrupt status bits to set the INT status bit.
and pull low the INT line.

**LEA**

Lockout on equal address pointer: If this feature is set from either port, it is active for the entire device. When this feature is enabled and the address pointer registers become equal, the port corresponding to the last address pointer register to be loaded from either port or incremented will be locked out of the RAM. The lockout will persist as long as the above condition remains true.

**SLOC**

Software lockout bit: This provides a software-accessible means of requesting that the remote port be locked out of the RAM.

**STATUS REGISTER**

Each status register is read only and allows its corresponding host to inspect the status of various bits. All may cause an interrupt if the appropriate interrupt enable bit is set to a 1. The bit assignment is shown in Table B.3;

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>MI</td>
<td>MO</td>
<td>LPE</td>
<td>RPE</td>
<td>LAK</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Where;

**INT**

Interrupt asserted: An interrupt status bit has been set,
and the INT* line is pulled low.

MI  Message in interrupt: A byte should be read from the MESSAGE IN REGISTER. It is set when the remote pointer loads its MESSAGE OUT REGISTER and is cleared when the local port reads its MESSAGE IN REGISTER.

MO  Message out interrupt: The local message register is available for use. It is cleared when a byte is written to the local MESSAGE OUT REGISTER and set when the remote MESSAGE IN REGISTER is read.

LPE LAP equal RAP: The address pointer registers are equal, and the local pointer is the last one to be loaded from either port or incremented. It remains true as long as the condition persists.

RPE RAP equal LAP: The address pointer registers are equal, and the remote address pointer was the last one to be loaded from either port or incremented. It remains true as long as the condition persists.

LAK Lockout acknowledge: This is set following the assertion of SLOC by the local port when the lockout of the remote port from the RAM becomes effective. It is cleared when SLOC is cleared.

(d) PIN DESCRIPTION
Table B.4 defines the TMS9650 pin assignment and describes the function of each pin.

**B.2 PAL PROGRAMMING**

Programmable logic arrays (PALs) devices are integrated circuits that hardware designers can program to perform specific logic functions. The PAL's logical functions are described in terms of a set of Boolean equations. These are then translated to a fuse map by a compiler that knows the target PAL's structure. The compiler generates a JEDEC file format (Joint Electron Devic Engineering). The JEDEC file is then loaded into a PAL programmer, and the fuses are then blown.

The PAL type used in implementing the function of the Access Control Block (ACB) is PAL16L8. The compiler used to generate the JEDEC file is available on the Intel Microprocessor Development System (MDS). The compiler requires the designer to describe the type of the target PAL, pin assignment and the Boolean equations (Fig. B.2). The generated JEDEC file and the chip diagram are shown in Fig. B.3,4.

The same PAL type is used to implement the Station Select and Address Recognition Block (SSAR). The Boolean equations, the generated JEDEC file and the chip diagram are shown in Fig. B.5,6,7.
B.3 TIMING SIGNALS GENERATION

A 32 x 8 PROM (27S19) is used (Fig. B.8) to generate the control signals that perform the transfer of data from the local TMS9650 to the remote TMS9650 (destination station). The input address signals to the PROM are supplied by a 4-bit binary counter (74LS163). Since the PROM has five address inputs, the fifth input (A4) is connected to ground, therefore only the first 16 bytes of the PROM are programmed. The stored data in the PROM is shown in Table B.5. The RWRT*, LRD* and LWRT* signals are taken from D0, D1, D2 respectively.

Table B.5 Stored data in the PROM

<table>
<thead>
<tr>
<th>Address</th>
<th>ABCD</th>
<th>Data</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>CC</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>CC</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>EE</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>FF</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>FF</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>DD</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>DD</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>CC</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>CC</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>CC</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>EE</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>FF</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>FF</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>DD</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>DD</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>CC</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The clock input to the counter is coming produced by a 16 MHZ oscillator. The
timing diagram of the generated control signals are presented in Fig. B.9. One byte transfer sequence starts every time the B input signal to the PROM goes from low to high. The output of the PROM is always in tristate condition unless the PROM's CS* input is pulled low.

START OF TRANSMISSION

The transmission cycle starts when the STX* signal is set low by the 8031 CPU (Fig. B.10). Provided the bus is free (BUSY* inactive) the D input of the two D flip-flops goes high. The second D flip-flop is clocked by the B signal which goes from low to high on the start of every transfer sequence. This guarantees that TXEN* is activated on the start of the first coming transfer sequence. When TXEN* signal is active the output signals of the PROM are enabled, therefore data is transmitted from the local to the remote TMS9650.

END OF TRANSMISSION

The D input of the first D flip-flop is high, but Q will remain low until EINT* is activated (Fig. B.11). The TMS9650 is programmed to activate its EINT* signal when the last byte of the message has been transmitted. The activation of the EINT* signal sets the Q output of the first D flip-flop, this in turn causes an interrupt to the 8031 CPU informing it that message transfer has been completed. The CPU then resets this flip-flop via the CLR* signal and deactivates the STX* signal.
FIG B.1 MPIF BLOCK DIAGRAM
TABLE B.4 PIN ASSIGNMENT AND FUNCTION
<table>
<thead>
<tr>
<th>REGISTER SELECT LINES</th>
<th>REGISTER FUNCTION</th>
<th>REGISTER SELECTED</th>
<th>READ-WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>DATA INCREMENT</td>
<td>DATA A</td>
<td>DATA B</td>
</tr>
<tr>
<td>001</td>
<td>DATA</td>
<td>DATA A</td>
<td>DATA B</td>
</tr>
<tr>
<td>010</td>
<td>MESSAGE IN</td>
<td>MESSAGE B</td>
<td>MESSAGE A</td>
</tr>
<tr>
<td>011</td>
<td>MESSAGE OUT</td>
<td>MESSAGE A</td>
<td>MESSAGE B</td>
</tr>
<tr>
<td>100</td>
<td>CONTROL</td>
<td>CONTROL A</td>
<td>CONTROL B</td>
</tr>
<tr>
<td>101</td>
<td>LOCAL ADDRESS POINTER</td>
<td>ADDRESS A</td>
<td>ADDRESS B</td>
</tr>
<tr>
<td>110</td>
<td>STATUS</td>
<td>STATUS A</td>
<td>STATUS B</td>
</tr>
<tr>
<td>111</td>
<td>REMOTE ADDRESS POINTER</td>
<td>ADDRESS B</td>
<td>ADDRESS A</td>
</tr>
</tbody>
</table>

TABLE B.1 MPIF REGISTER MAP
BEST COPY

AVAILABLE

Variable print quality
DAMAGED TEXT IN ORIGINAL
FIG B.2 Pin assignment and the Boolean equations for the Access Control Block PAL (ACB)
FIG B.3  The generated JEDEC file for the ACB PAL
FIG B.5  Pin Assignment and the Boolean equations for the station select and Address recognition Block PAL (SSAR)
FIG B.6 The generated JEDEC file for the SSAR PAL
FIG B.7 Chip diagram of the SSAR PAL
FIG. B.8 CIRCUIT DIAGRAM OF THE TIMING CONTROL BLOCK.
FIG. B.9 PROM OUTPUT CONTROL SIGNALS.
FIG. B.10 START OF TRANSMISSION CYCLE.