The design of a front end configuration for a digital telephone exchange serving digital multifunction subscribers

Bell, R.R.

Award date:
1980

ECB 8.7

THE DESIGN OF A FRONT END CONFIGURATION FOR A DIGITAL TELEPHONE EXCHANGE SERVING DIGITAL, MULTIFUNCTION SUBSCRIBERS.

R.BELL

The work described in this report was carried out
at Philips' Telecommunicatie Industrie, Hilversum.
The author thanks the management for this opportunity,
while no opinions stated in the report may be assumed
to be those of P.T.I.
Thanks are also given to the project coaches at T.H. Eindhoven,
Prof. Ir. Heetman, Ir. Stevens and Ir. Dortmans, and
colleagues at P.T.I., for the various discussions and
suggestions made during the course of the project.

Summary

The object of this project was the design of a front end configuration for a digital telephone exchange serving (digital) multi-function subscribers.
The relevant characteristics of these subscriber stations are discussed, based on a loop system supplying three independent circuits of capacity 64, 8 and 8 kb/sec to each station.
A logical structure for the front end unit is presented in which three distinct areas can be distinguished, namely:

- that region supplying the connection of user circuits with the main body of the exchange
- that supporting the signalling systems between this unit and the subscriber stations respectively CPU. A layered protocol structure, resulting in the definition of an Integrated Access Protocol for all users, is also defined within this region.
- the internal control function of the unit, used to co-ordinate the signalling activity, support connections and generally manage the unit operation. Structured programming techniques are used to create a flexible software format within this function.

Detailed implementations of the various hard- and software elements involved are described.
As testing of these solutions was limited, due to a lack of time, they may only be considered as a first step in the final realisation of such a system.

Summary

Contents

1. INTRODUCTION

1.1. General

The penetration of digital techniques into the public telephone network is of increasing magnitude. P.C.M. transmission systems, operating at frequencies of 2, 8, 34 Mb/s and higher, are already operational and their use, in the future, is expected to increase.
A further step toward an integrated digital network (IDN) is the introduction of digital switching centres. At present these are operational at transit exchange level but in the near future local exchanges of this form can be expected. The last step toward an IDN will then be the digitalisation of the subscriber loop and the subscriber unit itself.

This last 'link' has been given much consideration in recent years and various transmission techniques using 2 or 4 wire cable and optical fibres have been proposed (ref. 1), although no clear choice can as yet be considered as a standard. Due to the present huge investments in a cable network based on a 2 wire connection at local level, it would appear that a solution for this type of system would be attractive, certainly for the short to medium term.
The great majority of these solutions offer not only the basic 64 kb/s circuit capacity, necessary for operation with (CCITT) standard P.C.M. systems, but also an additional circuit capacity for use with the extra facilities which are considered necessary to make the relatively expensive digital solution economically feasible (ref. 2).

The transition period between the conventional analogue network and the future all digital system will extend over a relatively long period of time and the compatability problems in this mixed environment are expected to be great, though not insoluble.

This transition state will limit the use which can be made of the extended features created in a digital subscriber loop system and much effort will be required to design the required flexibility into the new systems.

The principle of an IDN leads, logically, to the idea of an Integrated Services Digital Network (ISDN).
This communications network, supplying a basic circuit capacity of 64 kb/s to any user and creating a connection using circuit switching techniques, will cater for the needs of both telephony and data subscribers in a uniform manner via common transmission and switching facilities. The problems arising here are not only those of the transition from analogue to digital techniques but also the integration of two basically different services, i.e. telephony and data. The features and quality of service offered by the public telephony and public/private data networks at present, differ greatly in such respects as error rates, information transfer speeds, features available to a user and etc. The only solution would be to create a common network offering a compromise between the requirements of both, while keeping the quality of operation as high as is possible.

Due to the demand for extra features it seems reasonable to assume that the initial penetration of digital subscriber loops will occur within the commercial sector. Here, while digital transmission will be used between the user and PABX, the exchange itself will also operate using digital techniques. The user extension (telephone?) in such a system, if the full possibilities are used, will consist of one or more telephone sets and/or data terminals in combination with a relatively powerful signalling system to and from the exchange. This multi function set, while offering extended facilities to the user, will in turn demand a more complex and powerful form of exchange than is conventionally available.

Three major areas are seen within this framework, namely:

- exchanges,
- subscribers,
- transmission systems.

A brief description of the first two will be made in order to clarify some of the points involved while the latter, not being included in the scope of this work, will not be considered as such in any further detail.

1.2. Exchange

The main function of a telephone exchange is to create, either a part of, or a complete connection between two subscribers (users). One of these subscribers, the calling party, indicates at the outset of the connection the identity of the other user - the called party - via dial digits. From this information the exchange must eventually make the required connection.

Two main types of exchange can be considered:

- local, in which the subscribers are connected directly, via subscriber lines, to the exchange;
- transit, which as the name implies, acts only as a link in the connection process and has no subscribers directly connected to it.

Considering further the local type exchange, it is normal, for economic reasons, to have more subscribers associated with it than it can serve at any given time, i.e. the traffic generated by the population of subscribers is concentrated. This is acceptable due to the statistics of the traffic (load) generated by a given subscriber but is limited by the quality of service which it must give the population as a whole.

The more conventional systems switch the incoming information, in its analogue form, through to the required output, while in more modern systems digital switching is used in the exchange with a/d, d/a conversion occurring in the peripheral equipments. The bit stream created has a standard capacity of 64 kb/s (see CCITT reports).
Both of these types of exchange are of the stored programme control variety, a computer controlling the overall exchange operation.
In the future it can be expected that these a/d and d/a conversions will take place in the user premises, leaving the exchange to deal with purely digital bit streams.
This does not alleviate the need for interface units, as the system itself must still be protected against the high voltages which can occur sporadically in the transmission systems and must (perhaps) act as source for the extension power supply.
There is also a tendency to split the physical exchange into smaller parts, which can themselves be placed closer to the user premises, e.g. concentrators or peripheral units, which have a more or less independent method of operation depending on the circumstances.
In general we can consider an exchange to be composed of four main parts, as shown in fig.1.1.

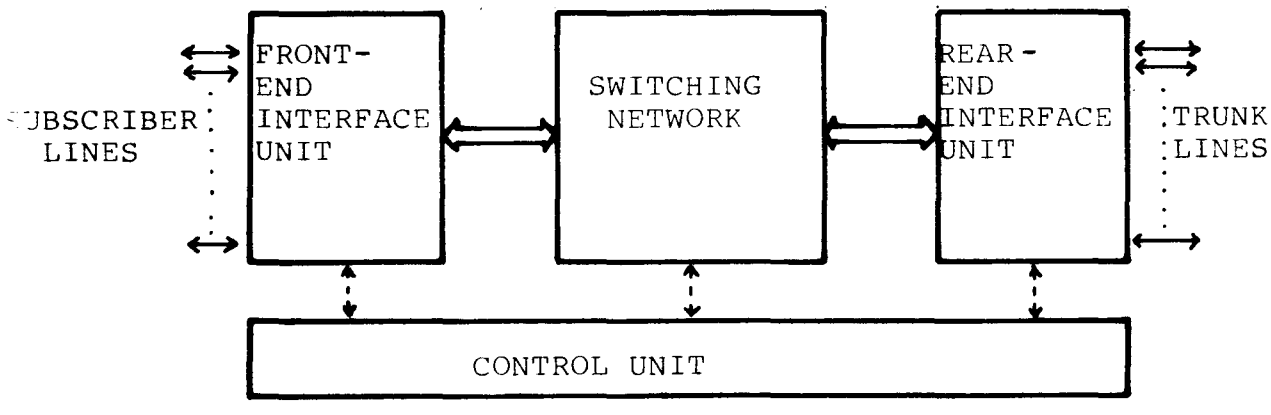


fig. 1.1. Basic Exchange Configuration

The front and rear end interface units are used as buffer functions between the switching network/network control functions and the subscriber, respectively trunk, transmission lines.

The switching network creates the connection between two (or more) points within the exchange and is so designed as to include the necessary concentration function.
The control unit organises the operation of the exchange as a whole. It stores any necessary administrative information and acts as the backbone over which the control information, necessary to create and maintain a connection, flows.
With the increasing power of the micro-processors/computers, it is no longer necessary to maintain a single, large, computer for this function. Multiprocessor control will soon be normal practice. Also the splitting up of the control function into a parent body with a number of satellites is a real possibility, thus reducing the workload within the parent body itself. This idea can be taken to the point where each of the units in fig.1.1 has a part of the control function incorporated within it - which in turn necessitates the use of some form of inter-processor communication between this satellite and the main control function.

The work to be carried out consists of the definition of operation, and the design of a front end interface unit for such a system in which the subscriber lines operate using digital transmission techniques.

1.3. Subscriber

Each user in the system requires two basic features, namely:

- a method of making a connection with a specific second party;
- the facility to transfer information of some sort to, and, if necessary, to receive corresponding information from, that second party.

The former is supplied via a signalling circuit between user and exchange, the latter via the actual connection, i.e. the user circuit.

Two types of users can be distinguished:

- one which transmits audio signals via a telephone;
- a second which transmits data (basically digital) via some form of terminal.

As is seen in conventional systems the onus lies in the handling of an analogue signal from the sources, digital signals must therefore be processed in some way to create an analogue equivalent. The telephone traffic far exceeds that from data sources. In the future, although the latter point may still hold, the former will shift to an onus on digital signals, with the consequence that the analogue signals will have to be processed in some manner.

As has been said previously, in the future systems the a/d and d/a conversions necessary will take place on the subscriber premises - transmission over the subscriber cable being accomplished using some form of digital technique.
Once this digital transmission has been accepted then the next step is to use some time division technique to multiplex bit streams from various sources onto this single transfer system. These sources could be considered as:

a) a further telephone set (with its a/d, d/a conversion);
and/or b) a data terminal (producing digital bit streams directly);
and/or c) some form of telemetry unit;
d) the necessary signalling for the combination.

As is seen, a user station is being considered which has multi function features, i.e. a combination of various information carrying circuits. Limits to the number of users which can be combined in such a manner are set by operational and technical considerations, such as the maximum transmission capacity in the system.

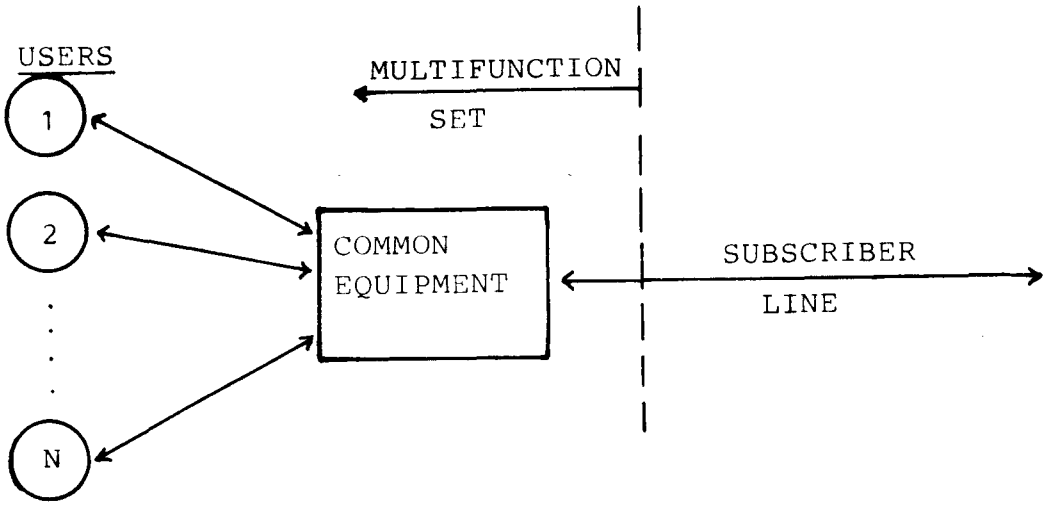


fig. 1.2. Multi-function subscriber

A depiction of such a unit is shown in fig.1.2, while a more detailed description will be given in the next chapter.

2. SUBSCRIBER STATION

2.1. Preamble

This brief discussion is meant to introduce the reader to the type of subscriber which the front end unit must service. The interfaces, and etc., within the station equipment are defined where necessary, considering the co-operative action with the front end unit. These definitions, though based on expected configurations, may not be those eventually chosen by the international standardisation bodies.

2.2. General

As has been said in the introduction, the subscriber station being considered is multifunctional in that not only conventional telephony is being considered but also additional features.
Digitalisation of the user information (speech or etc.) also occurs within this station, leaving binary digits to be transmitted to the exchange.
The additional features mentioned are to be in the form of:

- extra user equipments in addition to the conventional telephone;
- extended signalling facilities.

As has also been mentioned the precise form in which these features should occur and the bit rates available to them is, at present, a point of lively discussion throughout the world.

As a basis for the study being carried out the following subscriber station will be assumed, as shown in figure 2.1, consisting of basically:

- a conventional telephone, in combination with extended signalling facilities;
- a data terminal plus the necessary signalling facilities;
- a telemetry unit (e.g. fire alarm or etc.);
- the various equipments necessary to support the operation of these units.

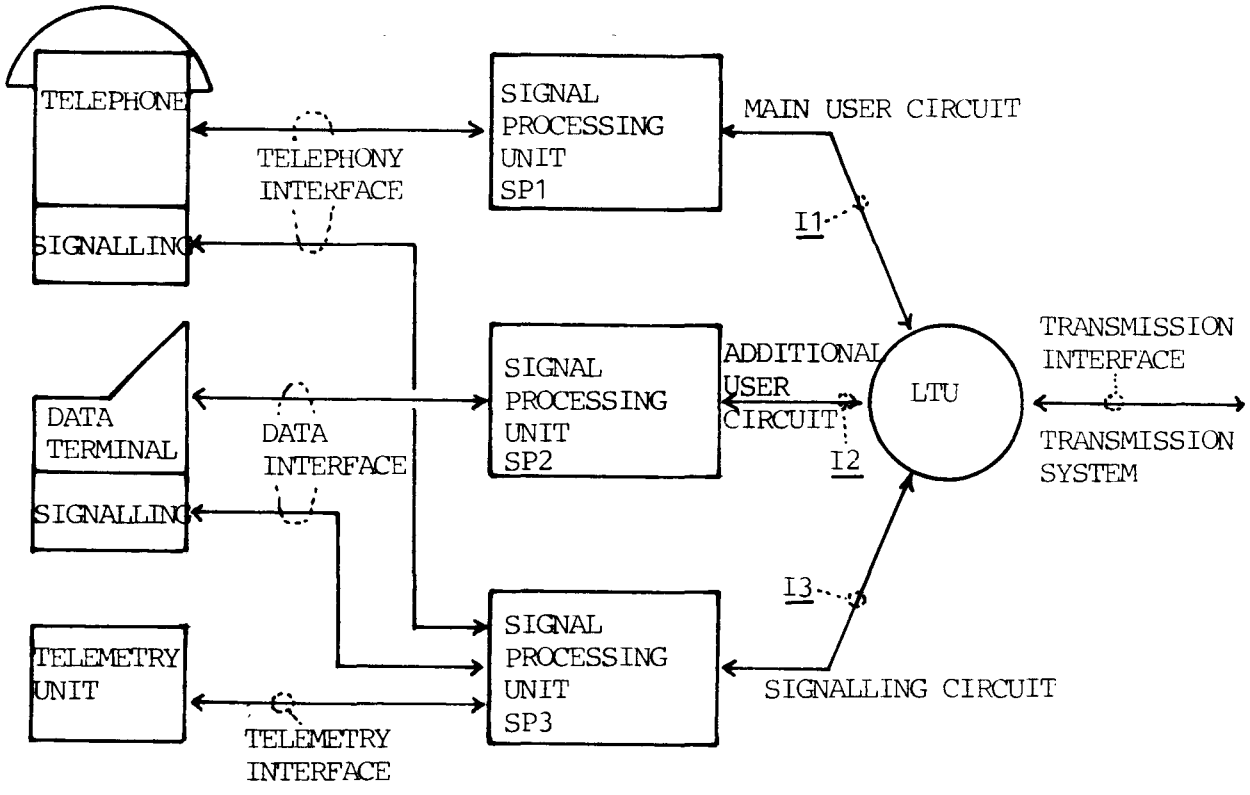


fig. 2.1

2.3. Operation

2.3.1. Interfaces

In order to create standard formats for the information carrying circuits shown in fig. 2.1, the three signal processing, or handling, units have been included. To the right (in fig. 2.1) of these units standard interfaces have been defined, as shown in fig. 2.2.

The scope of this study does not extend to the definition of those interfaces between user equipments and the signal processing units - although these are necessary for the global system operation.

Suffice it to say that:

- the telephony interface would be based on conventional analogue signals;
- the data interface would be standard X20, X21 or something similar;
- the telemetry interface and the signalling sub-interfaces of the former two would depend on the signalling protocols/circuitry used.

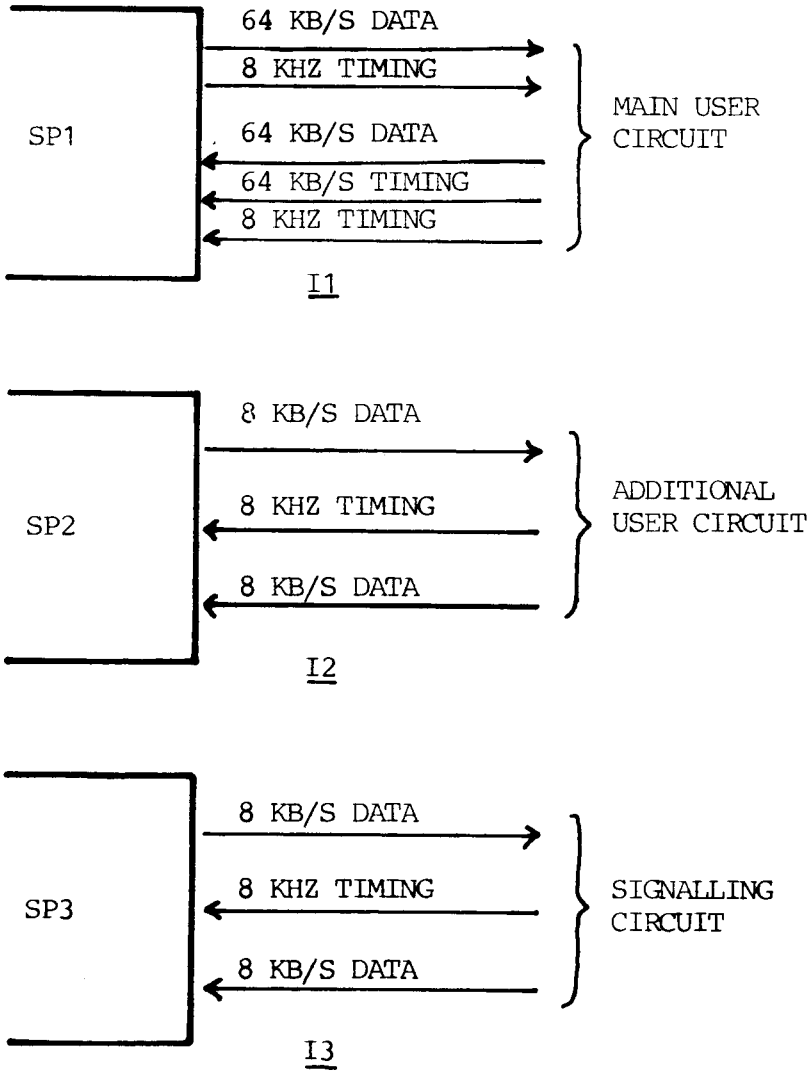


fig. 2.2

2.3.2. Support Units

These are used:-

a) SP1:

to implement the required a/d, d/a conversions, to sort the information into a form compatible with the corresponding interface structure and to support/maintain this interface.

SP2:

to act as the required DCE function for the data terminal equipment and, as in SP1, to support, and etc., the interface structure.

SP3:

This must maintain the interface structure (as in SP1 and 2). It must also control the signalling system between the extension and exchange. This signalling system will be considered in a later chapter - at this stage it will merely be stated that a link protocol must be supported over this circuit.

Any similar protocols operating over the user circuits would not be the responsibility of the front end unit e.g. the end to end protocol in telephony is controlled by the users directly.

Due to the infrequent and 'bursty' nature of information generated from its sources this signalling circuit will operate on a demand assignment basis. This entails that a signal to or from the exchange be allocated the whole channel capacity temporarily until the transfer is complete. This sharing of the circuit between users leads to the inclusion of a source/ destination identity field in any signal being transferred.

b) The line terminating unit (LTU) in fig. 2.1 must multiplex the three information streams into a single stream and cause this to be transmitted to the exchange in the required manner. It must also act as receiver (regenerator or etc.) for the information arriving from the exchange and demultiplex this into the corresponding output interfaces. The operation of the transmission system itself will not be considered, but it should be realised that various methods are possible. The LTU thus isolates the transmission problem.
This unit, common to all users, can be considered as the extreme front end of the exchange, i.e. the exchange makes contact with each user, individually, through the corresponding interface. This is important due to the structuring of the system and the possibility of creating maintainance procedures which in no way effect, or are detectable by, the user. A fourth user on SP3 can thus be considered as this common equipment within the subscriber station.

c) The power supply for the station can be obtained:

- from the electricity network or a battery on site;
- from a battery on site which is charged in some way from the exchange;
- centrally, purely from the exchange.

The choice, though not considered as part of this study, would seem, due to system reliability and economic considerations, to tend toward the last method, although this does raise rather complex technical problems. As far as is possible the system to be designed will cater for the use of all these methods.

3. FRONT END INTERFACE UNIT

3.1. Functions

Considering the division of the exchange as shown in fig. 1.1, the object of the project is the definition and implementation of the Front end Interface Unit (FEU). Globally this unit must act as a buffer between the users and the main body of the exchange. This buffering includes a pre-processing of the signalling/control information transferred between the two (which in turn leads to a reduction in the workload on the control function within the main body).
The main function of the unit is to connect corresponding input and output circuits together. This connection process is carried out on a concentrated basis, as described in appendix A, while the control and support functions necessary to accomplish this must be internal to the unit itself.
Certain 'real time' system actions and timing functions will, where efficient, be implemented in this peripheral area as well as the obvious internal management functions.

The unit itself is not considered as an independent exchange but merely as a partially autonomous slave to the main body where the more complex processing functions take place.

3.2. System structure

The unit is situated, physically, between the subscriber stations and the main body of the exchange. Connections with the former occur via the subscriber lines while the latter is connected to the front-end unit via an Information Highway.
This leads to the system structure shown in fig. 3.1.
Here the subscriber lines are terminated in the front-end unit with Line Terminating Units (LTU), while this is done for the Information Highway with a Highway Terminating Unit (HTU).

The main work to be carried out by the unit is done in the core, the definition of which is the object of this project.

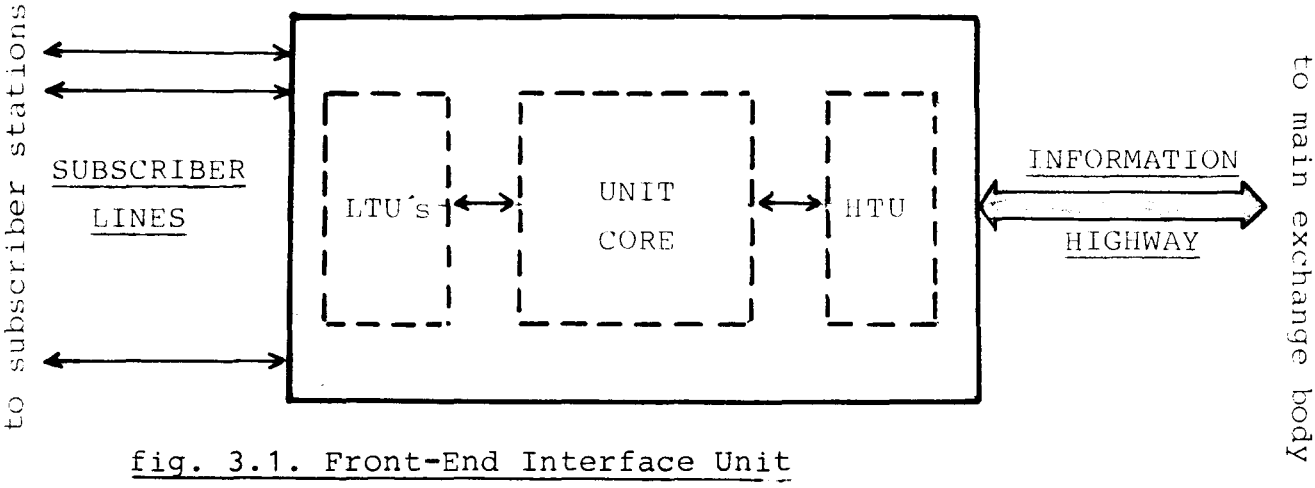


fig. 3.1. Front-End Interface Unit

3.2.1. LTU

These must carry out the same function as the corresponding units described in chapter 2 with respect to the subscriber station, namely the termination of the specific transmission system being used on the subscriber line. The interfaces between these and the unit core are of the standard type also described in chapter 2, the LTU's again being considered as part of the transmission system. Three interfaces are created per LTU for respectively:

- main user circuit,
- additional user circuit,
- signalling circuit.

The difference between these circuits is that while the former two, i.e. the user circuits, contain information to be transferred to (or from) the far-end user the latter contains purely control information and does not penetrate the system directly any further than the front-end unit.

The implementation of this unit is very much dependent on the type of transmission used and shall therefore be considered as a 'black box' supplying the necessary functions.

3.2.2. Information Highway

Information transfer (user and control) between the front-end unit and the main body of the exchange occurs via the Information Highway.

A serial bus structure, operating at a frequency of 2048 kHz, was chosen for this highway. The highway therefore consists of a four wire connection as shown in fig. 3.2.

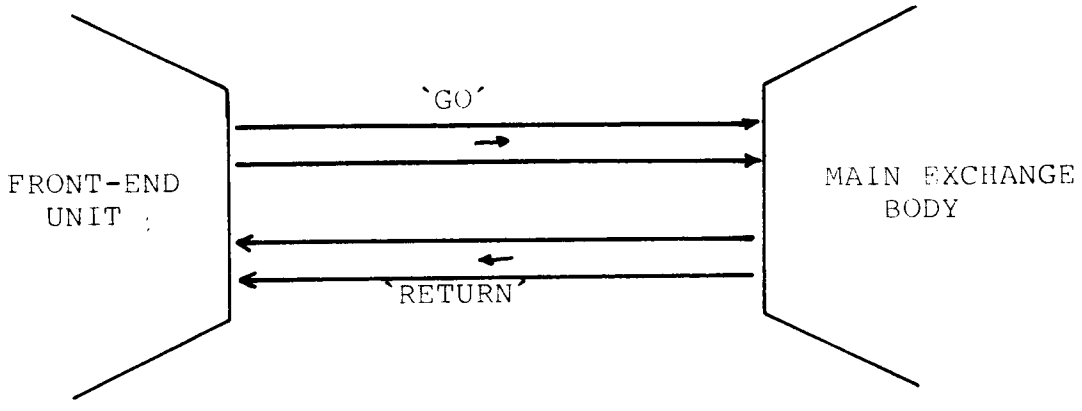


fig. 3.2.

Time division multiplexing techniques must therefore be used to implement the required information transfer to and from the highway.

Considering the 'Go' direction in fig. 3.2 each of the 32 output channels (see appendix A) is sequentially allocated a time slot on the highway.
This time slot consists of a portion of time corresponding to 8 (clock) periods of the carrier frequency, 2048 kHz, during which time information from a specified user may be transferred onto the highway. This group of 32 time slots is called a frame, which is repeated cyclically with the specific user being active during the same relative time slot in each frame. This leads to the required 64 kb/s transfer capacity per user (or output channel).

This is shown graphically in fig. 3.3 and 3.4 below.

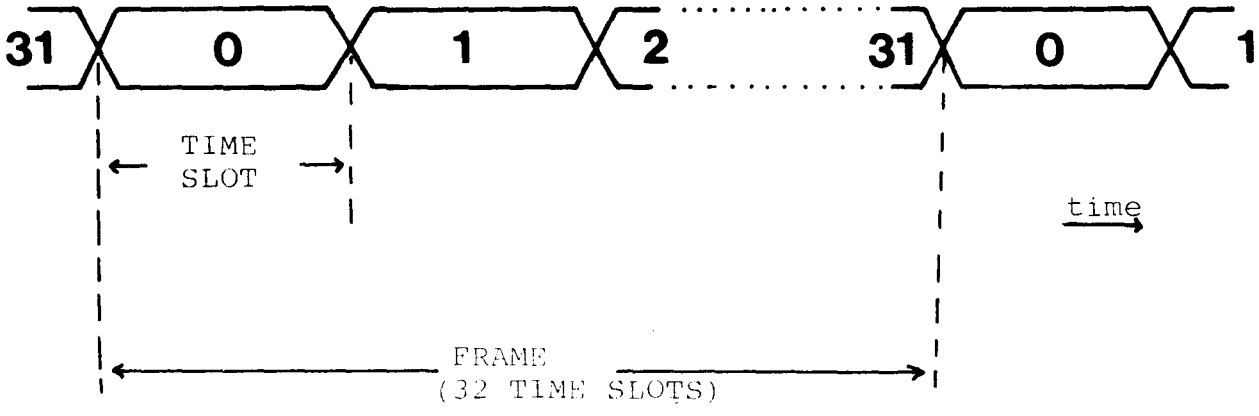


fig. 3.3. Highway Structure

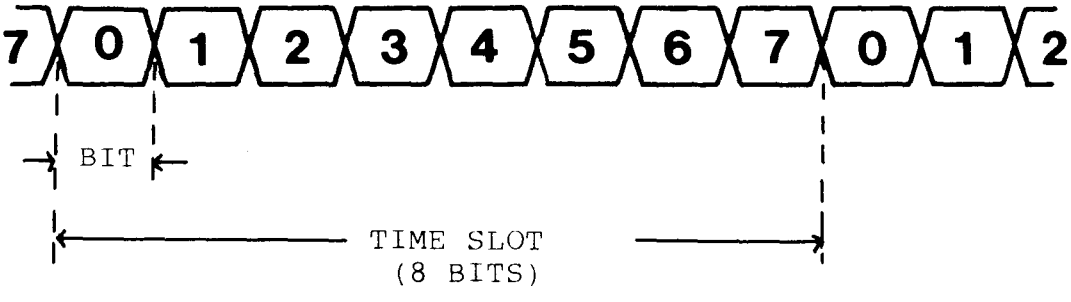


fig. 3.4. Time Slot Structure

Bit frequency = 2048 kHz → Highway capacity = 2048 kb/sec.

Time Slot Duration = 8 bits

Frame Duration = 32 Time Slots ≡ 125 μsec.

Effective capacity per circuit ≡ 8 bits/125 μsec.
= 64 kb/sec.

Of these 32 channels, created on the highway, one will be dedicated to control information transfer between the FEU and Main body of the exchange. The remaining 31 being thus available for assignment to any specific user.

The 'return' half of the highway has an identical structure thus creating the required duplex circuits. Timing signals (clocks) within the FEU could be obtained from:

- internal oscillator;
- regeneration of a timing signal if a transmission system is being used;
- directly from a dedicated 'clock line' from the main body of the exchange.

Synchronisation of the frame position, i.e. the start of a frame, could also be supplied via an apart line from the main body or, in the case where a transmission system is used, from a dedicated synchronisation slot.

The actual timing/synchronisation techniques used and the procedures which would be necessary on failure of the system will not be considered further.

This serial bus structure, in conjunction with a standard second order PCM transmission system, allows relatively long distances to be used between the front-end and main body of the exchange, i.e. satellite operation.

3.2.3. Highway Terminating Unit

This is simply the physical termination of the highway and any associated timing and synchronisation lines. It thus consists of the required line drivers and receivers plus any regeneration circuitry, and etc., necessary for system operation. This will be considered somewhat further in conjunction with the user circuit management in chapter 4.

3.2.4. Sytem Core

This is the area within which the main functions of the unit must be carried out.
This 'core' can be seen to consist of three major functions, as shown in fig. 3.5, these are briefly:

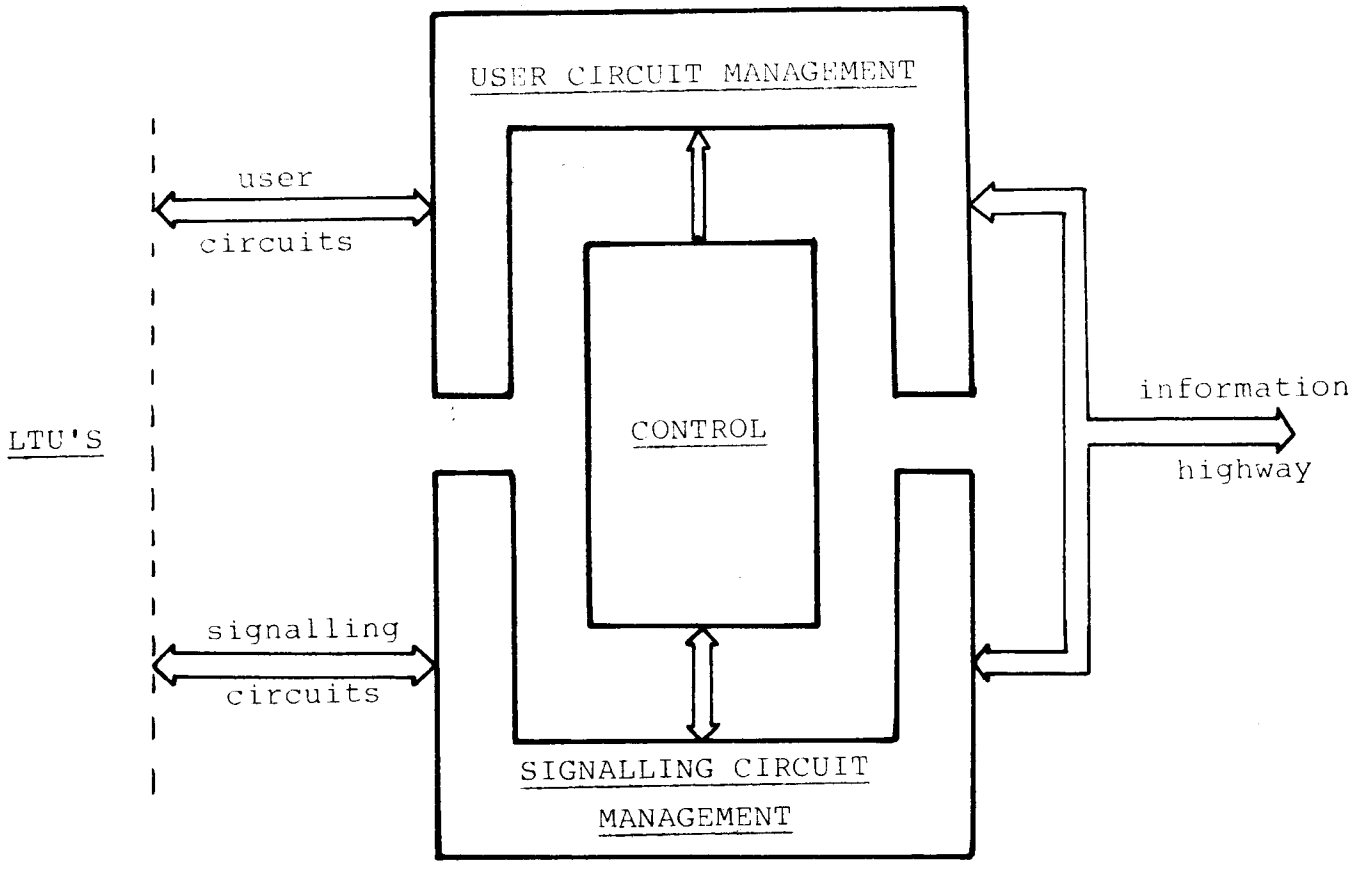


fig. 3.5. Core functions

1. User Circuit Management:
   This function must implement the mapping, or transfer, of the information streams on the user circuits to the specific time slot on the Information Highway (and vice versa).

2. Signalling Circuit Management:
   This function must transfer the messages received from the users (via the signalling circuit) to the control function and vice versa. It must also do this for message transfers between the front-end and main control functions via a dedicated circuit on the information highway.

The responsibility for protocol verification on the various circuits also lies within this function.

3. Control Function:
   This must carry out the actual pre-processing which is to be done in the unit. It must also support the FEU operation as a whole by initiating and/or synchronising any operations which must be carried out.

The physical structure of this 'core' is shown, albeit globally, in fig. 3.6.

As is seen, the control function consists of:

- a processor,
- a memory block,
- a set of 'internal' system buses,
- a collection of interface circuits.

I/O memory mapping is used throughout, the peripheral functions indicating, via an interrupt routine, that a message is to be transferred to the processor.

The precise form of the I/O buffers and other interface circuitry will be discussed more fully in combination with the functions which are being coupled with the control block. At this stage a simplified model of these buffers will be given, namely fig. 3.7.
As the messages transferred between functions consist of two bytes the buffers must contain some form of conversion unit - the data bus being 8 bits wide. Control circuitry plus a temporary storage facility used for synchronisation purposes are also included in these buffers.

In the remaining chapters of this report these three functions will be described in greater detail and where appropriate their implementation will be described.

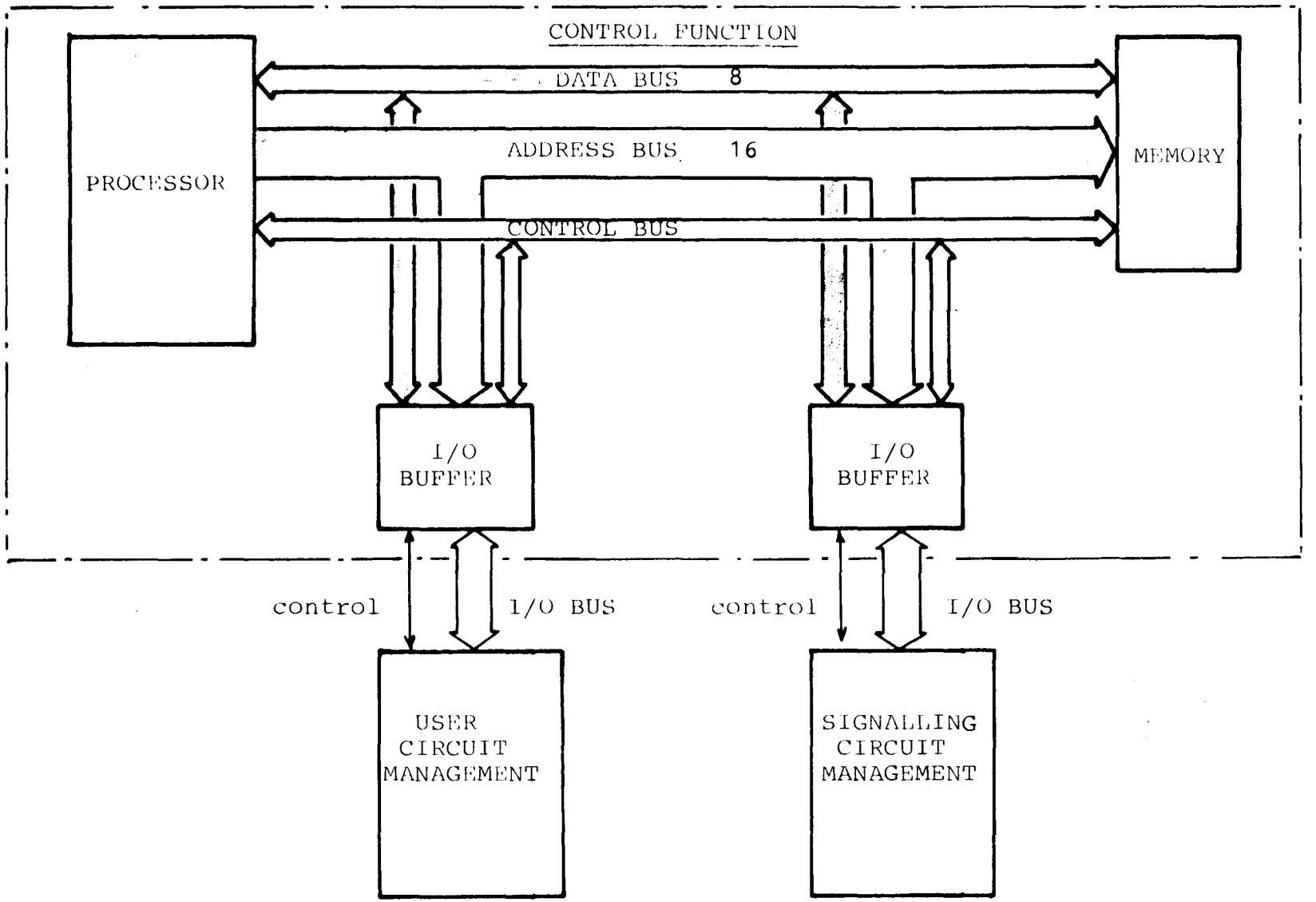


Fig.3.6 CONTROL FUNCTION STRUCTURE

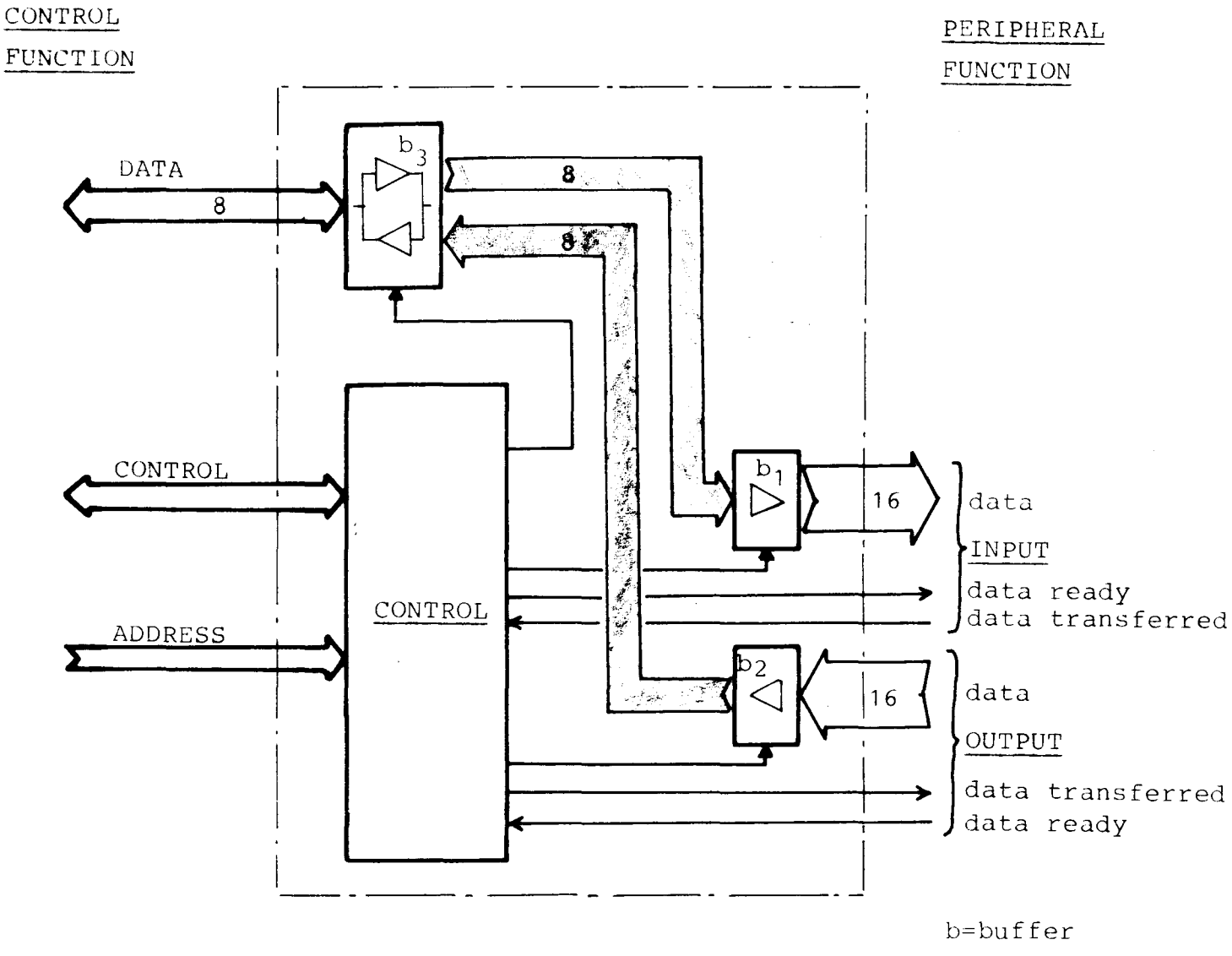


Fig.3.7 I/O BUFFER

4. USER CIRCUIT MANAGEMENT

4.1. Introduction

This function must implement the required connections between Line Terminating Unit (LTU) and Information Highway (IH) for each user circuit, as indicated in fig. 4.1. The specific details of such a connection (i.e. which time slot has been allocated to a given user circuit) is supplied by the control function via the I/O buffer of fig. 4.1.
As discussed in chapter 3 the interface with the Information Highway consists of an 8 bit burst, occurring with a basic bit rate of 2048 kb/sec., at intervals of 125 μsec. This is depicted in fig. 4.2.a. The circuit is formed by two of the channels thus created, one in either direction. The user circuit interfaces with the LTU consist of the continuous bit streams shown in fig. 4.2.b and c. These are identical to the corresponding interfaces in the subscriber stations, as discussed in chapter 2, but with an extra 'timing' connection added.
Two types of user circuits are thus met at the LTU:

- main user circuit, operating at 64 kb/sec. and containing the digitised speech signals;
- additional user circuit, operating at 8 kb/sec. and containing the data signals.

As the IH supplies a basic circuit capacity of 64 kb/sec. some technique must be used in connection with the latter in order to create compatability between the two bit rates. A transparent connection is to be created i.e. no real context processing of the user circuit contents is to be carried out. Any such action (namely w.r.t. the additional user circuit contents) is envisaged as occurring in the main body of the exchange, where the necessary equipment can be incorporated more economically than in the front-end unit (see appendix B).
This leads to the use of simple bit multiplication/division techniques to create the required compatability mentioned above.

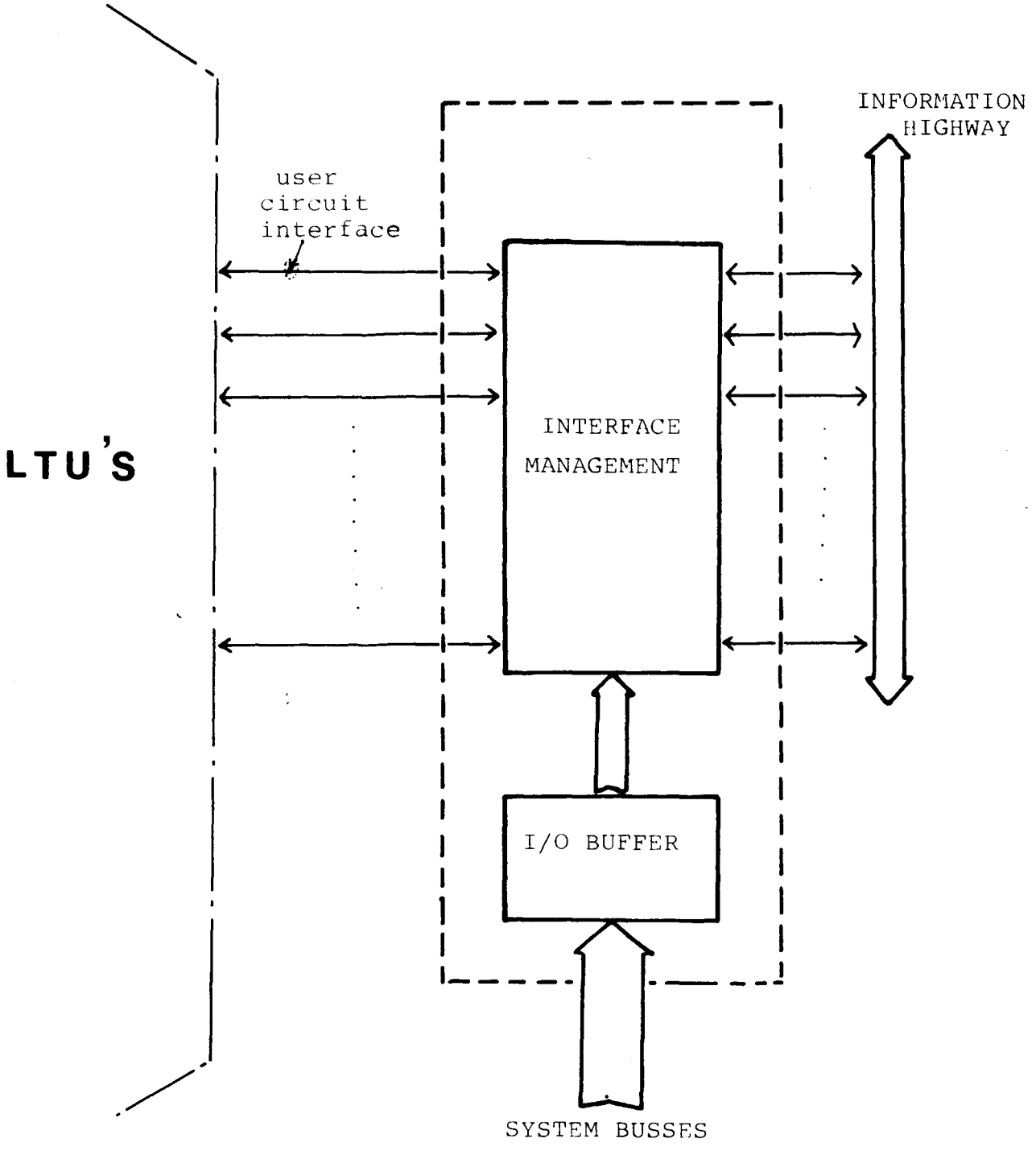


Fig.4.1 INTERFACE MANAGEMENT FUNCTION

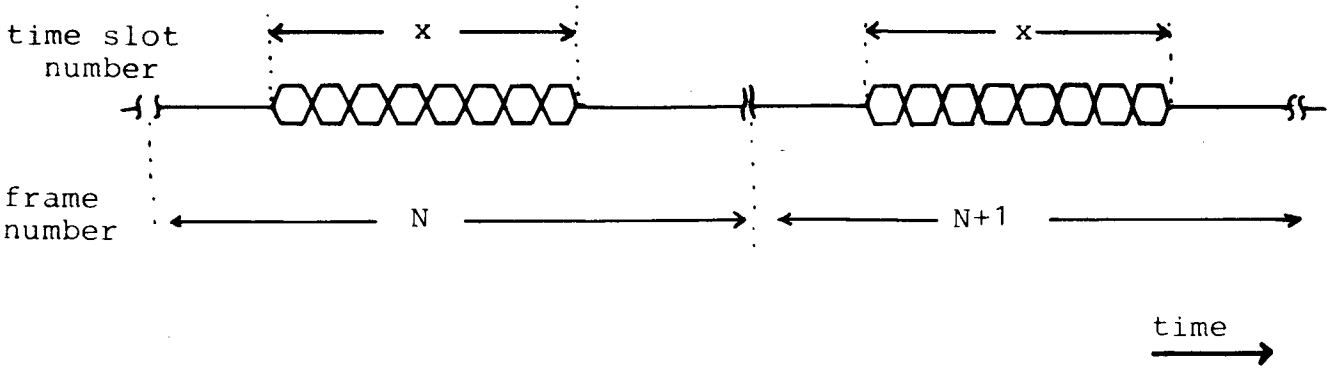


Fig 4.2.a INFORMATION HIGHWAY INTERFACE

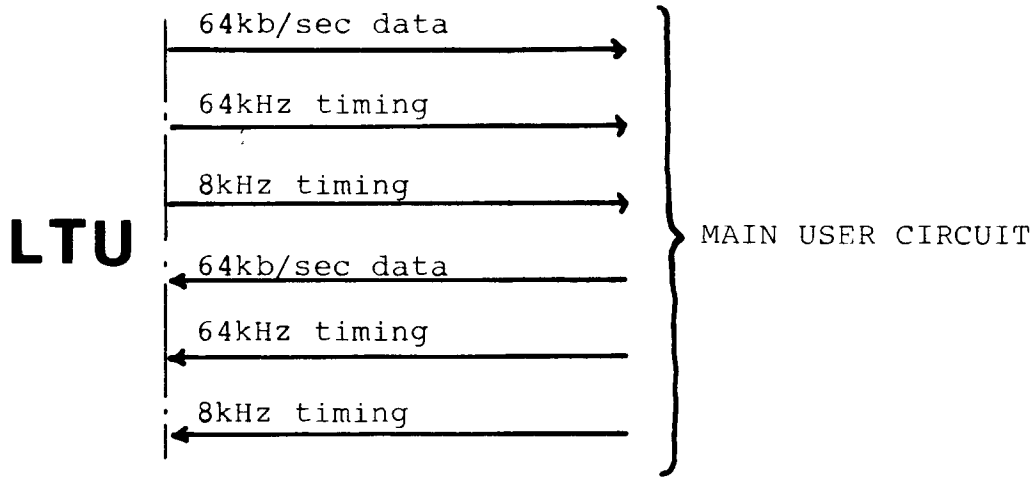


Fig 4.2.b MAIN USER CIRCUIT INTERFACE

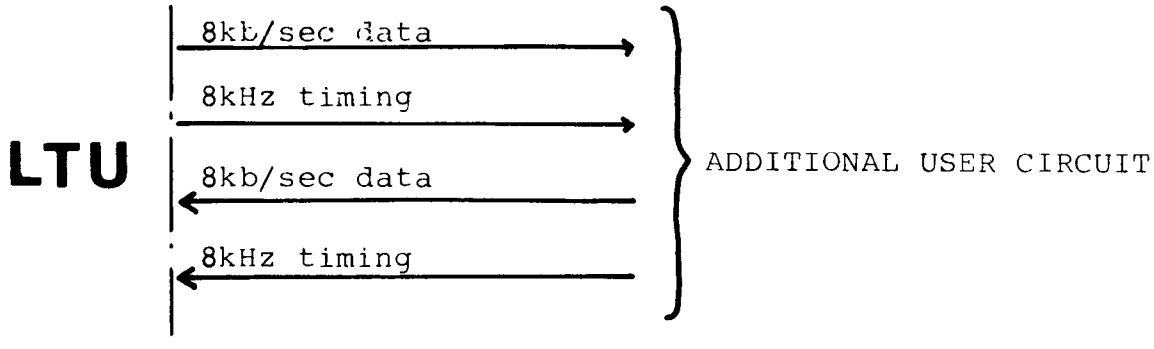


Fig 4.2.c ADDITIONAL USER CIRCUIT INTERFACE

A given bit on the 8 kb/sec. interface is simply repeated 8 times to give the required 64 kb/sec. on the IH. In the opposite direction any one of the (identical) bits within a time slot can be transferred to the interface with the LTU.

Messages are transferred from the control function via an I/O buffer, as shown in fig. 4.1. In the present case no messages, apart from a buffer status reading, are returned to the control function. This return path could be included at a later time w.r.t. error control, e.g. a received message could be transmitted back to the control function where its authenticity could be tested. Fig. 4.3 gives the basic operation of this buffer, i.e. as a temporary message store. The actual circuit used will be explained later in this chapter.

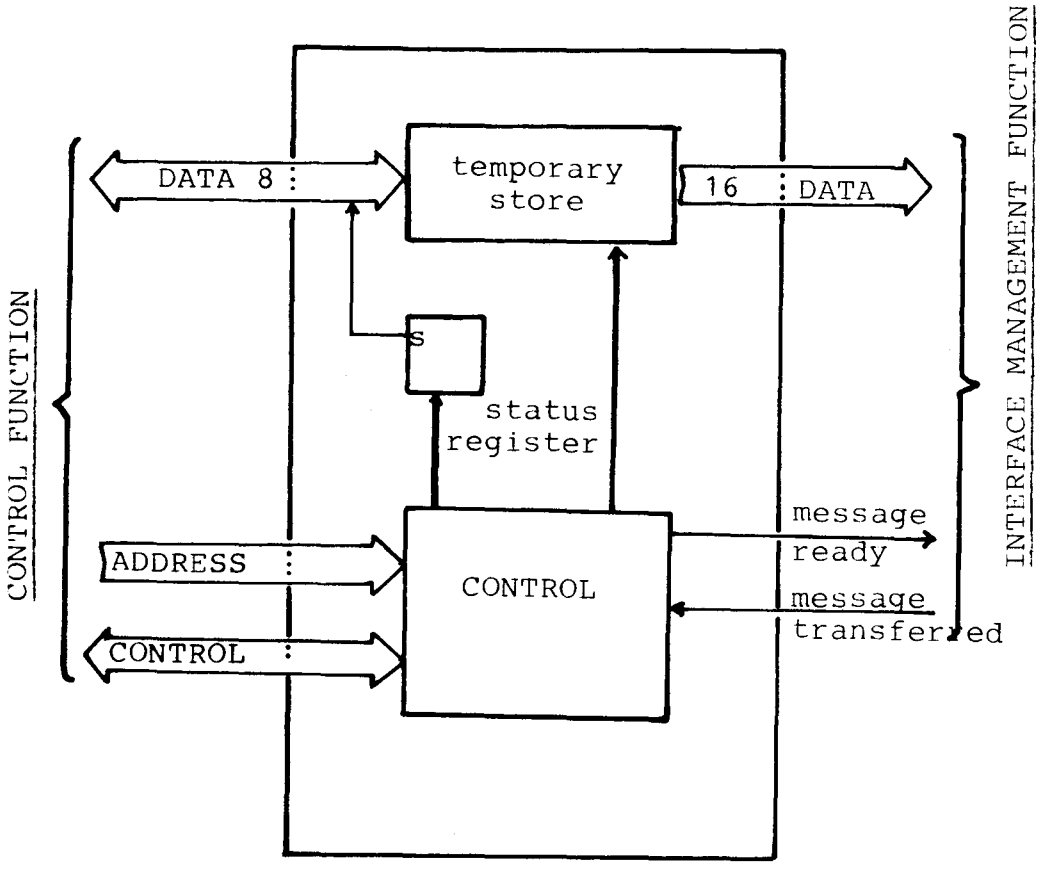


Fig.4.3 I/O BUFFER

The messages transferred from the control function must contain two parameters:

- the identity of the user circuit to which the message applies. This consists of the extension number plus the specific user number (main or additional) on that extension ;
- the message itself. This is either the time slot number to which the specified user circuit has been allocated or, in the case where a connection is to be ended, a command to free or de-allocate the time slot.

A two byte message has been chosen as standard for this and for message exchange throughout the front-end unit. The basic structure of the message is shown in fig. 4.4.

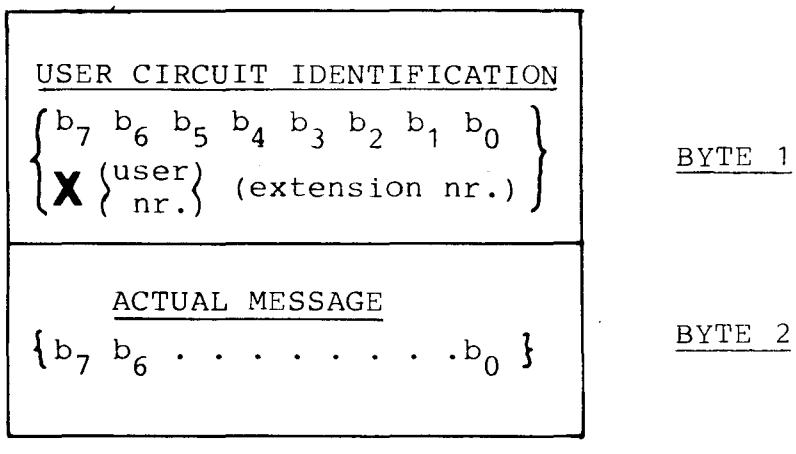


Fig. 4.4 MESSAGE FORMAT

4.2. Structure

Basically, for each user circuit an interfacing unit is inserted between LTU and IH, as shown in fig. 4.5.a. This interface unit consists of the elements shown in fig. 4.5.b, namely:

- input and output buffers, connected to the LTU;
- input and output buffers, connected to the IH;
- a transfer block which must implement the information transfer between these buffers in a way specified by the control function.

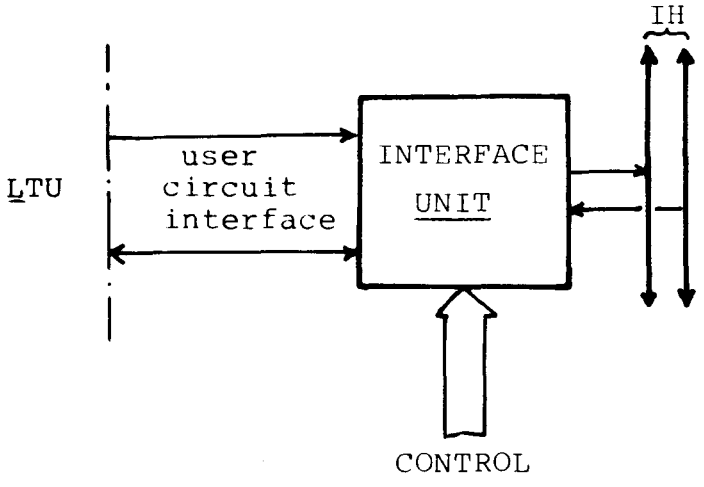


Fig.4.5.a INTERFACE UNIT

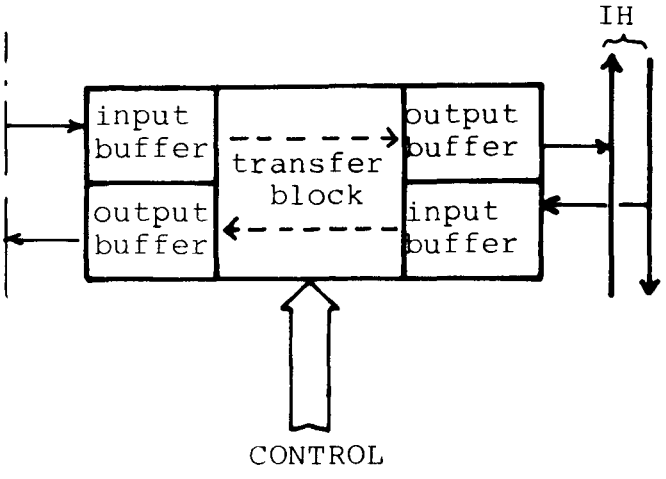


Fig.4.5.b EXPANDED INTERFACE UNIT

Due to the high drive capabilities and complex circuitry required when operating in combination with a long Information Highway, a Highway Terminating Unit (HTU) is included in the front end unit, as shown in fig. 3.1. This unit, inserted between the interface units and the highway itself, must supply the drive/receive capabilities for operation over the highway plus any timing/synchronisation control facilities required.
Connections between the interface units and the HTU occur via the Internal Information Bus (IIB), the basic structure of which is identical with the IH as previously described. Fig. 4.6.a gives an indication of the above structure pictorially, while fig. 4.6.b gives a more detailed circuit diagram of the unit proposed.

Each user circuit interface unit could be connected individually with the IIB, as shown in fig. 4.7.a.
Using this technique every interface unit would include:

- control circuitry,
- timing circuitry,
- circuitry for connection to the IIB, namely bus drivers and receivers.

The bus loading in such a system, i.e. 64 units, would be excessive.

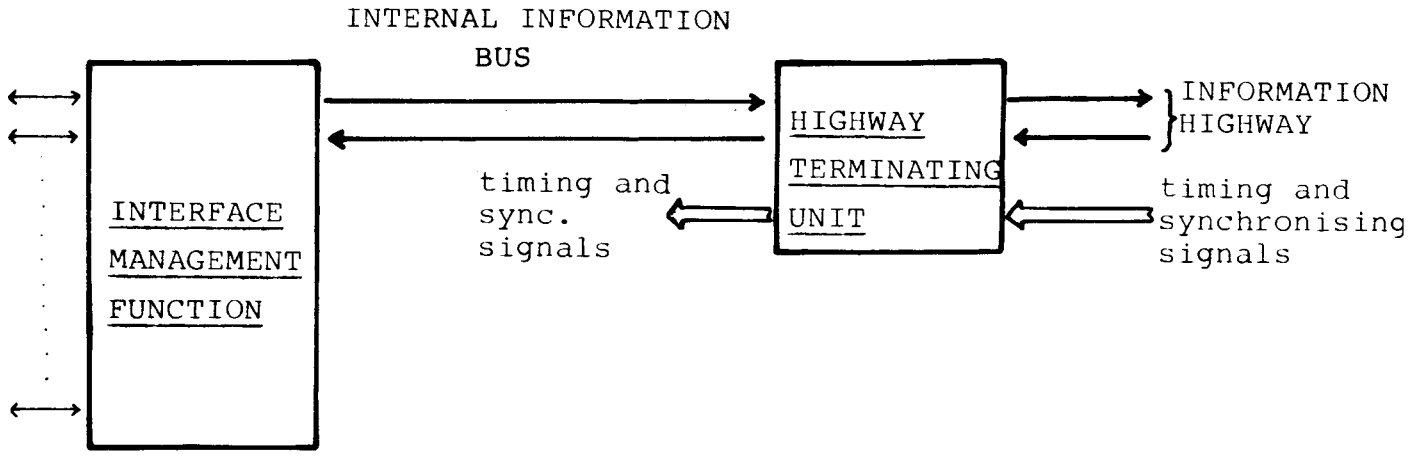


Fig.4.6 a) CONNECTION WITH INFORMATION HIGHWAY

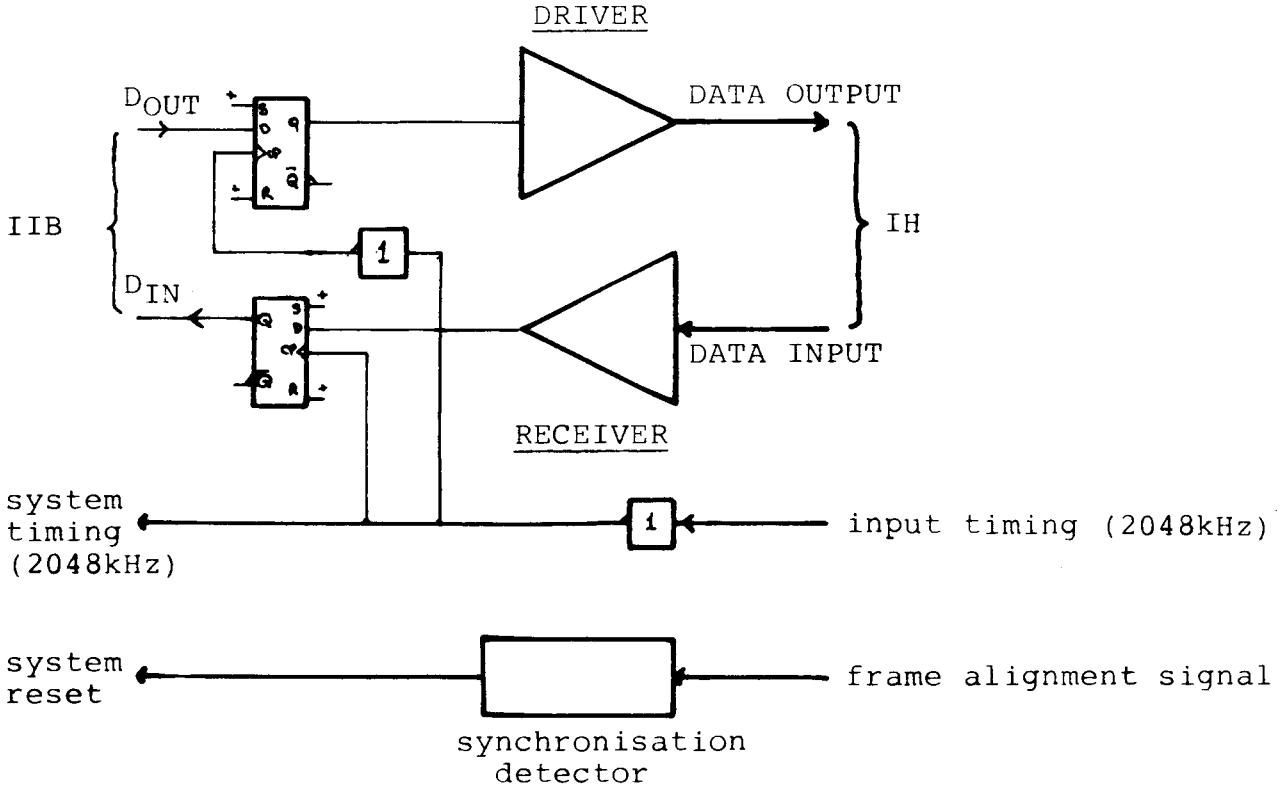


Fig.4.6 b) HTU BLOCK DIAGRAM

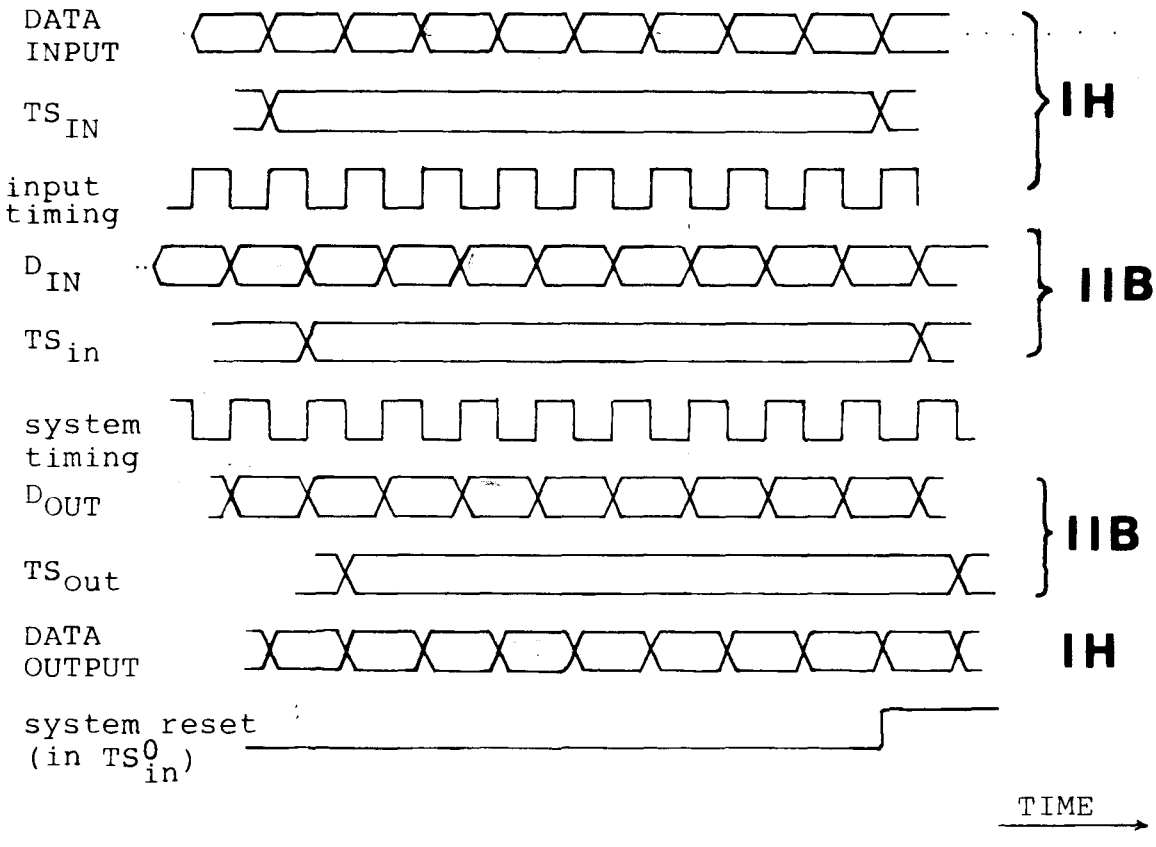


Fig.4.6 c) HTU TIMING DIAGRAM

By combining the interface units for the two user circuits connected to a single subscriber station, as shown in fig. 4.7.b an improvement can be achieved, namely a reduction of the bus loading by one half.

A further improvement is to concentrate the main control functions of the units into a single control block, as shown in fig. 4.7.c. This leaves a relatively simple residual control function in the interface unit which acts as a slave to this central block.
This last step could, when considering a custom designed integrated circuit solution for the interface unit, be argueable. It should be kept in mind that due to the required operating speeds - namely a new time slot allocation every ± 4 µsec., the 'core' processor could not be used directly to drive the individual interface units. Some form of centralised time slot clock would thus be necessary in any case. The hardware saved using the concentrated control is considered worthwhile.

4.3. Implementation

The structure, as given in fig. 4.7.c, is to be implemented. A description of the operation, and a detailed realisation of the required hardware, will be given in the following sections.

4.3.1. Control Block

This must basically:

- support the message exchange with the Front End Control Function (via the I/O buffer);
- store these messages;
- implement the user circuit/time slot mappings described by these messages, i.e. activate the interface unit specified during a given time slot;
- supply the control/timing signals to the interface units which are necessary to create the actual connections.

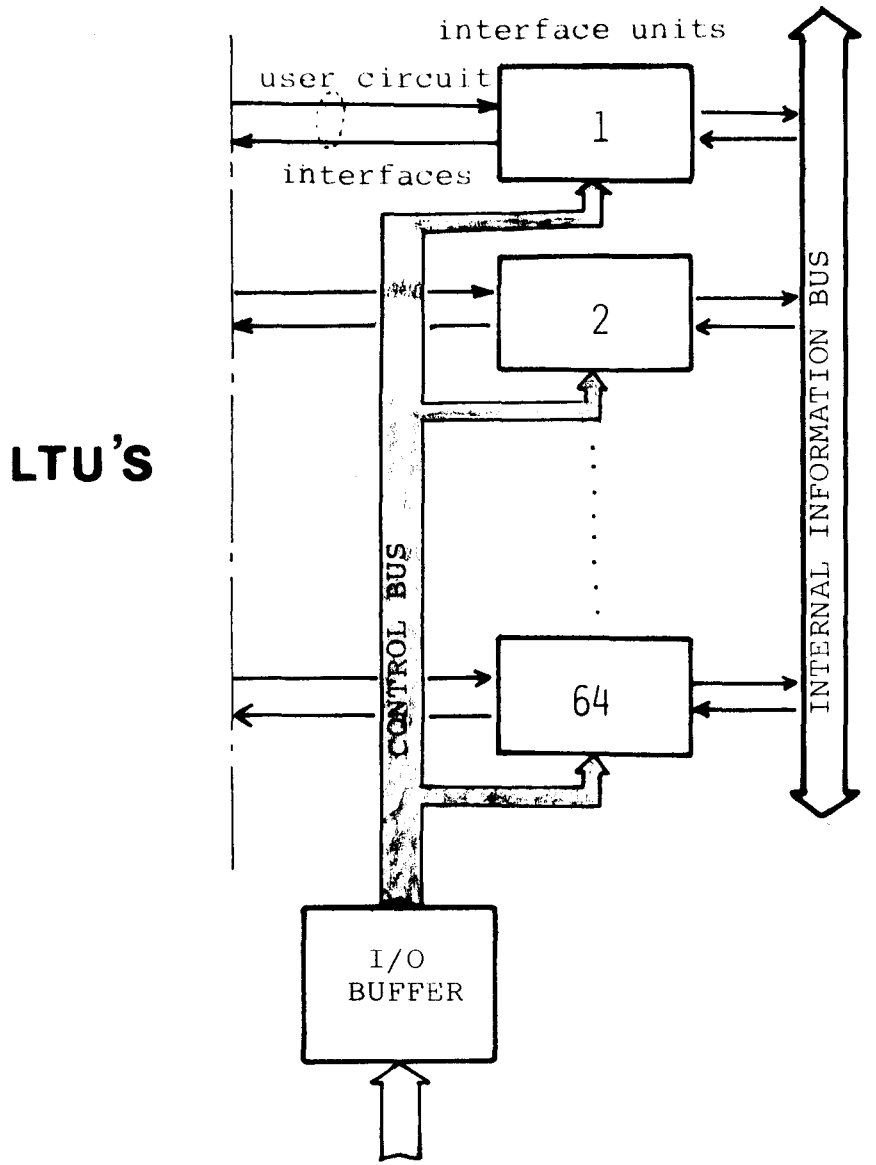


Fig.4.7.a

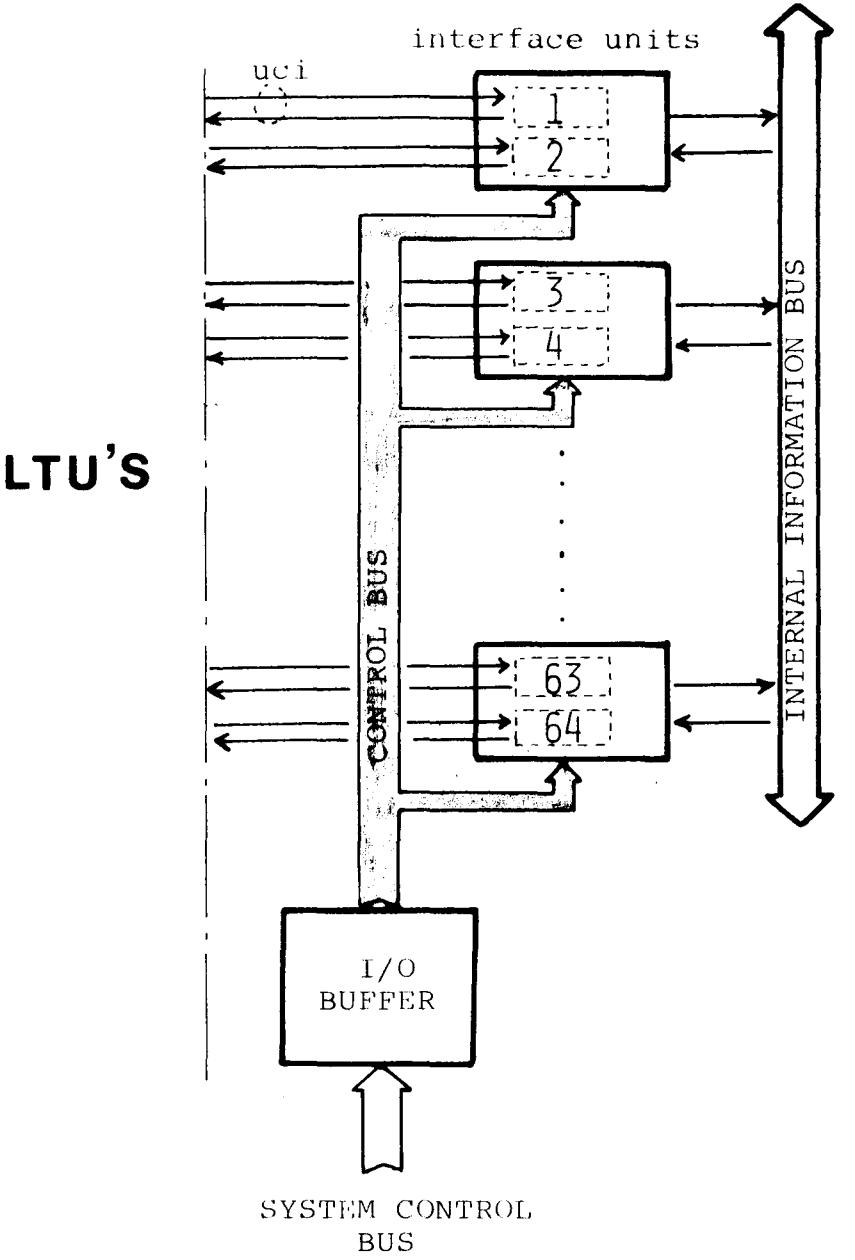


Fig.4.7.b

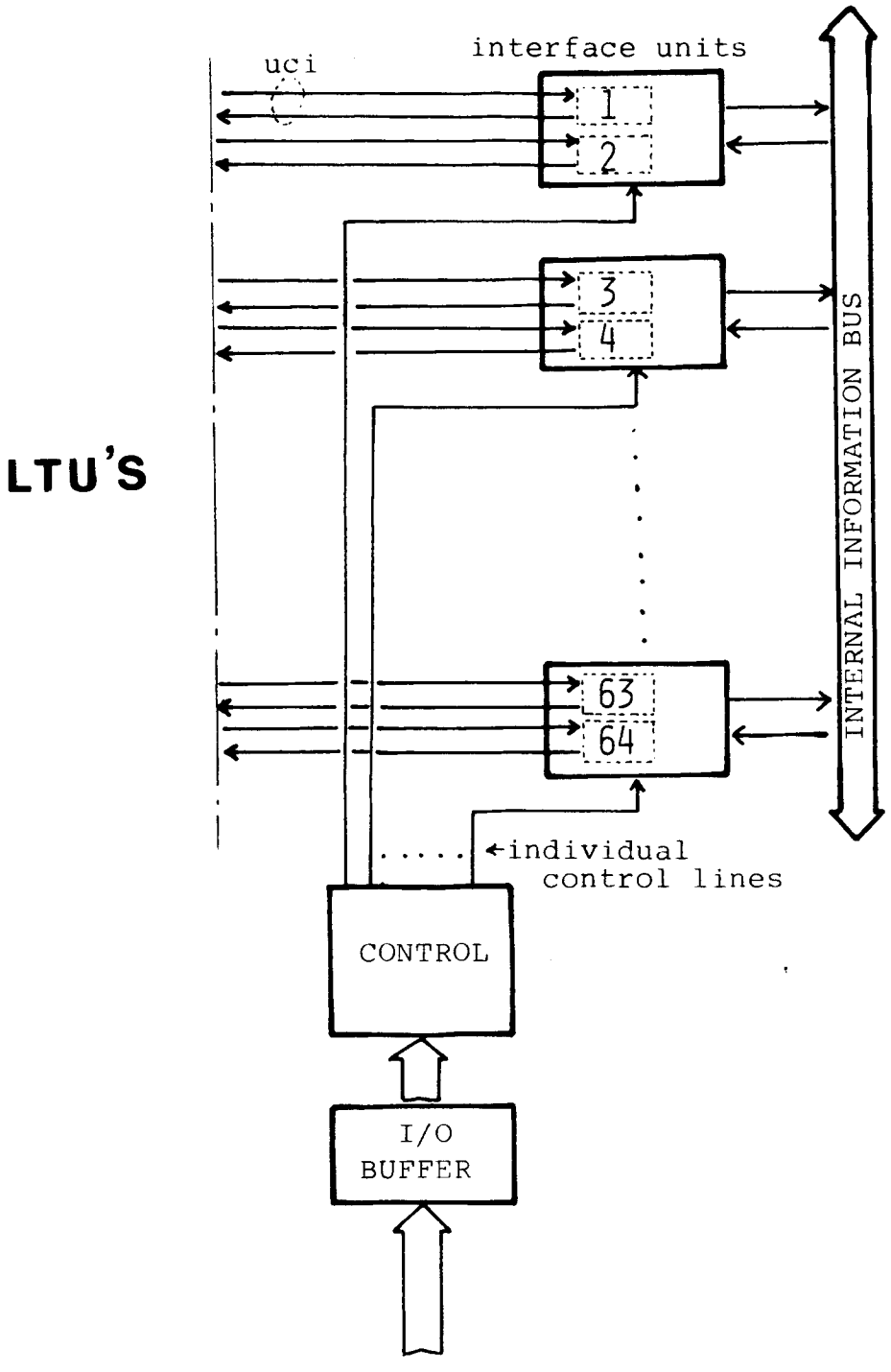

LTU'S
uci
interface units
1
2
3
4
63
64
INTERNAL INFORMATION BUS
individual
control lines
CONTROL
I/O
BUFFER

Fig.4.7.c

Fig. 4.8 gives a global indication of the structure of the control block designed to realise these requirements.

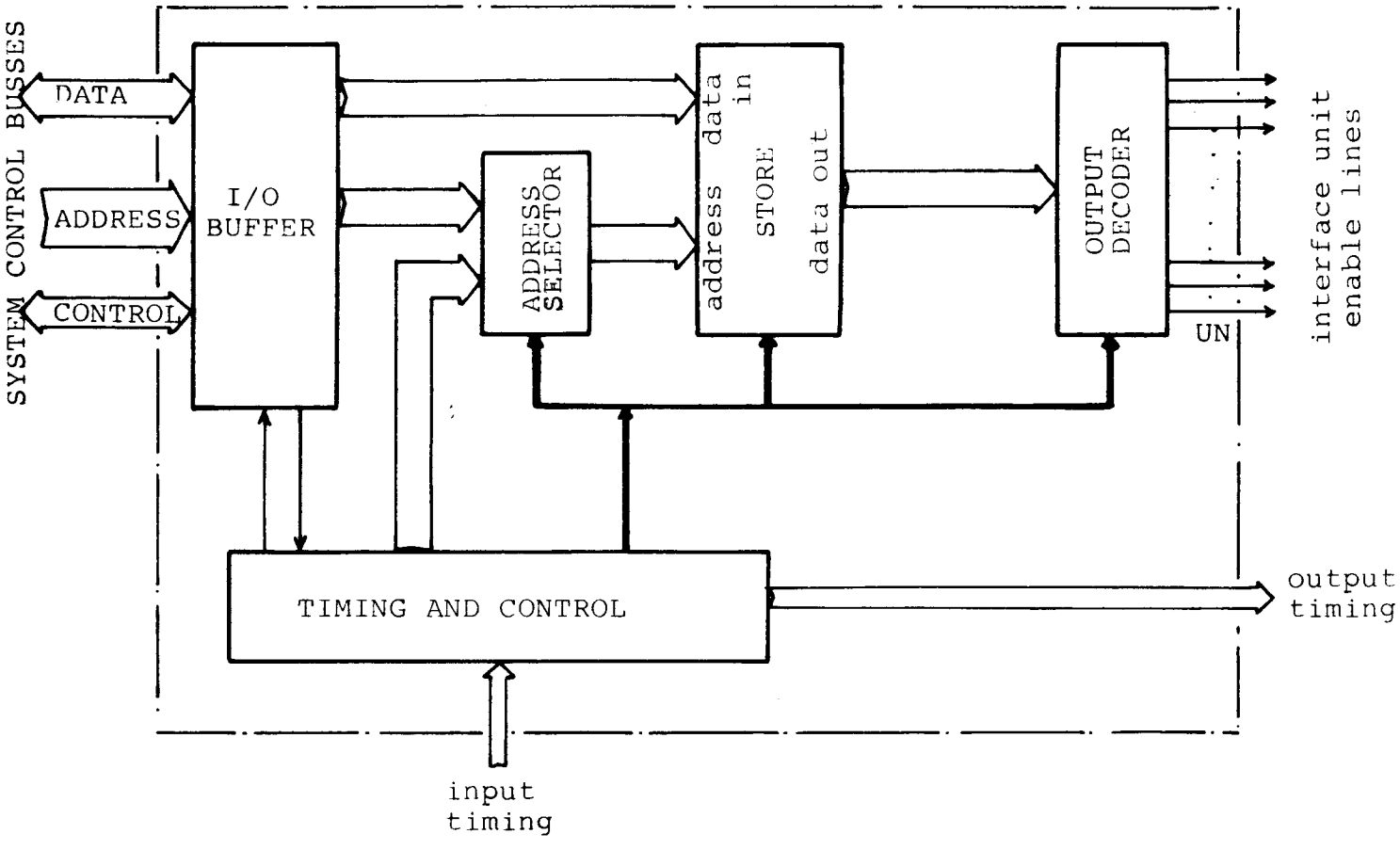


Fig.4.8 CONTROL BLOCK

a) I/O Buffer

Messages are transferred from the Front End Control Function via the I/O buffer, depicted in fig. 4.3. This operates exclusively in the Input Mode (i.e. apart from a status signal no messages are returned to the Control Function).

Preceding a message transfer the buffer status register is scanned.

This indicates whether a previous message is contained within the buffer - in which case no new message would be transferred. A new message entering the buffer causes the 'message ready' line to be activated, and the status register is set to 'busy'. The entry of this message into the store proper causes a 'message transferred' signal to be given which in turn resets the status register to 'free'.
The sixteen bit parallel output of the buffer is, in fact, the message. As has been explained previously this consists of the two parameters shown in fig. 4.9 below.

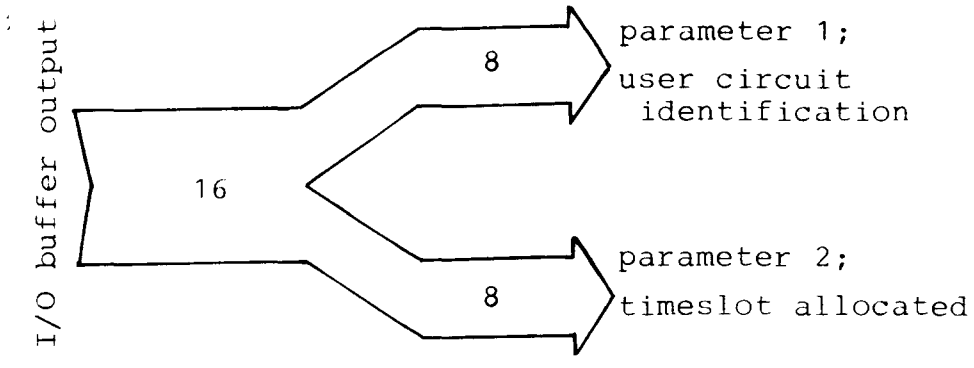


Fig.4.9 BUFFER OUTPUT

The message content here is thus the time slot, on the IIB, which has been allocated to the user circuit specified in the first parameter.
Fig. 4.12 gives a detailed circuit diagram of this unit plus an indication of the timing signals involved.

b) Message Store and Mapping

The messages in the buffer must be transferred to a relatively long term storage unit, as the time slot allocation described by these messages is valid for the whole duration of the connection. This will be referenced once per frame period during the connection.

A Random Access Memory (RAM) is used to both store these messages and to implement the required mapping operation.
The internal control unit supervises, among other things, this message transfer between buffer and RAM.

The actual mapping is accomplished through the use of a 32 byte RAM in which each memory location (0 to 31) is associated directly with a specific time slot number within a frame on the IIB (these are also numbered 0 to 31). The contents of a given memory location is the identification of that user circuit which is to be connected to the IIB during the time slot associated with the location address:

e.g. memory location 18 is addressed - this indicates that the user circuit identified by the contents of this location,is to be connected to the IIB during time slot 18 of each frame.

Causing this RAM to be read cyclically (modulo 32) gives as output, at each cycle, the identification of the required interface unit.
Fig. 4.10 gives a view of the typical contents of this RAM at an instant in time.

ADDRESS	RAM CONTENTS	EQUIVALENT TIMESLOT
00000	uc 12	0
00001	uc 48	1
00010	uc UN	2
.	.	.
.	.	.
.	.	.
10001	uc 3	17
10010	uc UN	18
.	.	.
.	.	.
.	.	.
11110	uc 16	30
11111	dedicated IP	31

uc=user circuit
IP=Inter Processr signalling

Fig.4.10 RAM MAP

The RAM is addressed by a cyclic counter, which is incremented each time slot, giving the required output.
This read cycle access time,plus other delays in the system circuitry, mean that measures must be taken to ensure that the required interface unit is activated at the appropriate time, i.e. at the start of the corresponding time slot. The problem is overcome by reading the RAM output at a point in time preceding the occurrance of the time slot, latching this information, and sampling it at the start of the time slot. The RAM address corresponding to the next time slot (and not the present) is therefore read at each cycle.
In order to facilitate the alteration of the RAM contents the time slot period is split into two regions.

One is used to carry out the read cycle described above,while the second is used to write the new messages into the RAM from the I/O buffer. In this write cycle the 'time slot number' parameter of the message is used to address the RAM at that location into which the second parameter - the user circuit allocated - is to be read. An address selector, as seen in fig. 4.8, is used to select the required address during a given cycle.
The detailed circuit diagram of a realisation of the above is given in fig. 4.13.

c) Output Circuit

The RAM output is thus latched at a time preceding the start of the time slot to which it applies. This output is then decoded to form an enable signal to the specified interface unit.
This enable signal is transferred to the interface unit via two signal lines, as shown in fig. 4.8. The first of these is common to all units and distinguishes between the two interface units in a duplet (see fig. 4.7.c). The second, which is in the form of a dedicated output line to each interface block, selects the appropriate block. A time slot clock line is also common to these blocks. This indicates the beginning of a time slot, at which time the above enable lines are sampled by the interface units.
The output decoding is carried out in the control block as this reduces the hardware necessary in the interface units . The disadvantage is, of course, the number of output lines then necessary to transfer the decoded information to the units.

Fig. 4.14 gives the detailed circuit diagram of this unit.

d) Timing and internal control

This unit must create the various timing signals mentioned in the previous sections plus the control signals necessary to support the units discussed. The input consists of the 2048 kHz system clock plus a synchronisation signal (RESET) which is used to reset the timing units at the start of operation,or after breakdown of the system. Fig. 4.11 gives the basic timing signals created and an indication as to their use, while fig. 4.15 gives a detailed circuit diagram of the unit and a description of its operation.

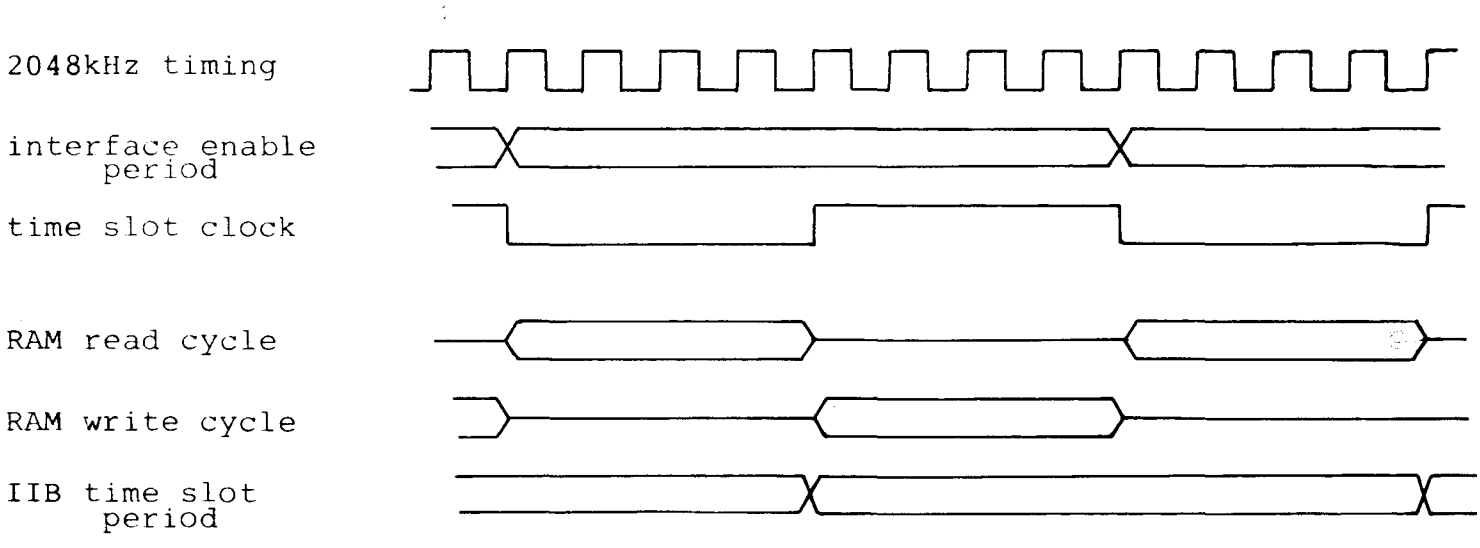


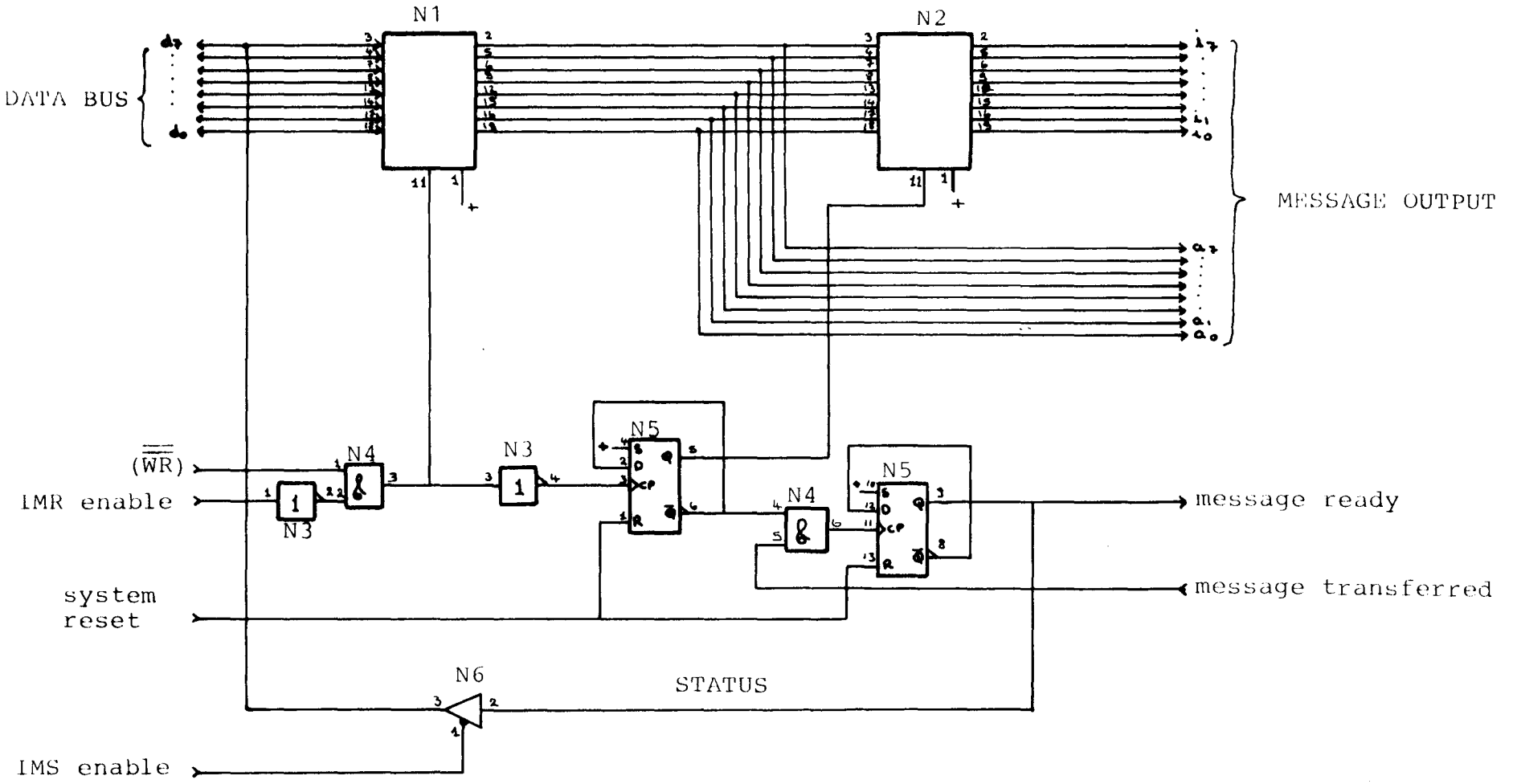
Fig.4.11 BASIC TIMING SIGNALS

Component List: Figs. 4.12 to 4.15

N1, N2, N11	: 74LS 273
N3, N19, N25	: 74LS 04
N4, N24	: 74LS 08
N5, N20, N21, N22	: 74LS 74
N6	: 74 LS 125
N7, N8	: 74LS 157
N9, N10	: HM 6551-9 (RAM 256 x 4)
N12, N13, N14, N15, N16	: 74LS 138
N17, N18	: 74LS 193
N23	: 74LS 21
N26, N27	: 74LS 11
N28	: 74LS 02

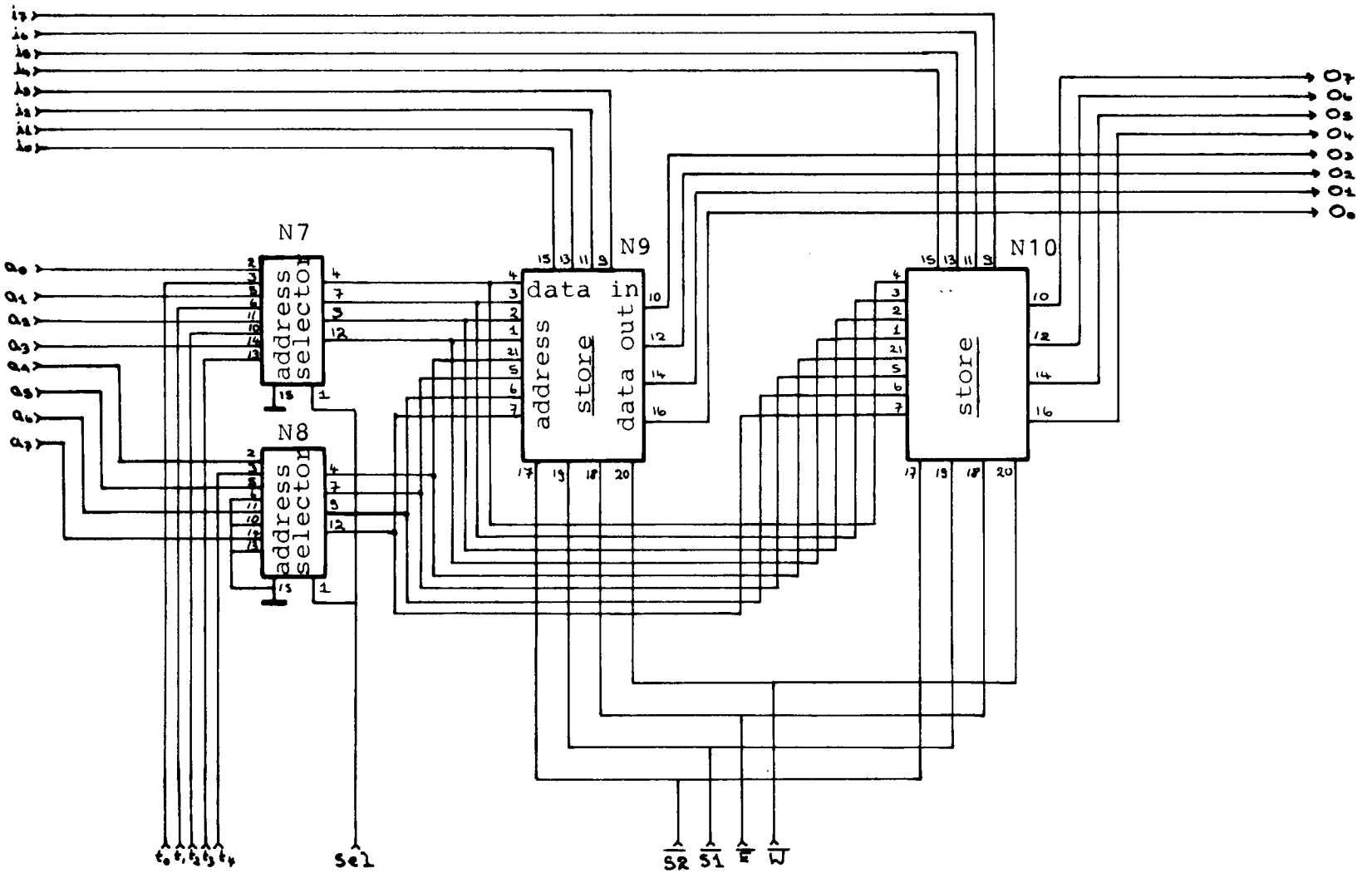
List of abbreviations

$\overline{E}$; $\overline{W}$; $\overline{S1}$; $\overline{S2}$	: See HM 6551 specification sheets
Φ	: 2048 kHz system timing
IMR enable	: Interface Management Register enable signal
IMS enable	: Interface Management Status enable signal
E^{CM}_{out} (UE)	: Common output enable signal
E^{n}_{out} (EE)	: Output enable signal to specific extension number, n, $0 < n < 31$
IP	: Output enable signal to Inter processor signalling unit
UN	: Unallocated timeslot indication
$t_0 \ldots t_4$	: Coded representation of next timeslot number
Sel	: Select signal (input/$\overline{\text{output}}$ address)
k_{64}	: 64 kHz timing
k_T	: Timeslot clock
Sp	: RAM output sample timing



a) CIRCUIT DIAGRAM

Fig.4.12 I/O BUFFER



a) CIRCUIT DIAGRAM

Fig.4.13 MESSAGE STORE

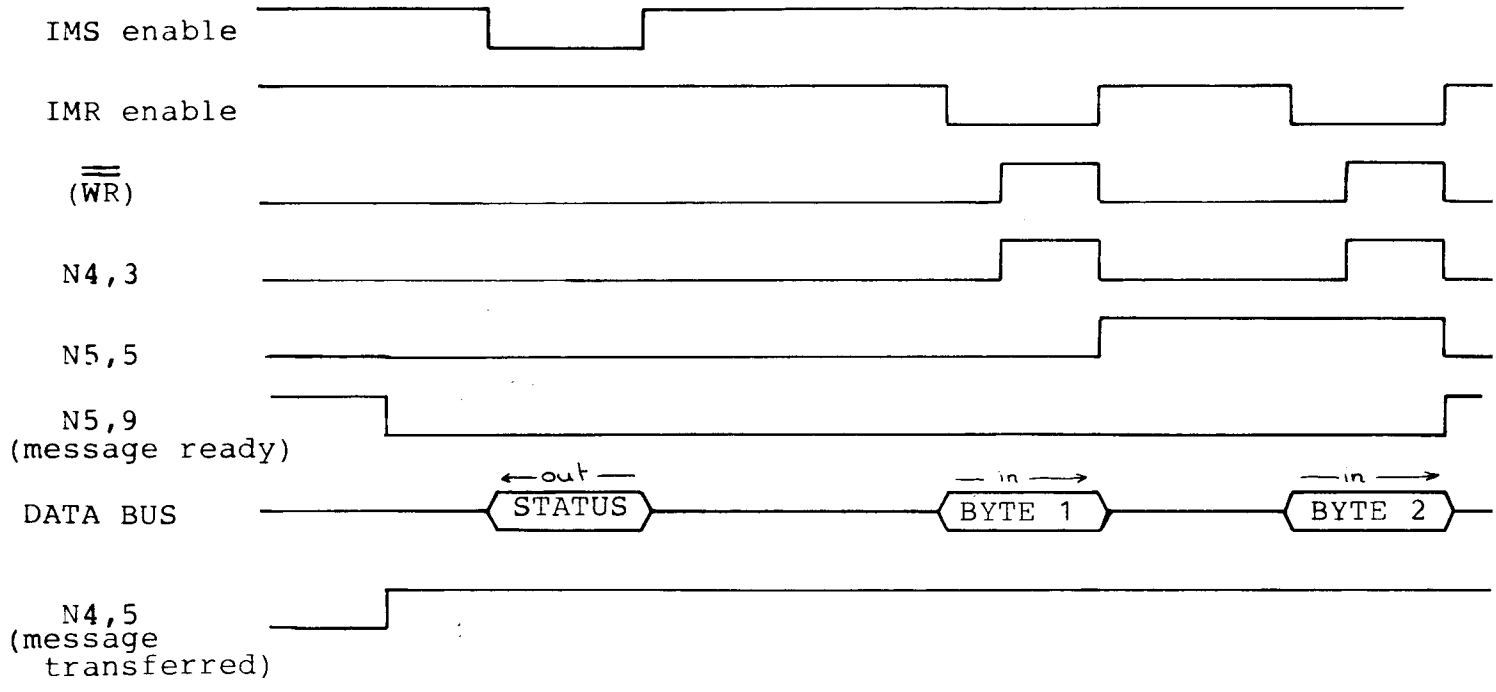


Fig.4.12 b) TIMING DIAGRAM

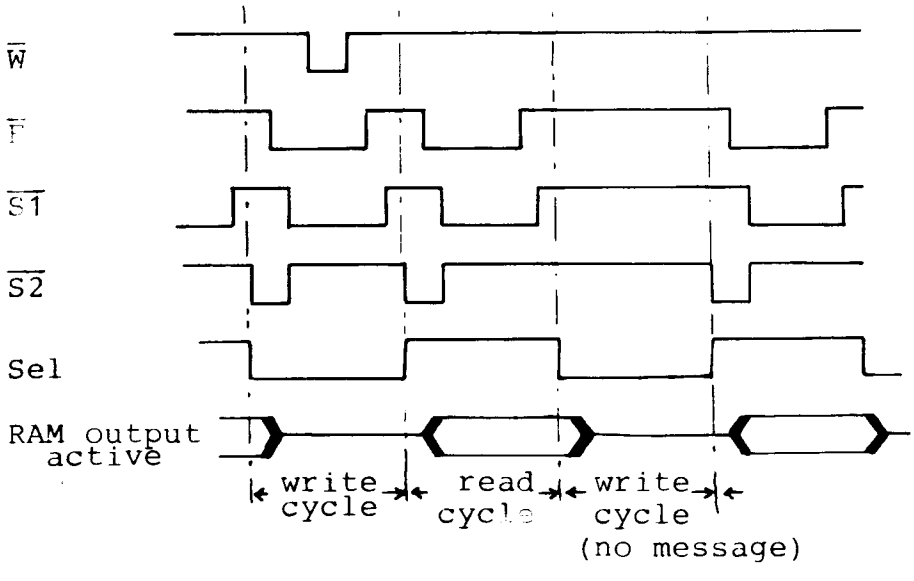
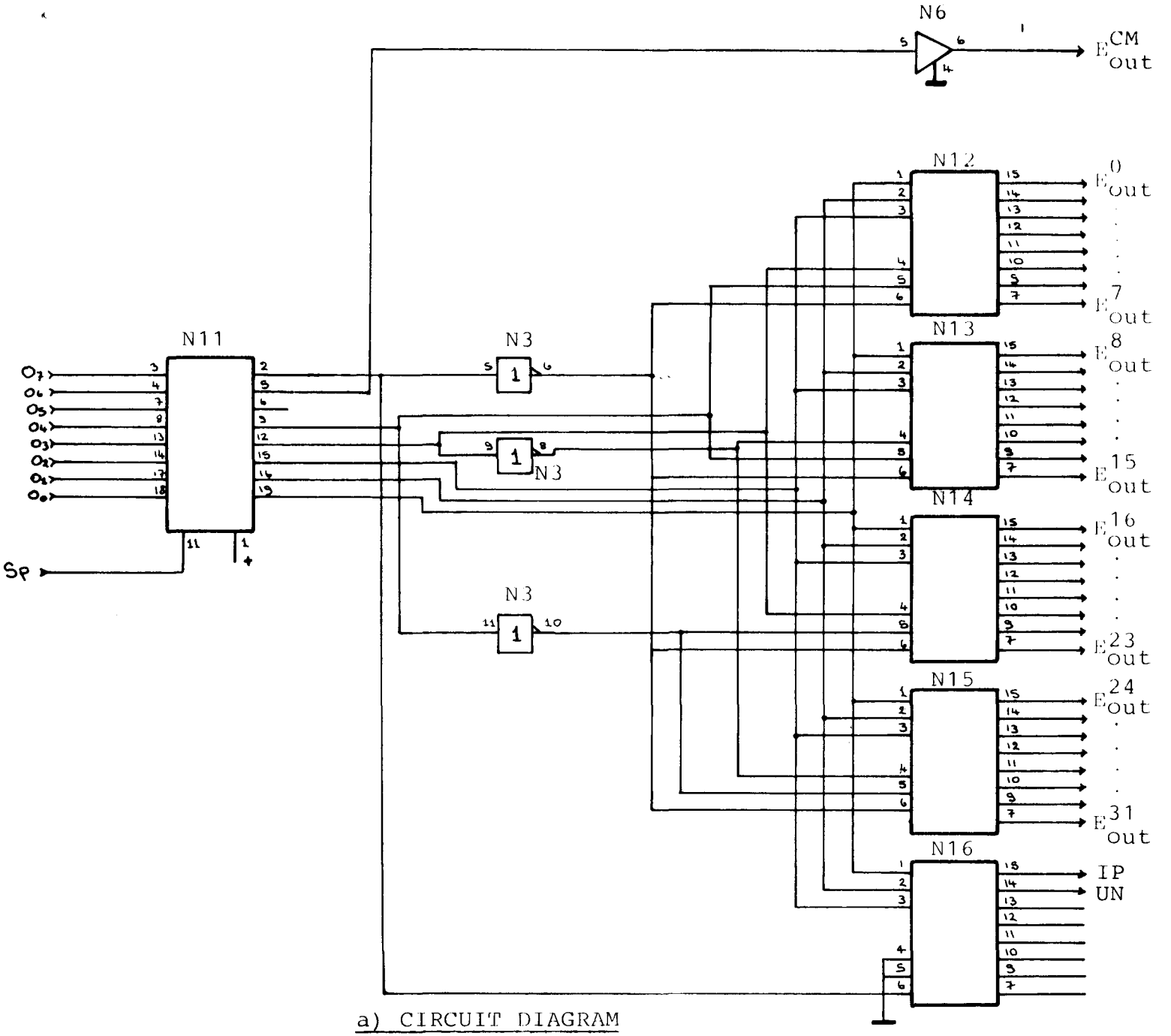
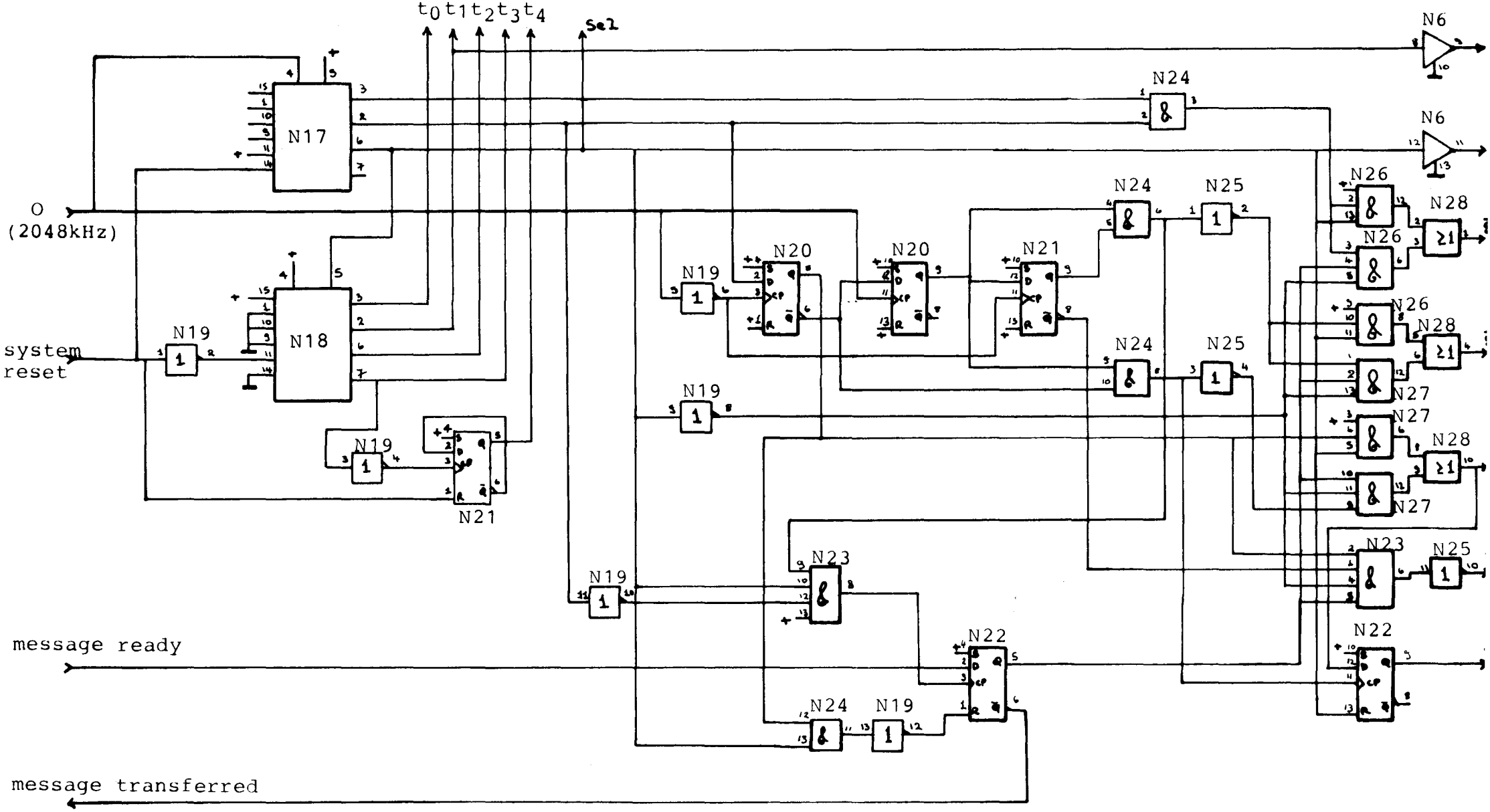


Fig.4.13 b) TIMING DIAGRAM



a) CIRCUIT DIAGRAM

Fig.4.14 OUTPUT UNIT



a) CIRCUIT DIAGRAM

Fig.4.15 CONTROL-TIMING UNIT

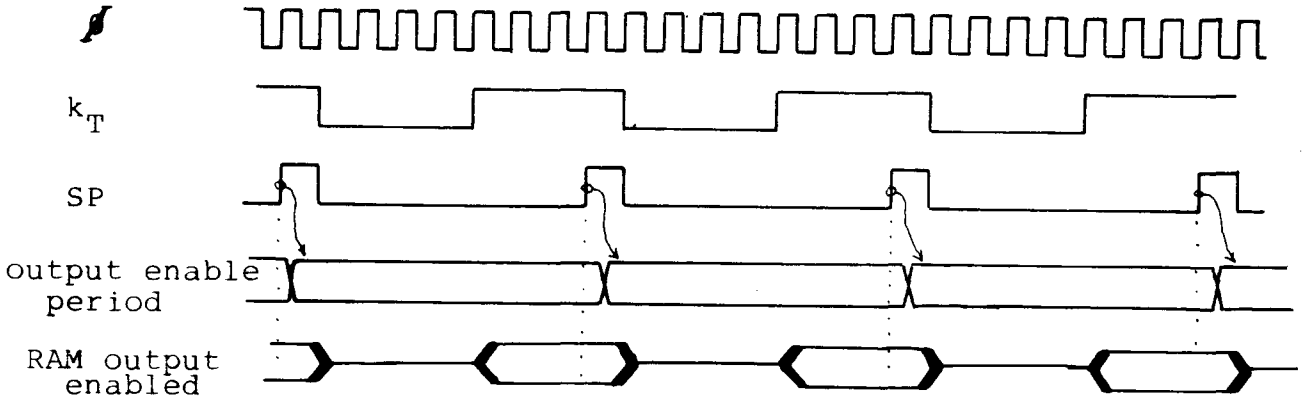


Fig.4.14 b) TIMING DIAGRAM

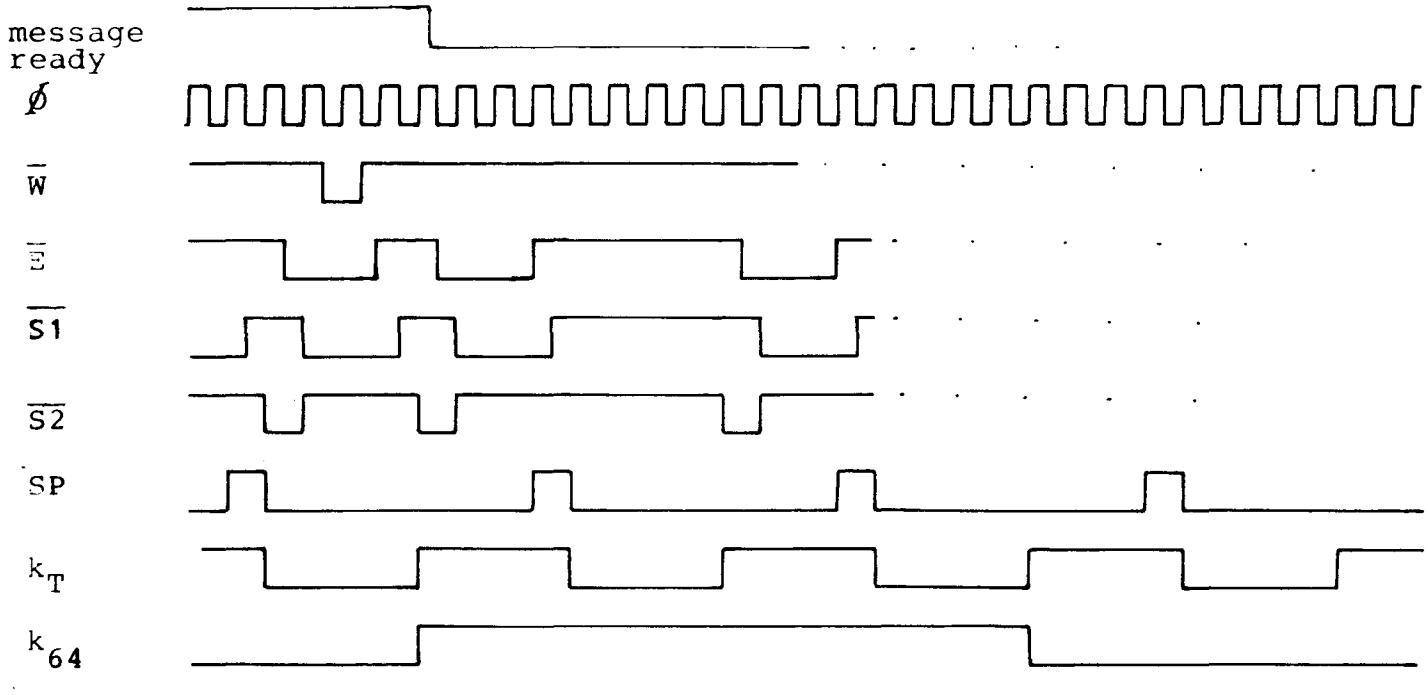


Fig.4.15 b) TIMING DIAGRAM

N11								UNIT ENABLED		
O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0	E^n_{out}	User	Extension
0	X	0	0	0	0	0	0	E^0	1	0
0	X	1	0	0	0	0	0	E^0	2	
0	X	0	0	0	0	0	1	E^1	1	1
0	X	1	0	0	0	0	1	E^1	2	
0	X	0	0	0	0	1	0	E^2	1	2
0	X	1	0	0	0	1	0	E^2	2	
.									.	.
.									.	.
.									.	.
.									.	.
0	X	0	1	0	0	0	0	E^{16}	1	16
0	X	1	1	0	0	0	0	E^{16}	2	
.									.	.
.									.	.
.									.	.
.									.	.
0	X	0	1	1	1	1	0	E^{30}	1	30
0	X	1	1	1	1	1	0	E^{30}	2	
0	X	0	1	1	1	1	1	E^{31}	1	31
0	X	1	1	1	1	1	1	E^{31}	2	
1	X	X	X	X	0	0	0	IP	Inter processor Signalling cct.	
1	X	X	X	X	0	0	1	UN	not allocated	

INTERFACE CONTROLLER

user number (d_6, d_5)	unit	E^{CM}
00	common equipment	-
01	MUC (user 1)	0
10	AUC (user 2)	1
11	telemetry unit	-

EXTENSION

Fig.5.14.c OUTPUT MAP

A RAM entry corresponding to an unallocated time slot, is filled with a pattern indicating an unused output enable line (UN in fig. 4.8). The de-allocation of a time slot at the end of a connection will then consist of a message from the Control Function in which the 'user circuit' parameter contains this pattern. This pattern will be equivalent to user number .
As is seen in fig. 4.10 time slot 31 is dedicated to the inter processor signalling unit. The actual use to which this is put will be explained in chapter 5.

The control block is thus functionally complete.

4.3.2. Interface Units

These implement the actual connection between LTU's and IIB for each of the user circuits. Their global configuration has already been considered to some extent in sections 4.1 and 4.2.
For reasons explained in these previous sections, namely the differences in format of the user circuit interface and the IIB interface, a direct connection is impossible. This leads to the use of various buffers in the interface units in order to control the information transfer. These buffers are driven by the output signals from the control block via an internal slave control function in the interface units.
Fig. 4.16 gives a block diagram of the fundamental interface unit structure.

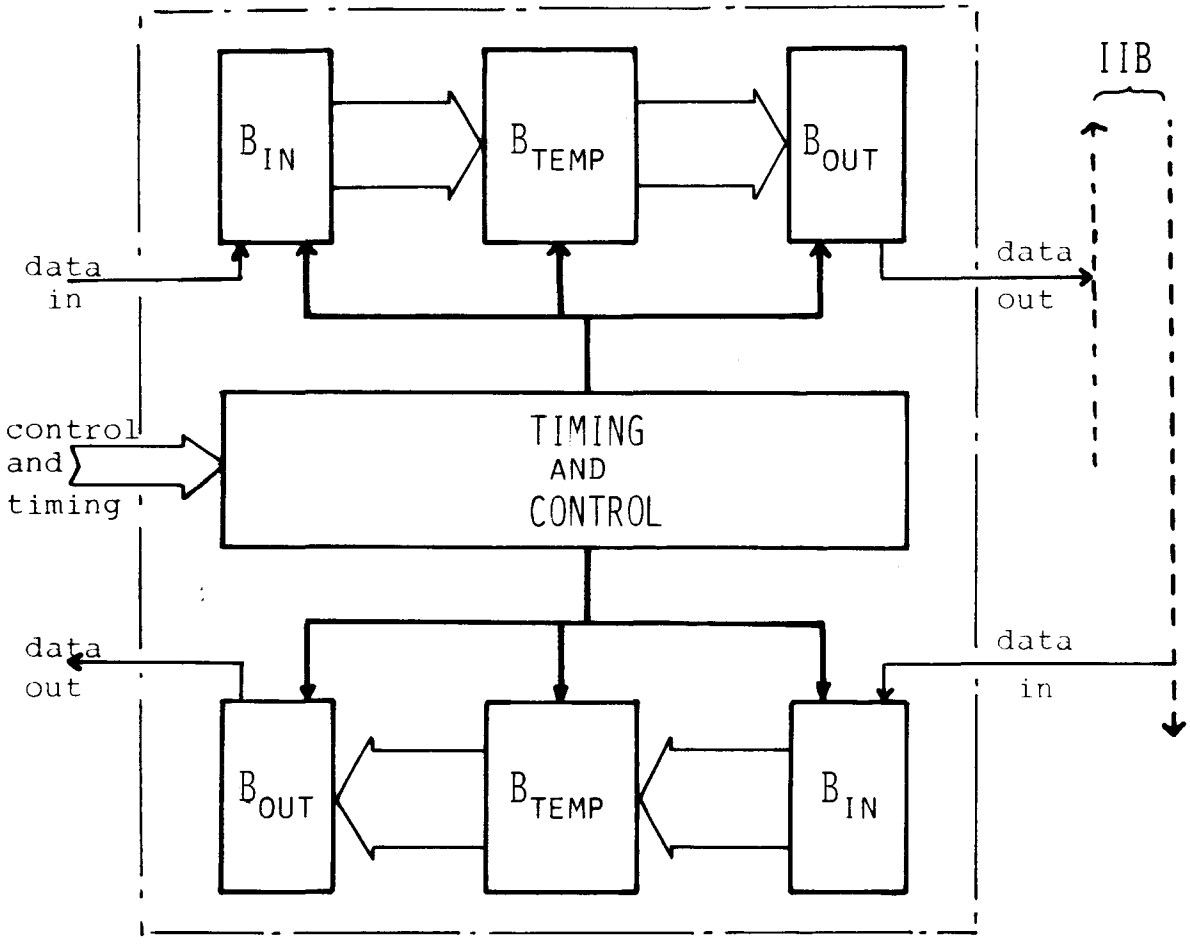


Fig.4.16 INTERFACE UNIT STRUCTURE

Here input data from the user circuit interface is read serially into an input buffer, B_{IN}. When full the contents are transferred to a temporary storage area, B_{TEMP}, where it is held until the present contents of the output buffer, B_{OUT}, have been written onto the IIB. At this time B_{OUT} is loaded with the contents of B_{TEMP}. This process is repeated cyclically. In the opposite direction, i.e. bus to user circuit interface, the mirror image of the above process occurs as shown. The required connection has thus been created.

As described previously the interface units for both user circuits from an extension are combined at this stage, leading to the configuration of fig. 4.17.

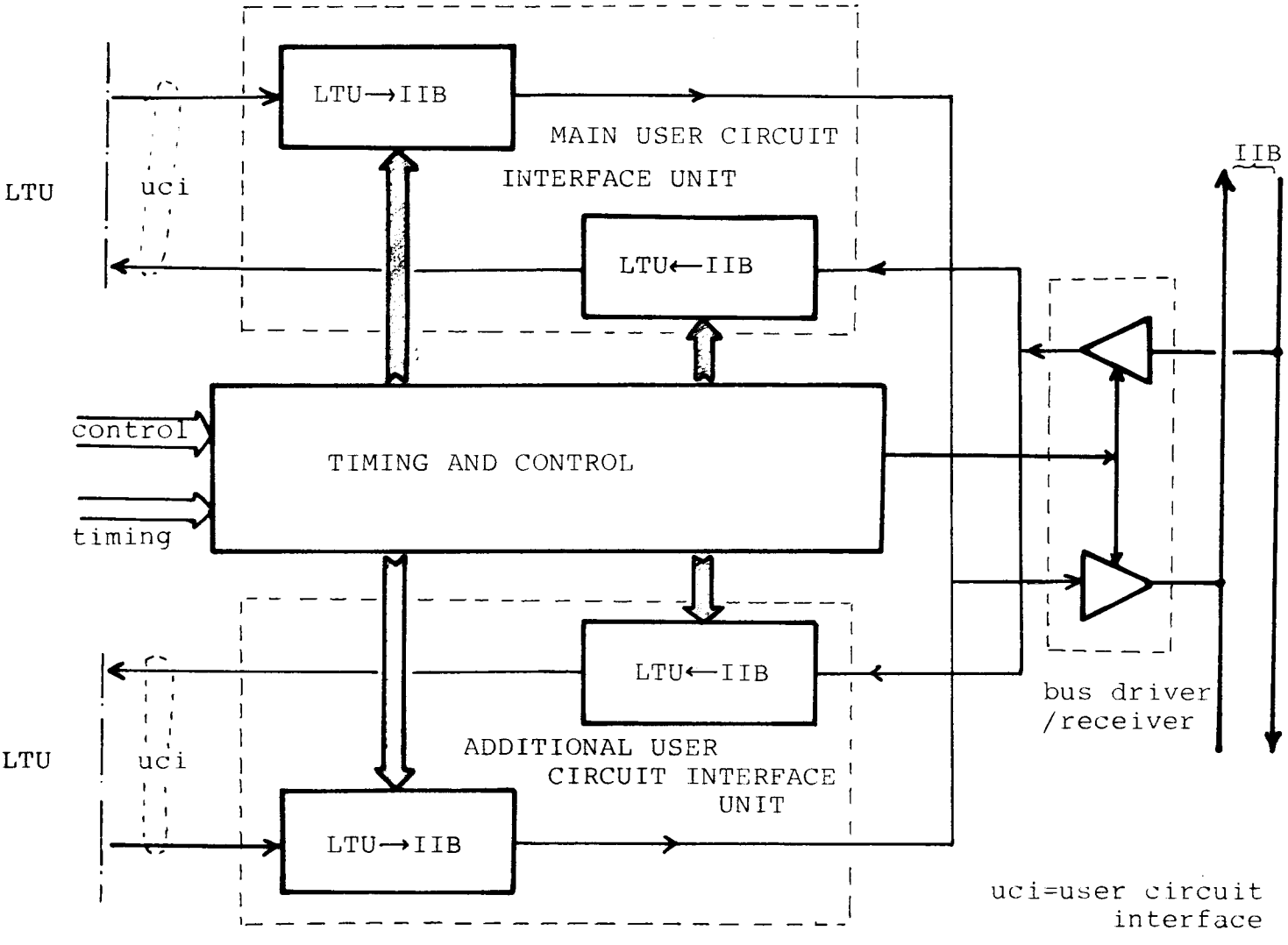


Fig.4.17 COMBINED INTERFACE UNITS

As is seen from this figure the bus driver/receiver is common to both circuits. This is possible as only a single circuit has access to the IIB during any given time slot. A pre-selection is therefore made as to which of the two circuits is active and this is then allotted the driver/receiver.

Figures 4.18 to 4.22 give the proposed hardware implementation of the various elements of this multiple interface unit along with timing diagrams of their operation.

Each of these multiple units, 32 in total, is connected in parallel with the IIB, leading to the required system configuration.

Component List: Figs. 4.18 to 4.22

N1, N4, N18	: 74LS 164
N2, N5	: 74LS 374
N3, N6, N16	: 74LS 165
N7, N9, N11, N12, N13, N15, N17, N19, N22, N23	: 74LS 74
N8, N29	: 74LS 04
N10, N20	: 74LS 00
N14, N21	: 74LS 194A
N24	: 74LS 02
N25, N26	: 74LS 08
N27	: 74LS 32
N28	: 74LS 125

Abbreviations used in figs. 4.18 to 4.23

Main_User_Circuit

d_{in}^{64} : 64 kb/sec. data input
k_{in}^{64} : 64 kHz timing
k_{in}^{64} : 8 kHz timing
} from LTU

d_{out}^{64} : 64 kb/sec. data output
k_{out}^{64} : 64 kHz timing
k_{out}^{1} : 8 kHz timing
} to LTU

D_{out}^{64} : 2048 kb/sec. data output burst } to Bus
D_{IN} : 2048 kb/sec data input } from Bus

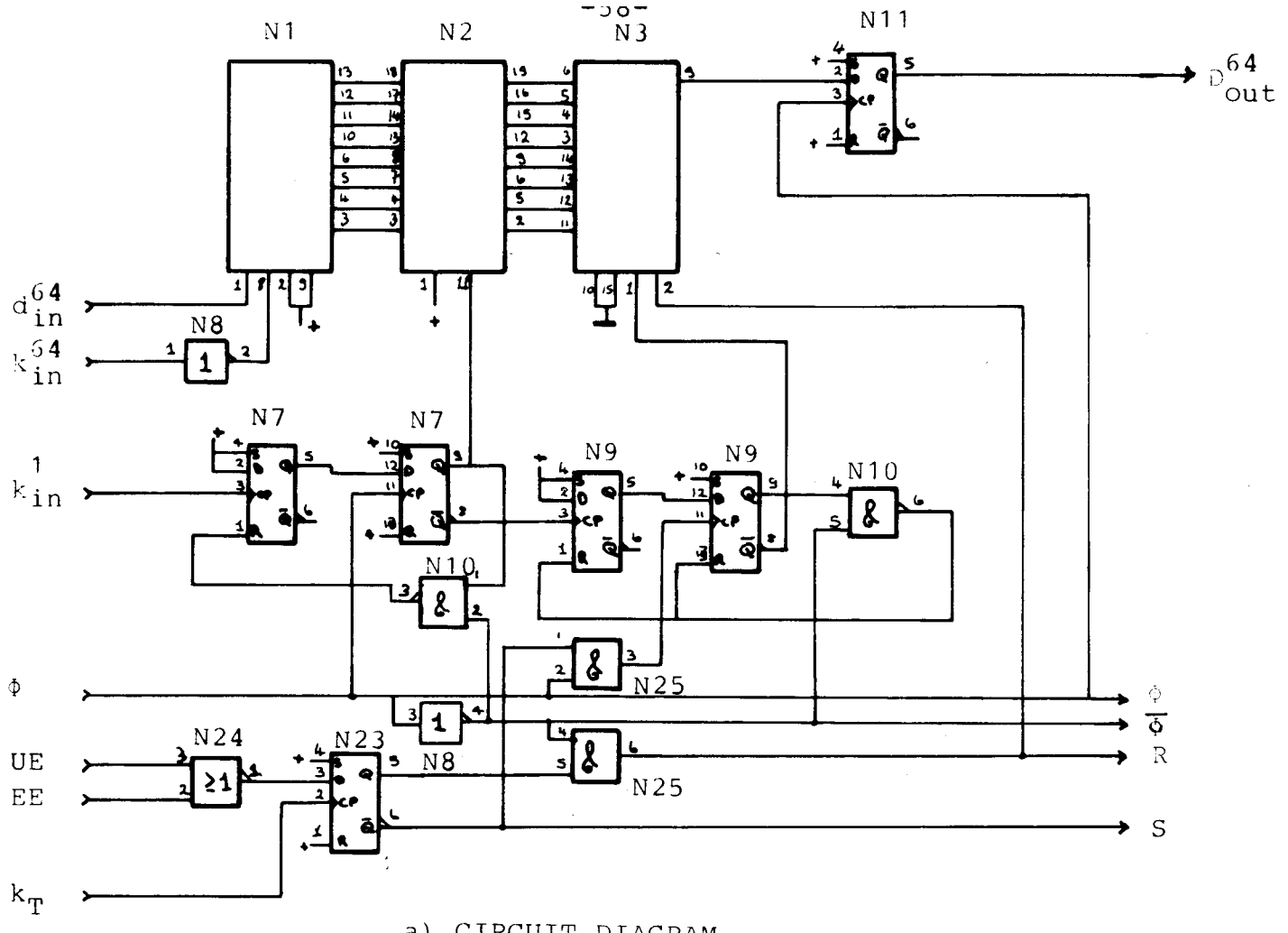
Additional_User_Circuit

d_{in}^{8} : 8 kb/sec. data input
k_{in}^{8} : 8 kHz timing
} from LTU

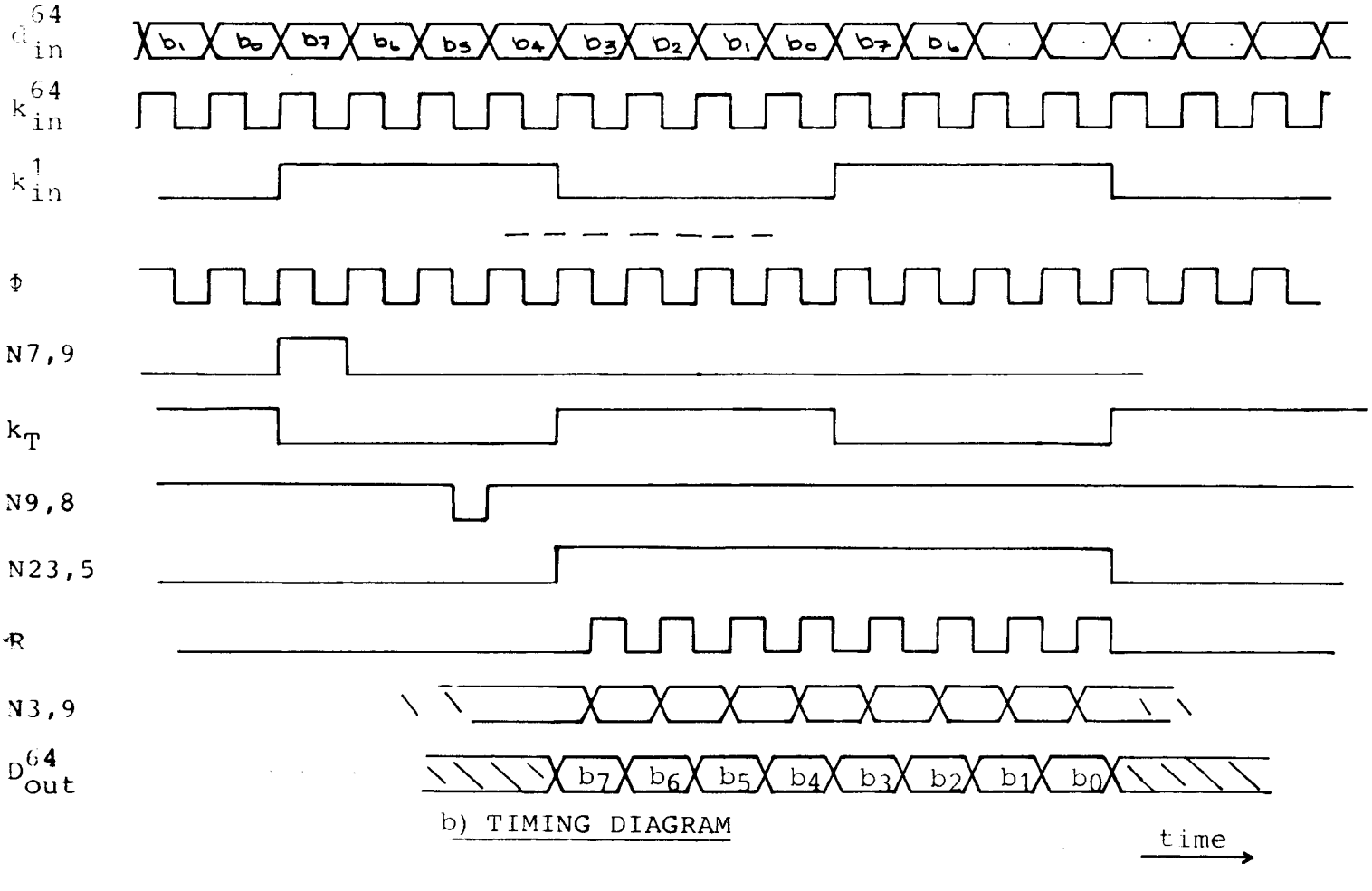
d_{out}^{8} : 8 kb/sec. data output
k_{out}^{8} : 8 kHz timing
} to LTU

D_{out}^{8} : 2048 kb/sec. data output burst } to Bus
D_{IN} : 2048 kb/sec data input } from Bus

Φ : 2048 kHz timing
k_T : timeslot clock
k_{64} : 64 kHz timing
UE : user enable signal
EE : extension enable signal
} from concentrated control unit of the Interface Management

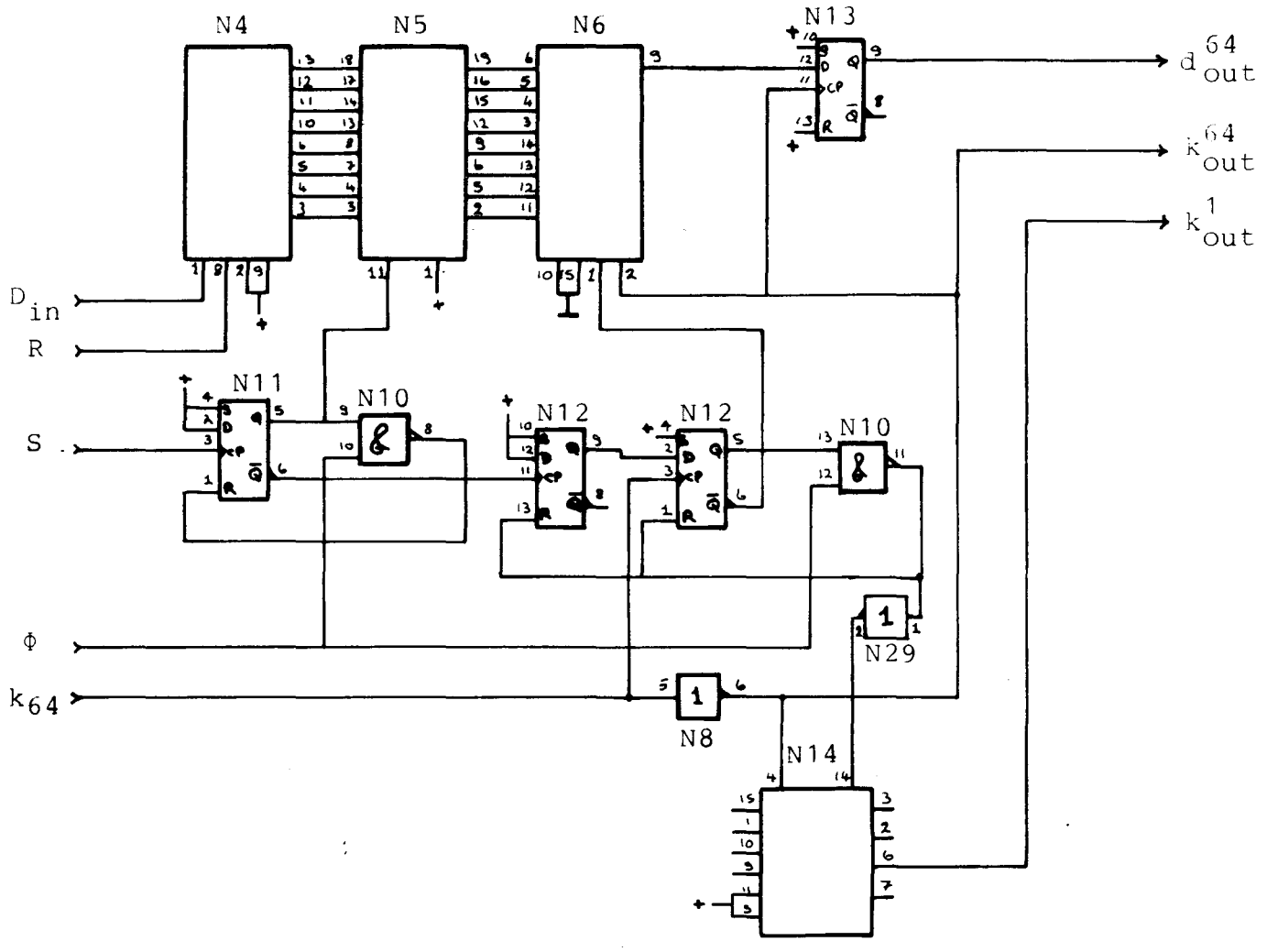


a) CIRCUIT DIAGRAM

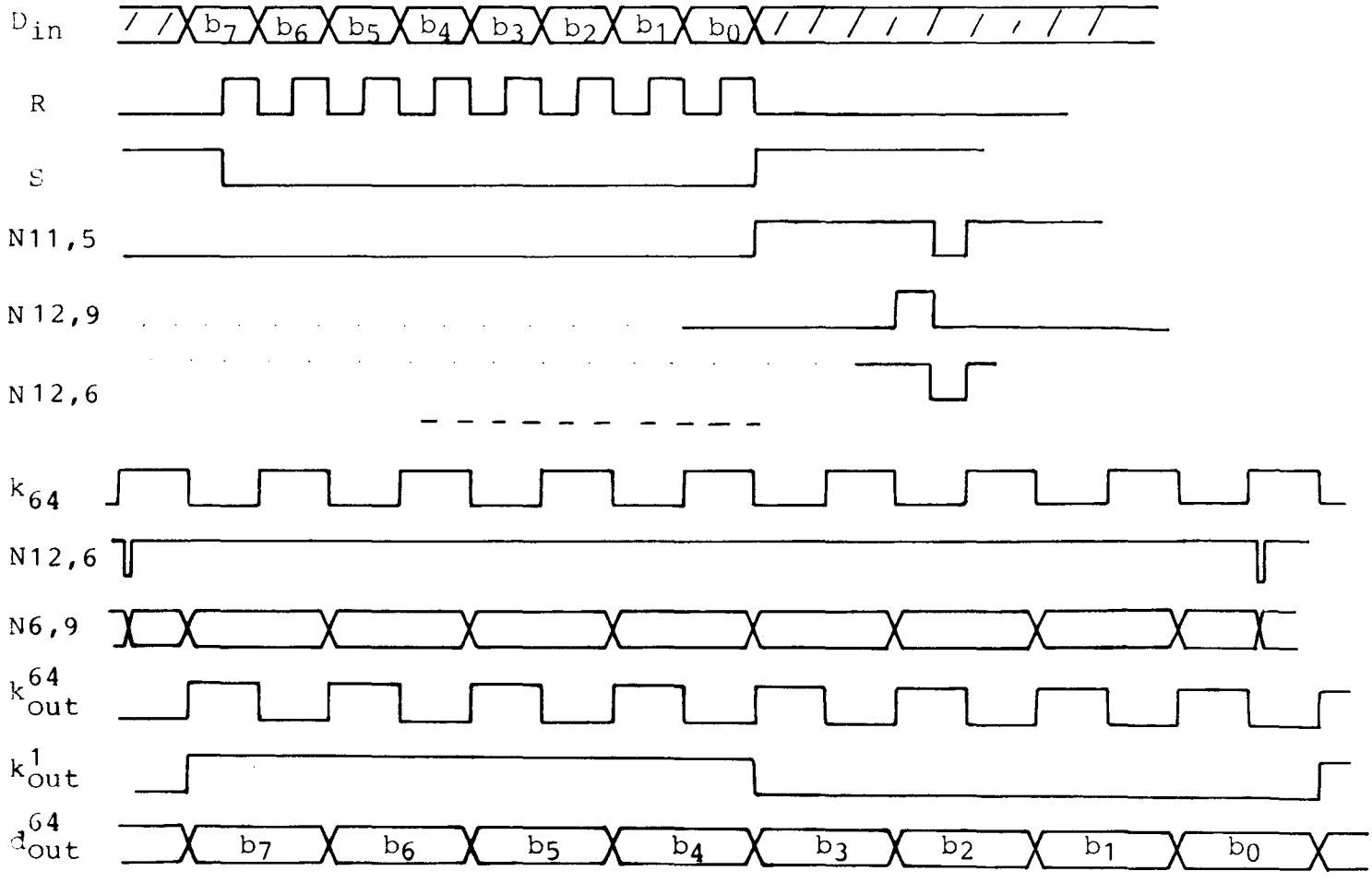


b) TIMING DIAGRAM

Fig.4.18 MAIN USER CIRCUIT:LTU→BUS

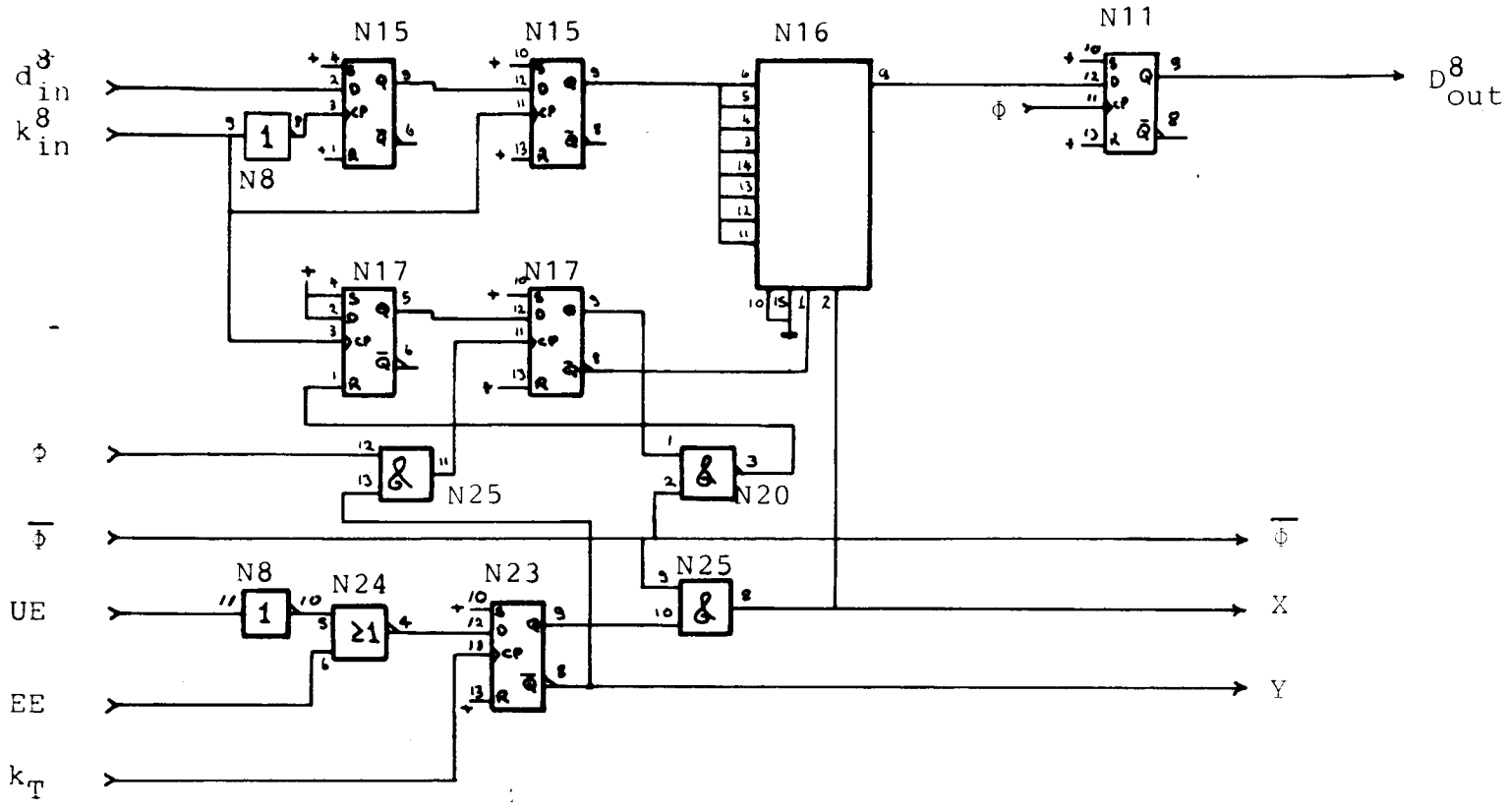


a) CIRCUIT DIAGRAM

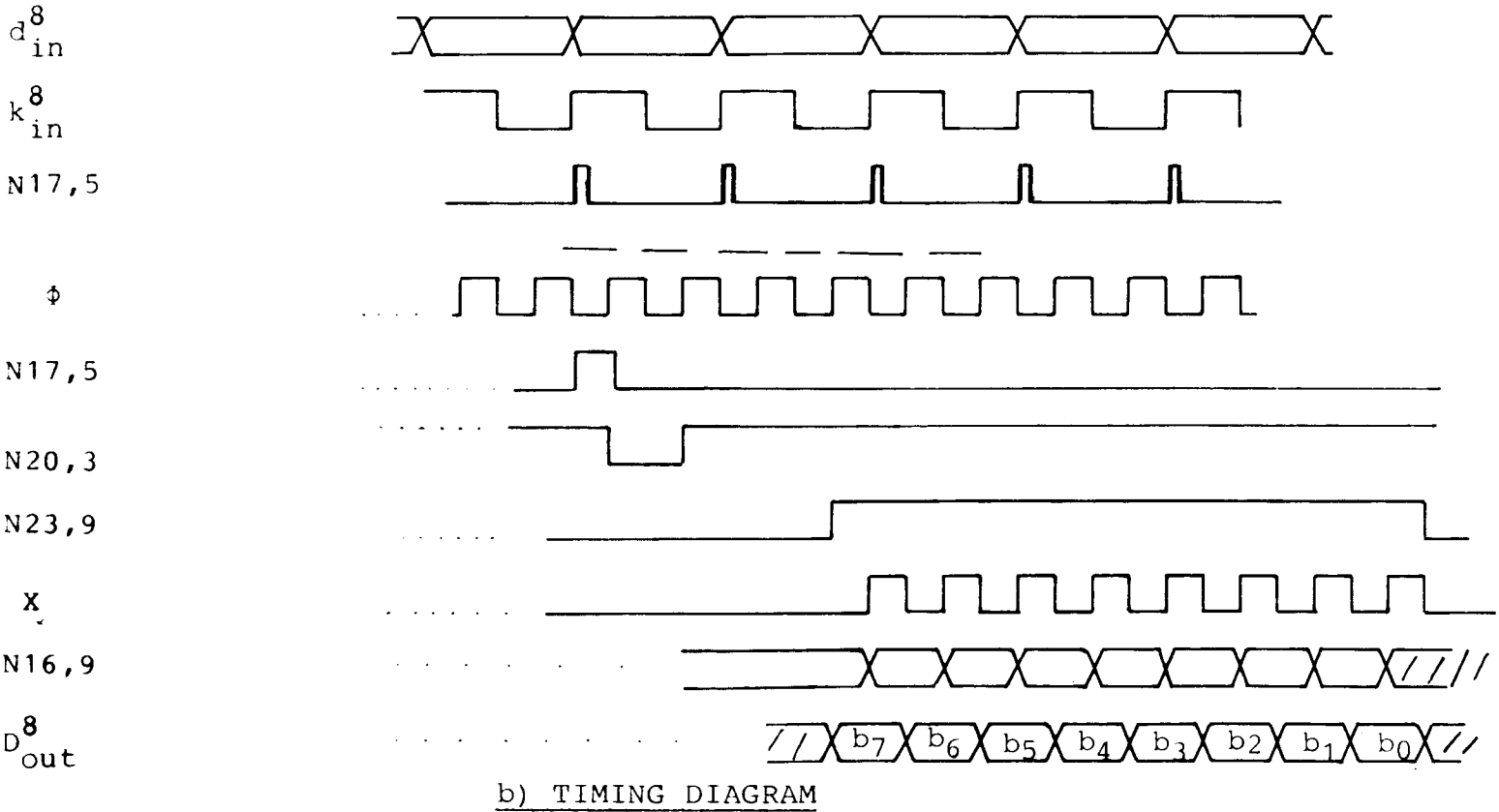


b) TIMING DIAGRAM

Fig.4.19 MAIN USER CIRCUIT:BUS→LTU

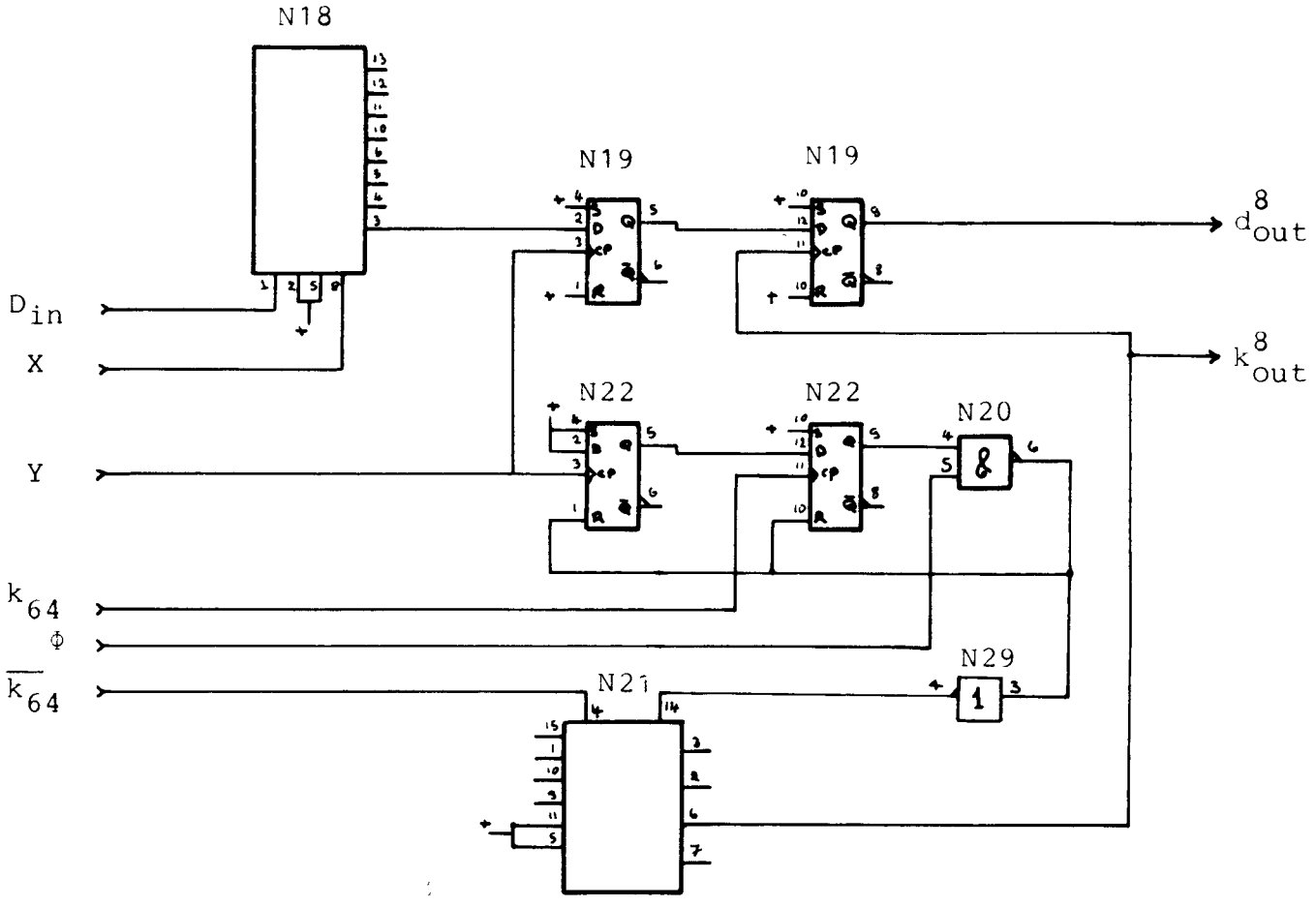


a) CIRCUIT DIAGRAM

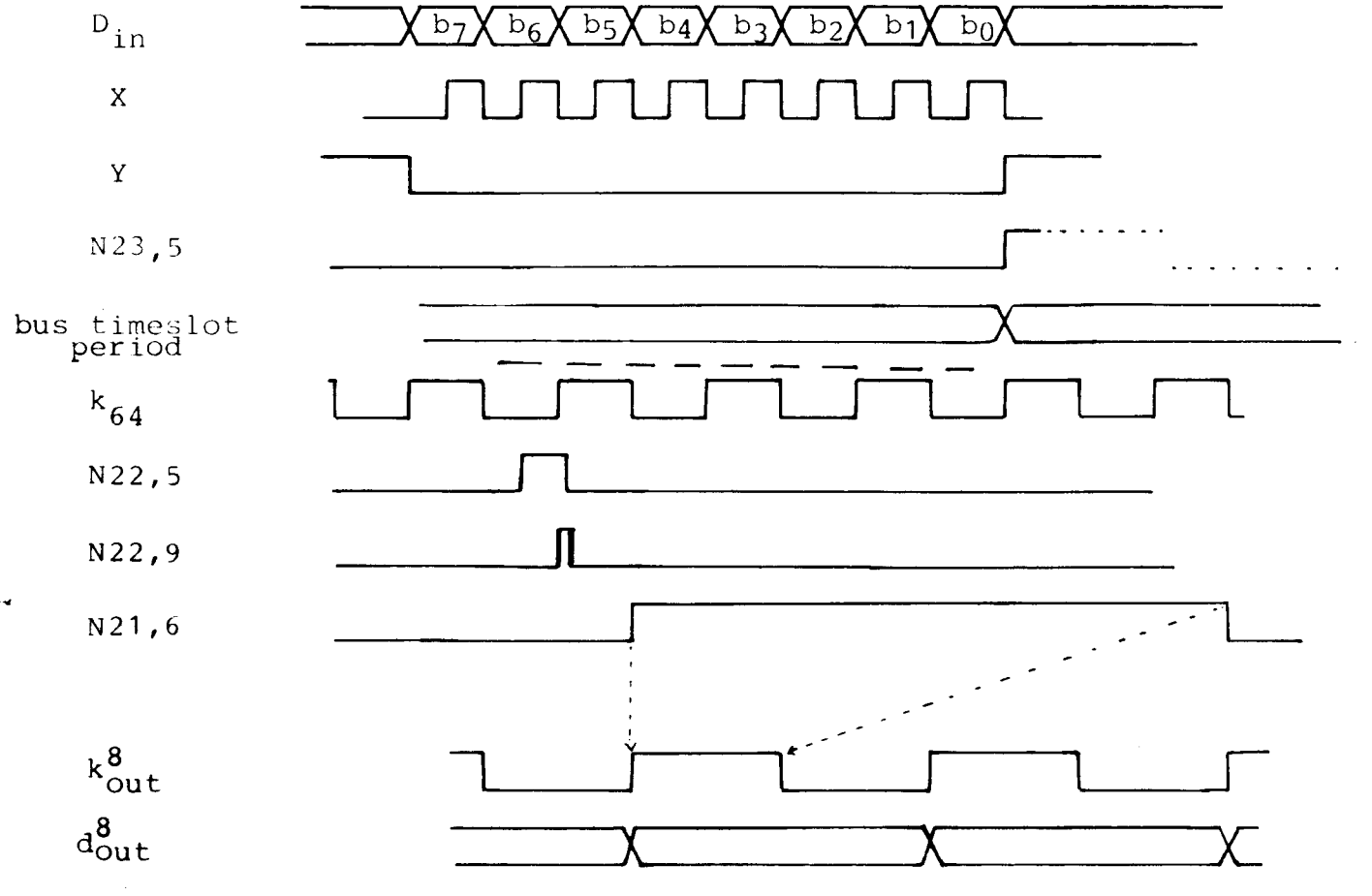


b) TIMING DIAGRAM

Fig.4.20 ADDITIONAL USER CIRCUIT:LTU→IIB

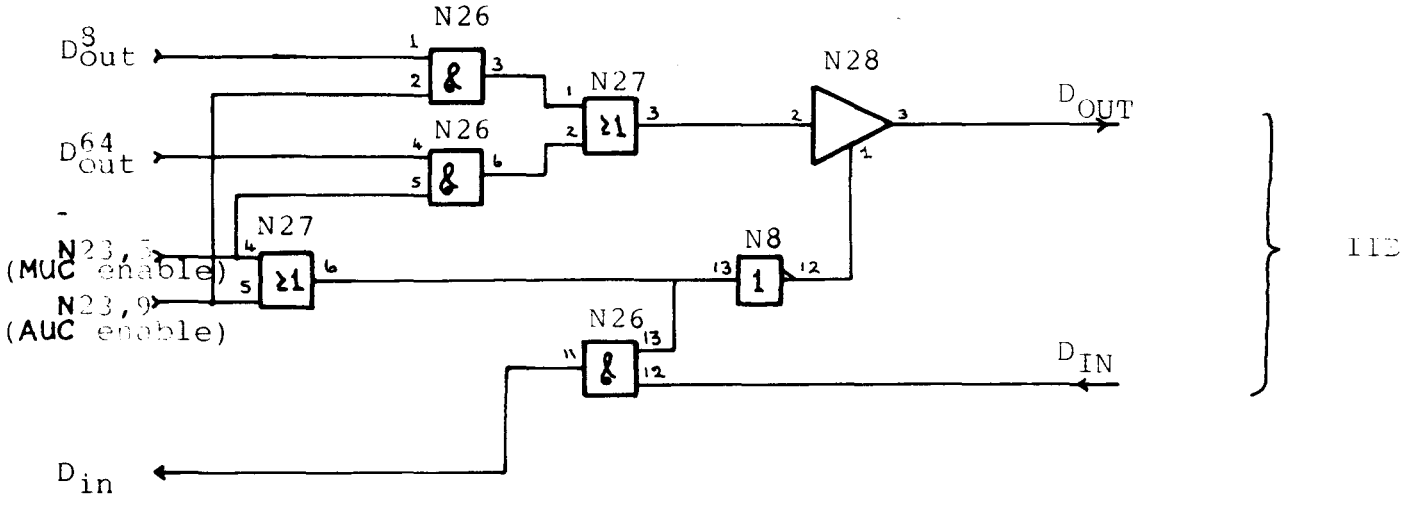


a) CIRCUIT DIAGRAM

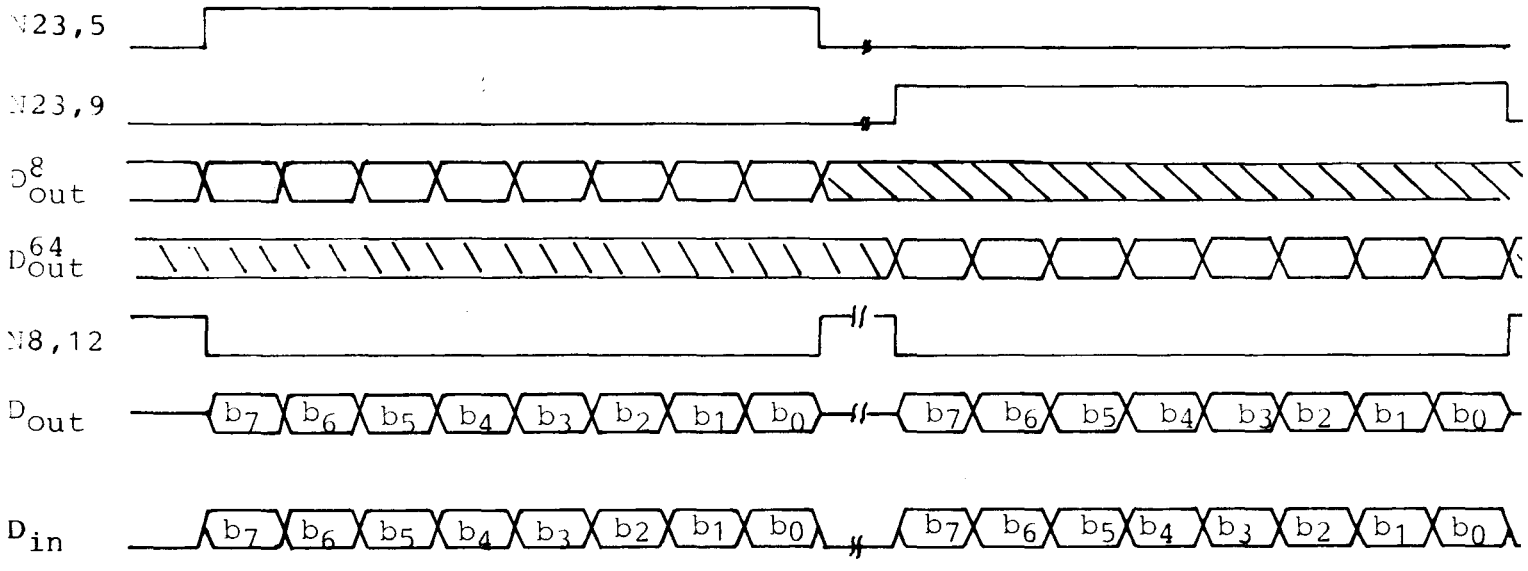


b) TIMING DIAGRAM

Fig.4.21 ADDITIONAL USER CIRCUIT:BUS→LTU



a) CIRCUIT DIAGRAM



b) TIMING DIAGRAM

Fig.4.22 BUS DRIVER/RECEIVER

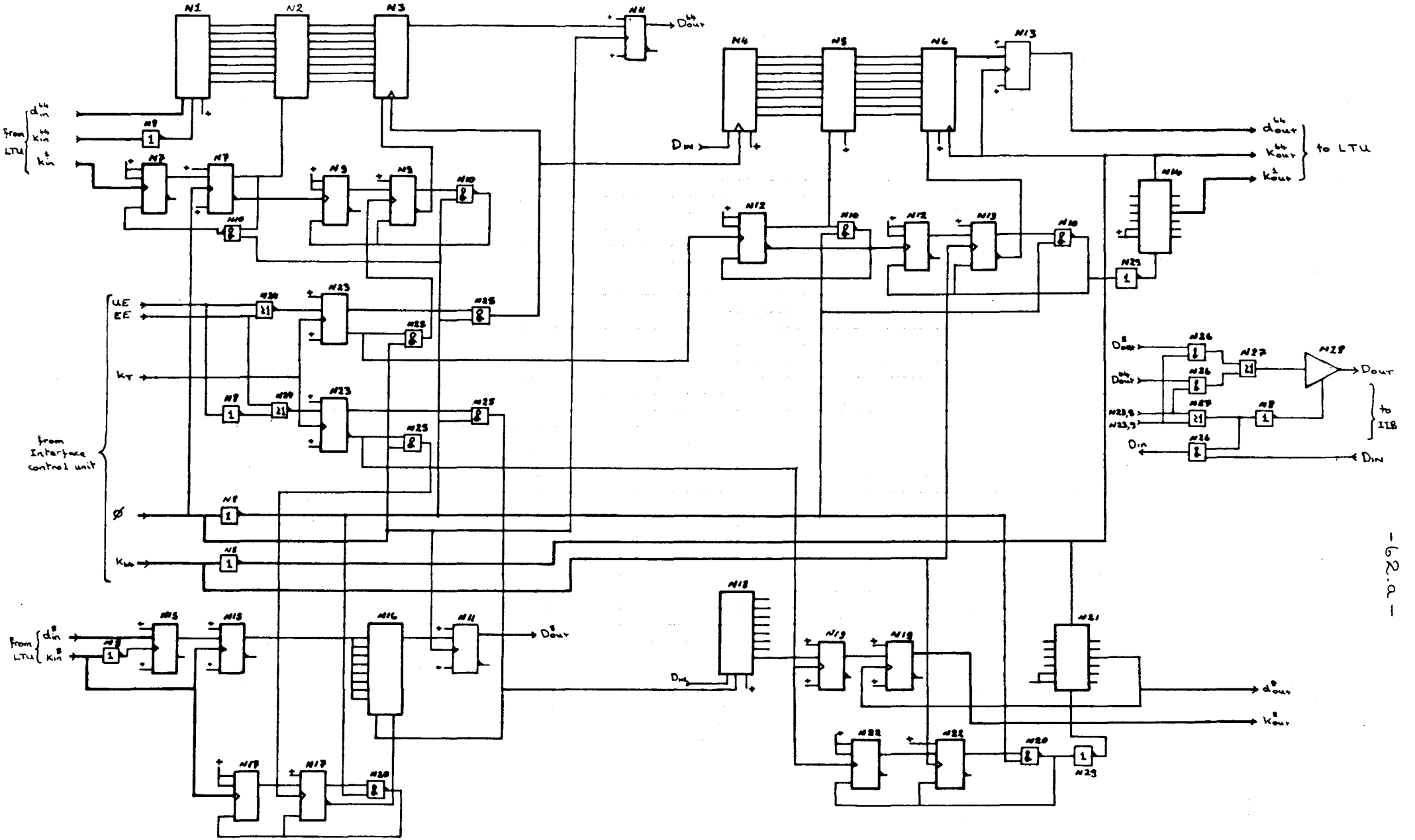


Fig. 23 COMBINED INTERFACE UNIT

4.4. Remarks

The hardware realisation proposed was, as far as possible, designed using elementary components. This was done with a view to the customised integration of the more frequently occurring elements, namely the interface units and the I/O buffers.

A disadvantage of the chosen configuration, namely the centralised control block, is that simultaneous allocation of a single timeslot, on the IH, to more than one user circuit is not possible. This leads to two main restrictions:

1) The direct creation of 'conference' type connections between users on the FEU is excluded. This would require the allocation of a single circuit (and therefore timeslot) on the IH simultaneously to all users involved.

2) The generation of conventional ('in band') audio tones, such as congestion and dial tone, in the FEU is not feasible. This would require, economically, the inclusion of only one generator per tone. As this could then only serve a single user at any one time the resulting level of stagnation would be unacceptable.

The former restriction would lead to such a conference connection being created by the main body of the exchange. The FEU would be unaware of such a state, each user involved being allocated a separate circuit on the IH as for a normal connection.

As far as the latter restriction is concerned the system operation must be considered further.
The system states, leading conventionally to these tones, are indicated to the user involved via dedicated messages in the corresponding signalling circuit. These would be displayed in some way, or be used to generate the required tones internally within the subscriber station. Alternatively, the system could be easily programmed to allow these tones to be generated within the main body of the exchange and transferred, via the corresponding user circuit, to the actual user.

If necessary these disadvantages could be solved by using the system configuration as shown in fig. 4.7.a in combination with a central timeslot clock counter.
Using this method, in which each interface unit is programmed separately by the Control Function, the allotment of a single timeslot simultaneously to multiple users is possible.

5. SIGNALLING CIRCUIT MANAGEMENT

5.1. Introduction

As stated previously, all control information necessary to support a connection is transferred to and from the subscriber stations via dedicated signalling circuits. Furthermore the control information transfer between the FEU and the Central Processing Unit (CPU), in the main body of the exchange, occurs via an inter-processor signalling circuit on the IH.
The former, consisting of a circuit with capacity 8 kb/sec., has an interface at the LTU precisely the same as that for the additional user circuit of chapter 4. The signalling circuit is terminated in the FEU, where its contents must be processed in some way, while the user circuits of chapter 4, as shown in fig. 5.1, are merely switched onto a corresponding circuit on the IIB for further transmission.

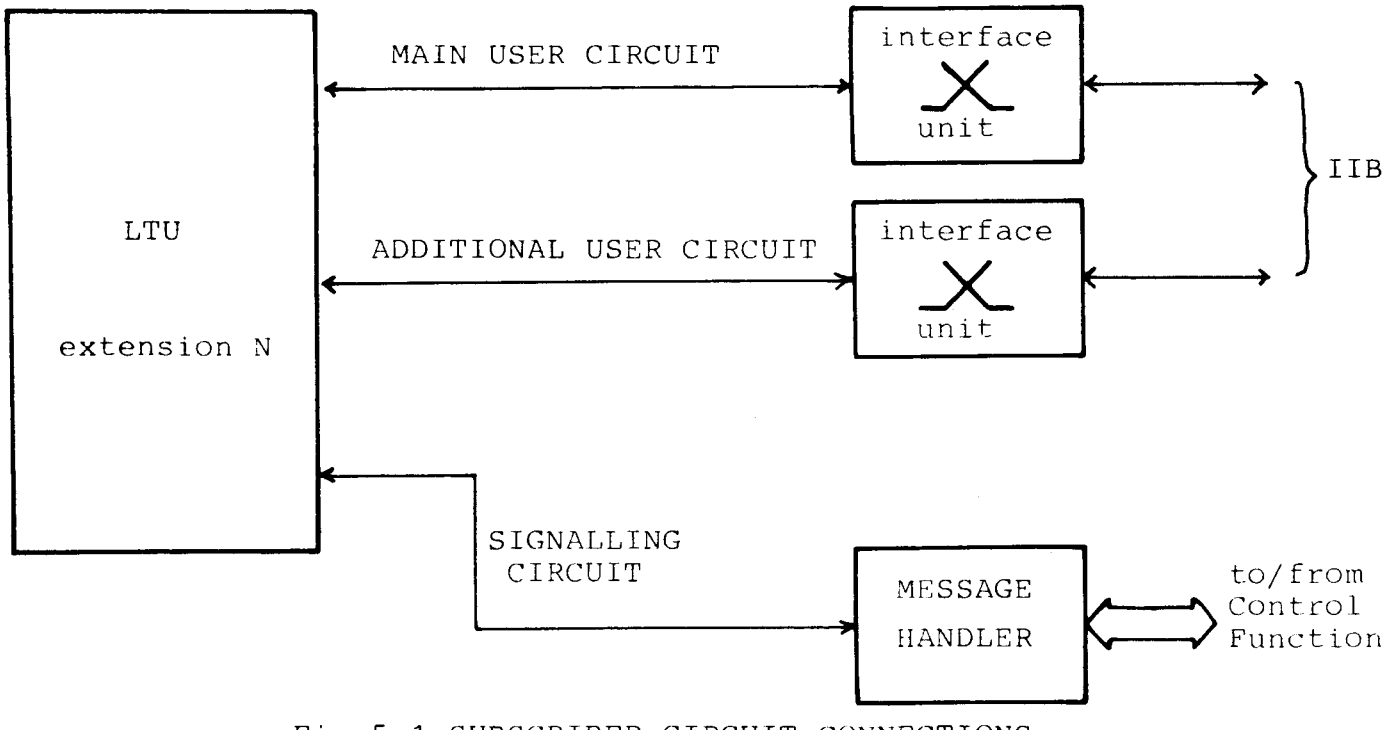


Fig.5.1 SUBSCRIBER CIRCUIT CONNECTIONS

Messages on the subscriber signalling circuits generally either result in, or occur from, some action in the Central Processing Unit (CPU). The Control Function of the FEU pre-processes messages from either the CPU or subscriber station, and carries out any actions or further message transfers specified by that processing. An Inter Processor signalling circuit (IP) is used to transfer these messages to and from the CPU. This circuit consists of a dedicated timeslot (number 31) per frame on the IIB, and has thus a capacity of 64 kb/sec. This Inter-processor signalling circuit must then, cater for the concentrated signalling to and from the 32 subscriber stations, plus any internal system control signals.
Fig. 5.2 indicates the method of connection with the IIB for this circuit, the interface being identical with that for the user circuit interface units of chapter 4 (timing signals being supplied indeed by the user circuit management function).

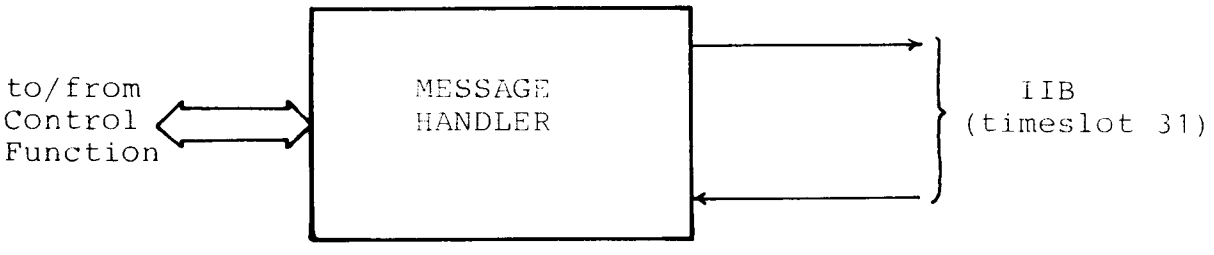


Fig.5.2 INTER PROCESSOR SIGNALLING CIRCUIT

In figures 5.1 and 5.2 a message handler has been included; the significance of this will become clear in later sections.

The signalling circuit management function is therefore split into two regions, dealing with respectively:

1) subscriber line signalling circuits,
2) inter-processor signalling circuit.

The messages mentioned consist of the coded description of an event - an event being some state or occurrance, e.g. dial cipher 6, off hook condition, proceed to select signal or etc. These events are discrete in time and each is to be described by a dedicated message. A message, then, either arises from the occurrance of a given event, or is used to cause the corresponding event to take place, depending on the source of the message.
As mentioned in chapter 4 the standard message format to be used in the system consists of two bytes. The first identifies the user with which the message is concerned (source or destination). The second, the actual message content, is the description of the event. Using one byte for this latter purpose leads to a maximum of 256 discrete messages (or events). This is considered to be more than adequate for expected needs but, should more be required, then either:

1) a multiple byte 'message content' should be used
or 2) each signalling circuit, subscriber and IP, should be considered as having an independent vocabulary, i.e. identical messages on the two could have different meanings.

The following sections deal with the general operation of these circuits. Where appropriate, implementations of parts of the message handlers, and etc., mentioned will be considered. The two types of circuit, namely subscriber and inter-processor, are handled separately.

5.2. Subscriber Loop Signalling Circuits.

5.2.1 General Structure

As explained previously this circuit is to be shared, on a demand assignment basis, between the various users within the subscriber station.
The responsibility of this circuit is to transfer signalling characters between a generating source, which could either be the exchange or a user, to a destination, in an orderly and reliable manner.

Reliability here means that a signal offered at the input, leads to the same signal eventually occurring at the output of the circuit. In the transfer medium, i.e. the transmission system, there is a possibility that errors occur in the bits being transferred, thus the signalling circuit must include some form of error correction technique.

The signalling system can be described by a layer structure (somewhat similar to that used in data transmission networks), as shown in fig. 5.3. A higher order layer uses those below it as transfer mechanisms.

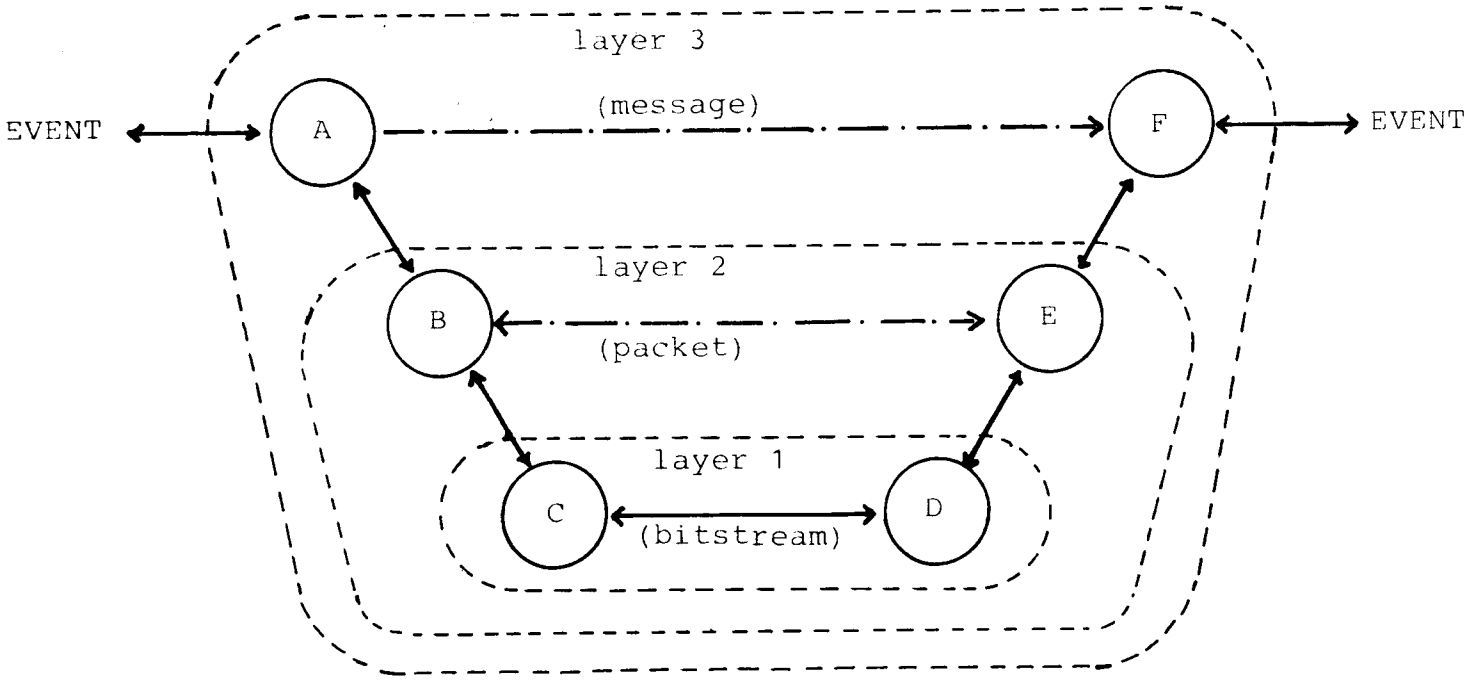


Fig.5.3 PROTOCOL STRUCTURE

Layer 1 is the physical bit transport over the transmission system.

Layer 2 facilitates the reliable message transfer by correcting errors occurring in layer 1.

Layer 3 is the actual signal transfer layer. Here the signals have a meaning (whereas in lower layers the contents are not considered specifically).

his layer structure, specifically the nodes, can be translated into the physical elements of fig. 5.4.

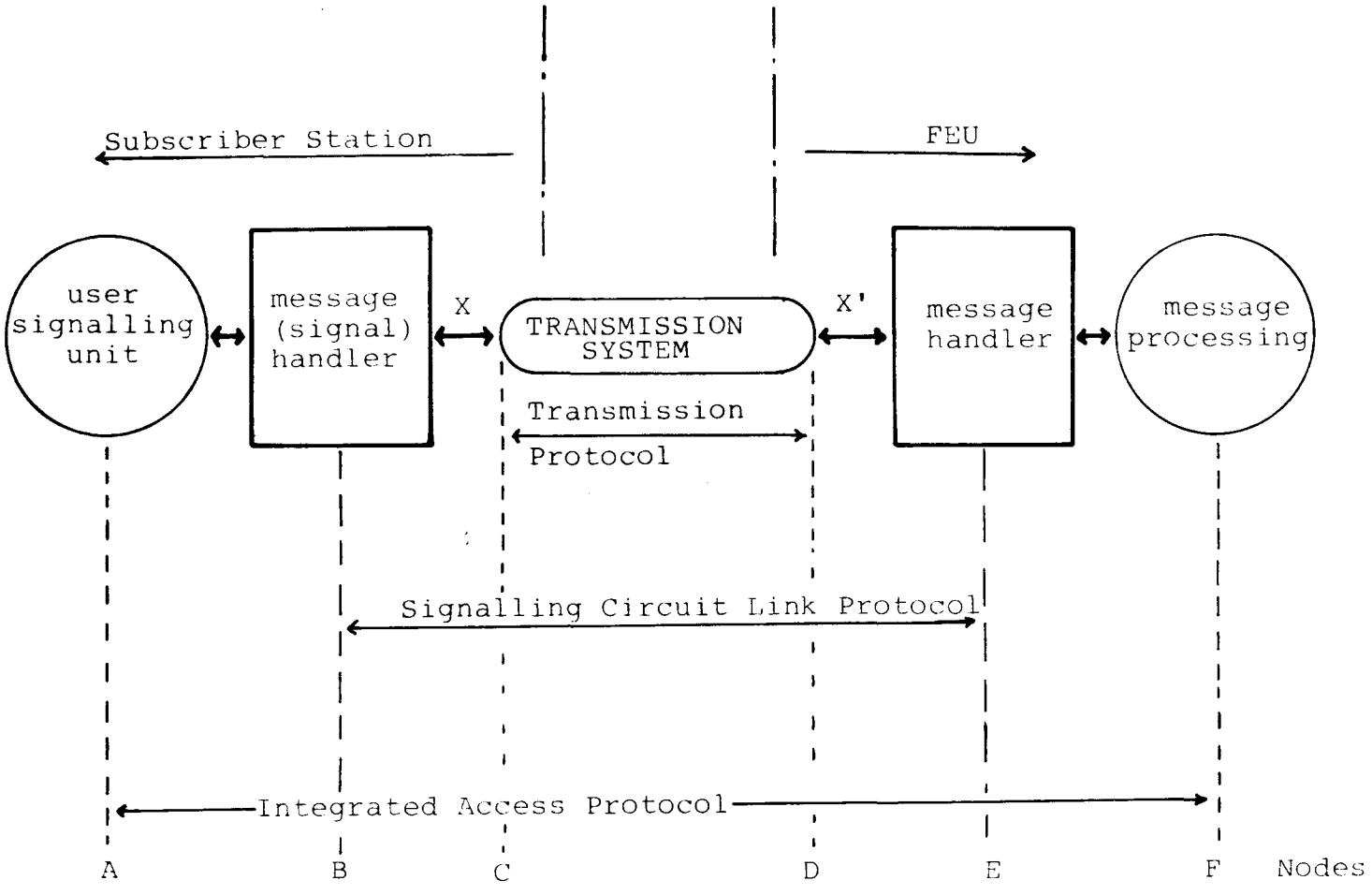


Fig.5.4 PHYSICAL SIGNALLING SYSTEM

Here a number of protocols have been indicated. A protocol is seen as a set of operating rules, governing the transfer of information between two nodes, which must ensure that the required reliability of operation is achieved.
These are:

1) A transmission protocol, which defines the method of transmission, i.e. the form of the pulses on the transmission system, the method of achieving a duplex circuit, any (de)multiplex function required within the LTU and etc.

2) Signalling Circuit Link Protocol, which defines the methods of detecting and correcting errors in the bits transferred over the transmission system.

3) Integrated Access Protocol which describes the sequences of messages (and therefore state changes) necessary to create and maintain a connection, i.e. this deals with the actual contents of the messages.

The actual protocol in 1) will not be considered further — access to the transmission system being obtained via the signalling circuit interfaces with the LTU (X-X' in fig. 5.4).
The work described in the following sections is concerned with:

a) The message handlers and the Link Protocol operating between them. This protocol being implemented in the handlers.

b) The coupling, in the FEU, between the message handlers and the Control Function. This coupling being implemented for the group of 32 extensions.

c) The Integrated Access Protocol, operating between users within the subscriber stations and the Control Function within the FEU.

5.2.2. Signalling Circuit Link Protocol

This operates between, and is implemented by, the message handlers situated in the subscriber station, respectively FEU, as shown in fig. 5.5. The signalling circuit consists of two independent channels, both operating with a capacity of 8 kb/sec. (Here, the specific transmission system used is considered as a 'black box'.)

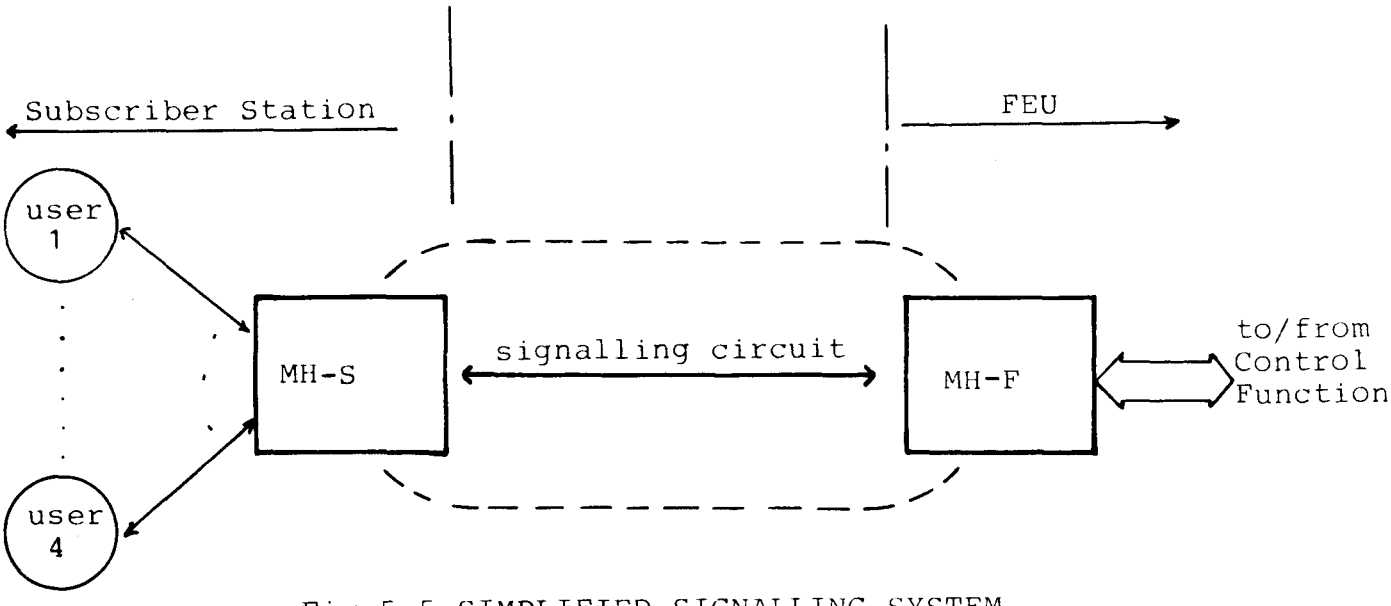


Fig.5.5 SIMPLIFIED SIGNALLING SYSTEM

The protocol must ensure that messages are exchanged between users/FEU Control function reliably and in an orderly manner.
To accomplish this, use must be made of some error correction technique,plus some form of flow control.

Many such systems are possible. Of these one has been designed which has the qualities:

- robustness against errors, due to a rigorous error control, i.e. reliable;
- flexibility, as the signalling system is considered as an apart entity with its own synchronisation system;
- relative efficiency; the amount of extra redundancy used to combat errors is dependent on the error conditions occurring at that time;
- simplicity, a fixed length packet, as will be used, simplifies any hardware necessary in the implementation.

This design, due to shortage of time, is limited to the theoretical description of the protocol, a hardware implementation not being made. The characteristics of the protocol must be known due to interworking with, and effect upon, the control function of the FEU. The implementation of this could be considered as part of any further work on this subject.

The basic concept is that a message to be transferred has a packet formed arount it, as shown in fig. 5.6. This packet contains the necessary control information used to combat errors.

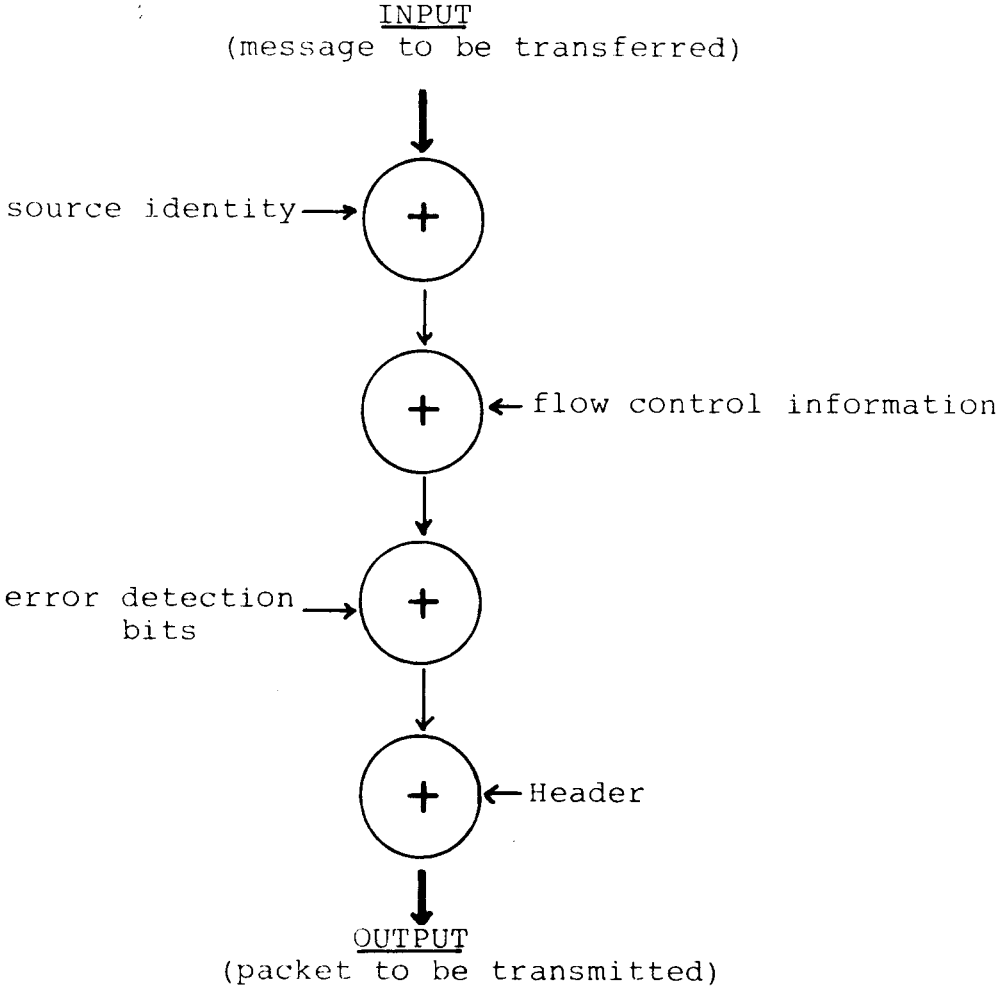


Fig.5.6.a PACKET FORMATION

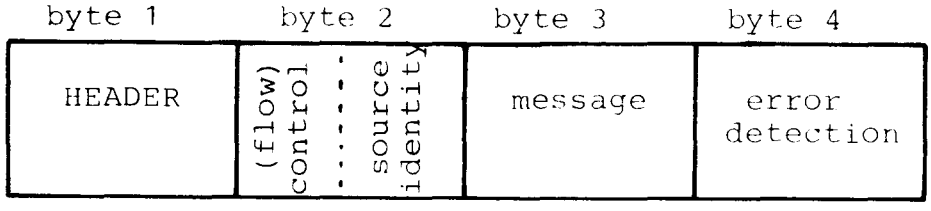


Fig.5.6.b PACKET FORMAT

Here the output packet consists of:

a) Header; a fixed word which indicat the start of a packet at the receiver, i.e. this synchronises the receiver to the incoming packet.

b) Flow control; which consists of:

- a cyclic number (0 or 1) which is used to check that the message sequence is correct, i.e. that no messages have been lost;
- a class identification bit, which indicates whether an actual message is included in the packet, or that only flow control information is present;
- an acknowledgement of the last message received, in the form of its cyclic number.

c) Source Identity; the identity of the user with which the present message being transmitted is concerned.

d) The actual message, if applicable, or some suitable 'idle' pattern.

e) Error Detection Field; consisting of an 8 bits Cyclic Redundancy Check of the contents of fields b), c) and d).

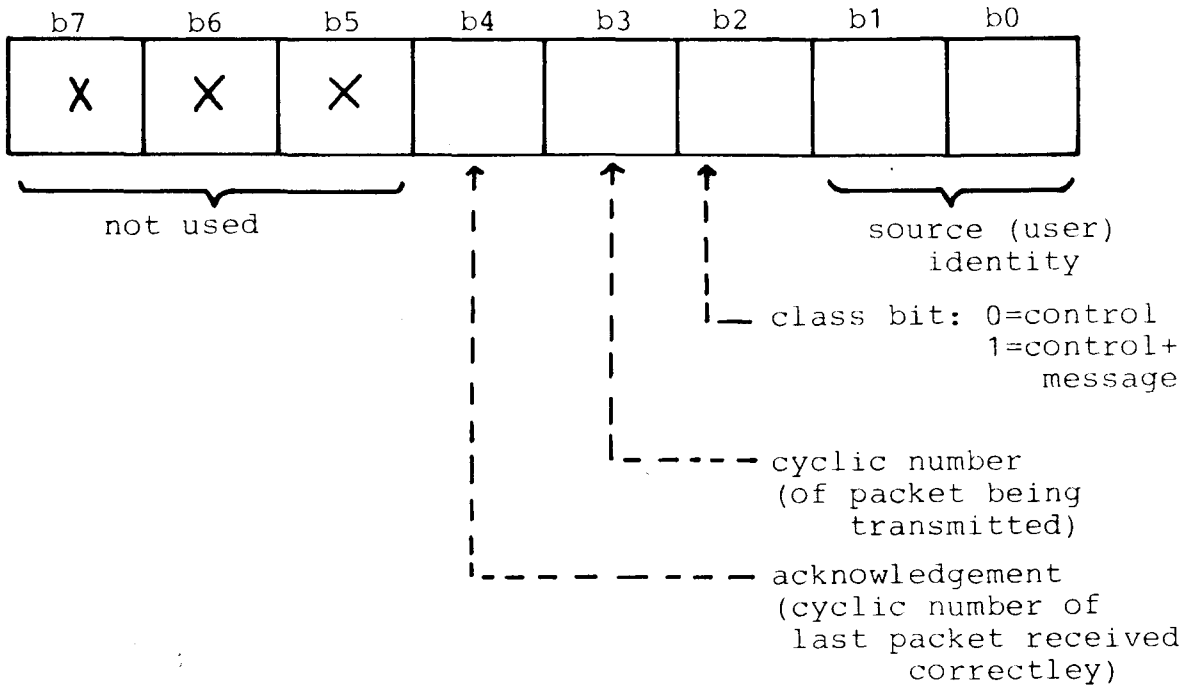


Fig.5.7 BYTE 2 OF PACKET

Figure 5.7 gives the detailed contents of the second byte of the packet. It is seen that two logical channels are created - as well as the obvious transmit channel. b_4 represents a channel connected with the information in the receive direction.

The header field would be choosen in such a way as to reduce, to a minimum, the chance of its imitation by a message. The CRC field would be created by one of the standard polynomial generator codes.

Operation

This will be described with reference to fig. 5.5.

An event occurring in one of the user signalling units leads to a message being transferred to message handler S (MH-S). This is then used to form a packet, which is, in turn, transmitted to MH-F.

This packet is repeatedly transmitted until an acknowledgement of its correct reception at MH-F is made (this is indicated by the reception, at S, of the cyclic number of the present packet being transmitted by S, in bit 4 of an incoming packet from F). The repetition of this packet may then cease.

This acknowledgement indication is repeated in every packet transmitted from F, until a new packet is received from S, in which the cyclic number has been incremented wrt the last previously received and verified packet in the sequence.

The occurrance of a further message at S, leads to the same process (assuming that the previous packet has been acknowledged by F) occurring, but with the cyclic number of this new packet being incremented, modulo 2, wrt its predecessor.

The case can arise, at S, in which the last packet transmitted to F has been acknowledged, and the acknowledgement of a packet received from F is now to be sent - thus no new message is to be transmitted from S. The packet formed around this acknowledgement signal, although otherwise normal, does not have its cyclic number incremented. This means that, in this case, any new message occurring at S can now be transmitted, without excessive waiting times, to F (the appropriate packet parameter alterations must be made). The acknowledgement of a packet containing purely control information is avoided. The cyclic number of the packets is therefore message orientated. The consequence of this is then, that in the receivers, account must be taken of the fact that consecutively received packets, of the same cyclic number, may contain differing control fields.

The acknowledgement of the 'acknowledge signal' from S occurs inherently in the receipt of the next packet in the sequence, from F. This also means that the transmitters are continuously active during a connection, transmitting acknowledgement packets if nothing else.

At system initiation time, or at startup after failure, it is important that both message handlers do not take a random state. If S starts transmission with an acknowledge to the packet with cyclic number 0 from F, while F begins transmission with the packet, cyclic number 0, then this message may never be received at S. An orderly startup procedure would be— if S acknowledged packet 0 from F while F begun transmission with packet number 1. (A dedicated message, or bit, could also be used to bring the system into this state, i.e. a restart state.)

Only one channel of the circuit has been considered - the second, namely from F to S would be identical and, due to the duplex nature of the circuit, could act concurrently with that from S to F.

This was the basic protocol to be used on the signalling circuit, its operation being shown graphically in figs. 5.8 to 5.10 and block diagrams of the message handlers in fig. 5.11.

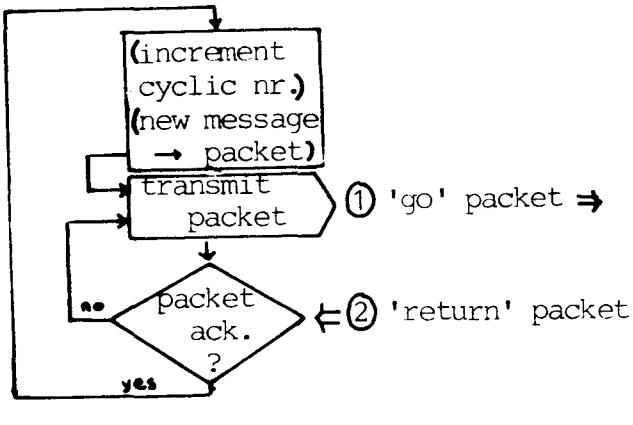


Fig.5.8 TRANSMITTER

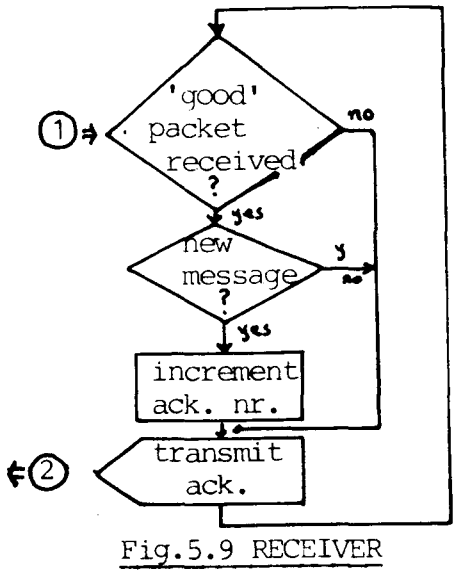


Fig.5.9 RECEIVER

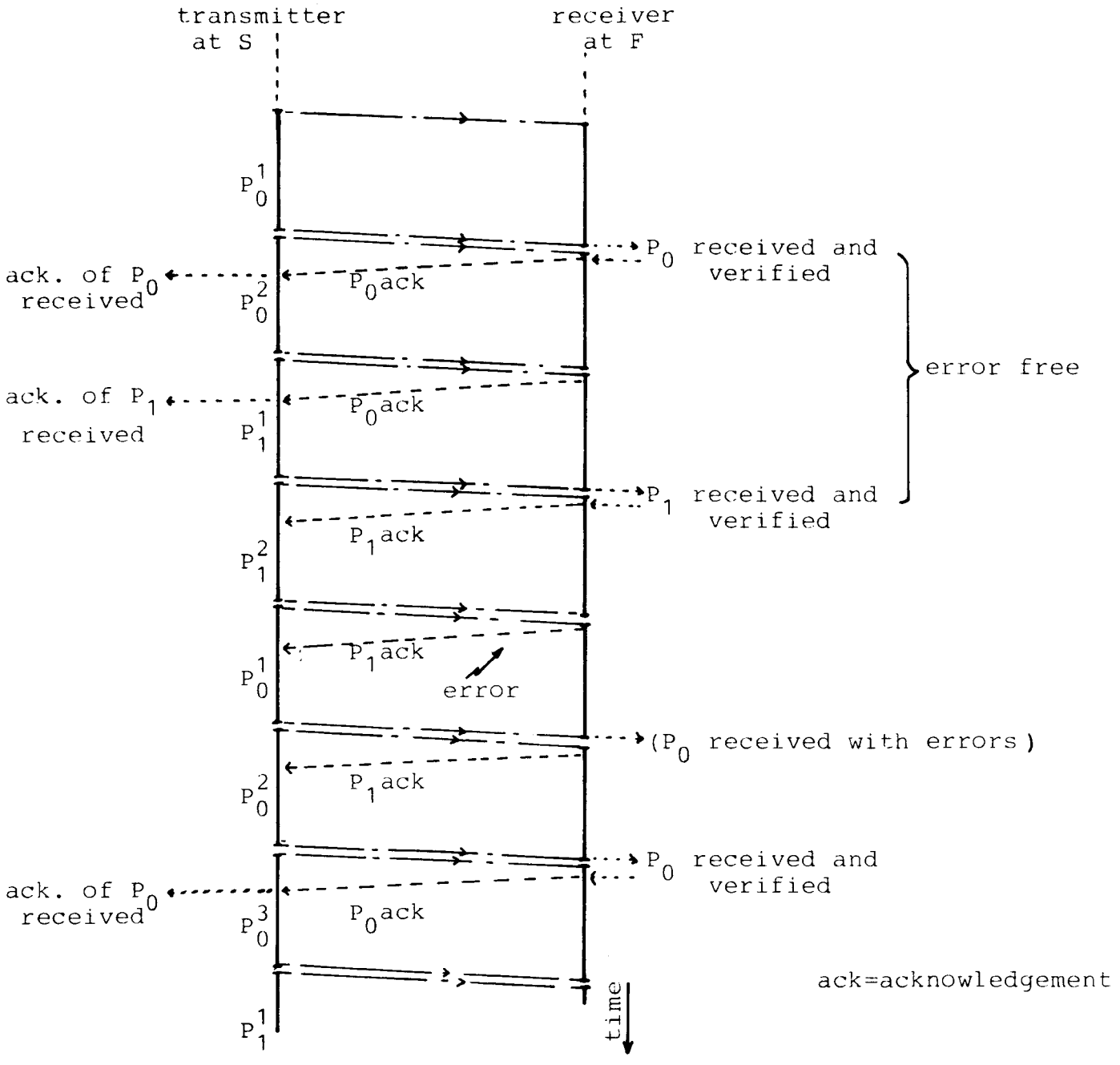
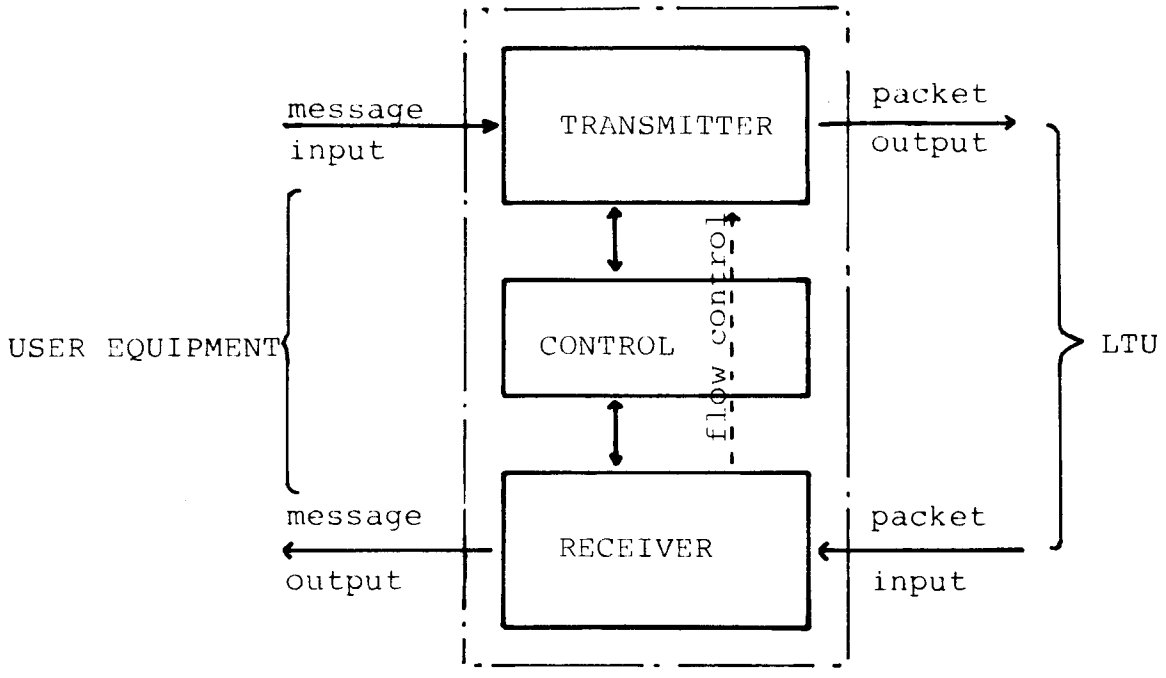
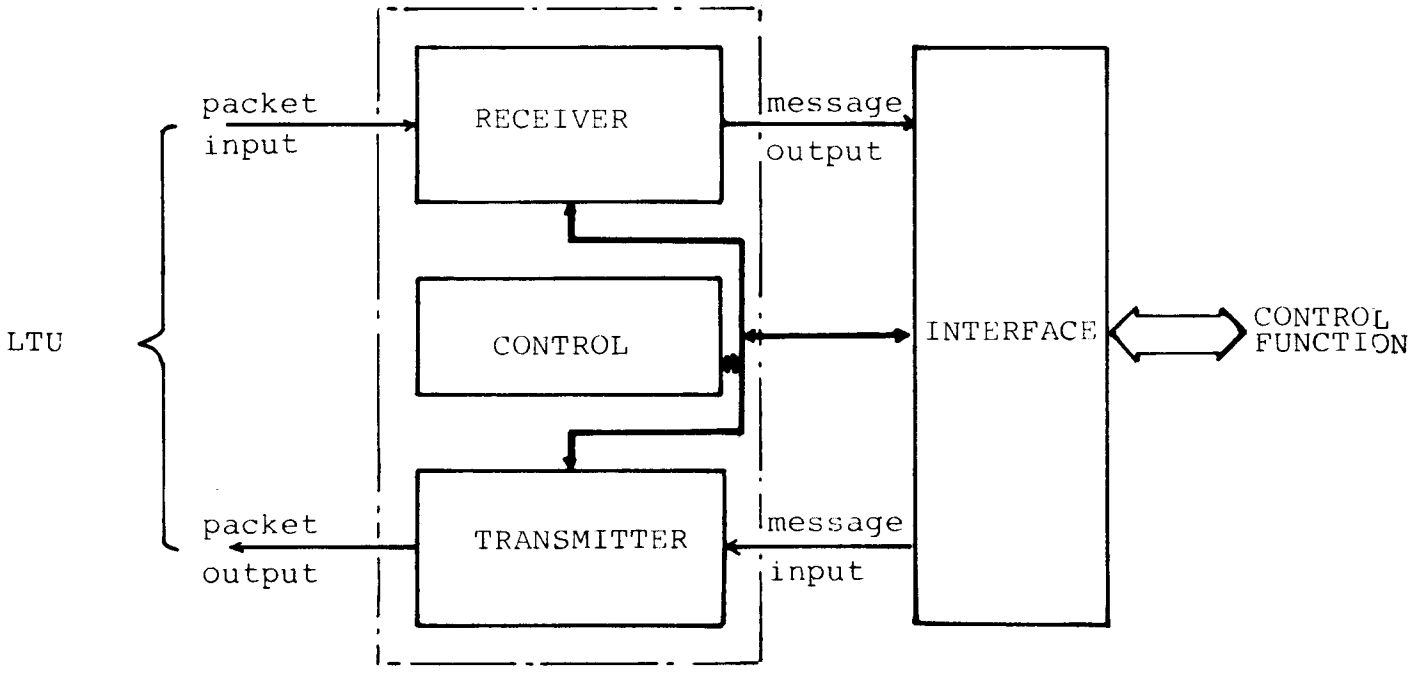


Fig.5.10 PROTOCOL OPERATION: TIMING



a) SUBSCRIBER STATION



b) FEU

Fig.5.11 BLOCK DIAGRAMS: MESSAGE HANDLERS

5.2.3. Calculation of the approximate throughput of the system

A numerical approxiation of the throughput, i.e. the number of messages/second, which the system can achieve, must be obtained. As will be seen, this is a rough estimate due to the boundary conditions assumed as well as the uncertainty as to the conditions under which the system must operate.

basic circuit capacity = 8 kb/sec.
message length = 10 bits (i.e. message + user identity)
The 'naked' system could then, transfer 800 messages/sec.

packet length = 32 bits
packet duration = 4 msec.
This leads to the absolute maximum signalling rate, SR_{amax}, of 250 messages/sec.

This is still further reduced by the acknowledgement sequence, as shown below in fig. 5.12.

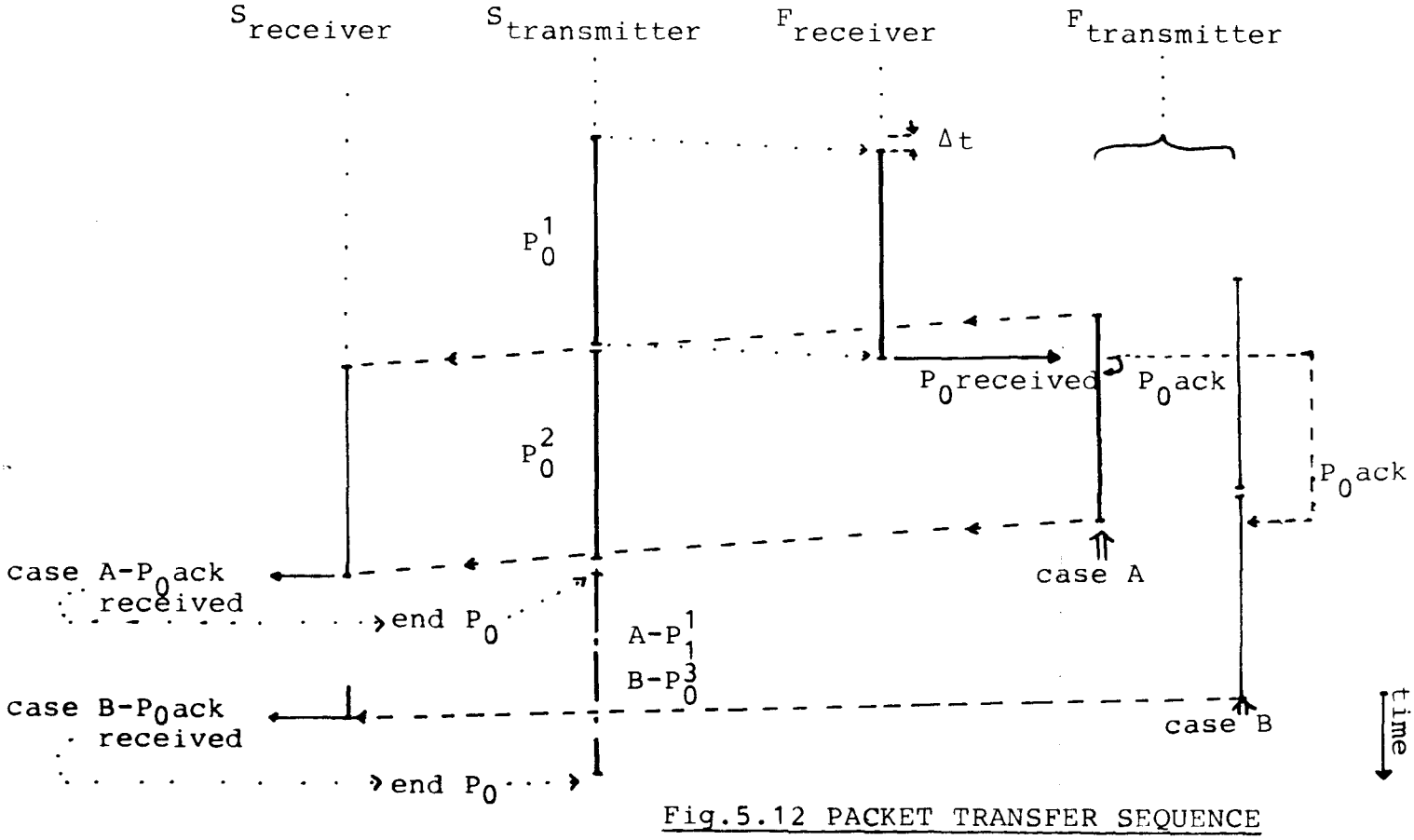


Fig.5.12 PACKET TRANSFER SEQUENCE

It is seen from this that under ideal circumstances, i.e. with propogation delay ($\Delta t = 0$), and when the packet returned from F can have the acknowledgement of P_0 entered directly (as in A), there will always be at least two repetitions of a packet. Case B, in which the acknowledgement can no longer be entered into the present packet from F, leads to a minimum of three repetitions of a packet. As it has been assumed that the time at which a packet is transmitted is random, then a maximum signalling rate is obtainable, of:

$$(\frac{250}{3} < SR_{max} < \frac{250}{2}) \quad \text{i.e.} \quad 83^1/_3 < SR_{max} < 125 \text{ messages/sec}$$

In order to increase the throughput it would then be worthwhile to either:

1) sample the acknowledgement signal just before the control field of the packet in which it is to be included is transmitted.

or 2) to synchronise the system in such a way that the return packet from F begins just after receipt of an incoming packet from S.
In this way the packet header could also be discarded. The consequence of this, is that a relatively complex synchronisation check would become necessary.

The above description has not taken into account any errors arising in the transmission system - these will tend to reduce throughput even further, due to the extra repetitions which they cause.

Assuming that errors occurring do so randomly, and show a Gaussian Distribution with an average probability of error, for any given bit, p, the following effects occur:

The probability of errors occurring and not being detected would require then, 8 or more errors occurring within the field protected by the CRC - and even then, not all such cases would be undetected.

Pr (undetected errors) < Pr (8 or more errors in 24 bits)

= 1-Pr (7 or less errors in 24 bits)

$$= 1 - \sum_{1=0}^{7} P_i^x$$

where P_i^x = prob. of i errors in x bits

$$= \binom{x}{i} p^i (1-p)^{x-i}$$

$$= 1 - \left[(1-p)^{24} + \binom{24}{1} p(1-p)^{23} + \ldots\ldots + \binom{24}{7} p^7 (1-p)^{17} \right]$$

Even with a very high error rate of 1 in 100, i.e. p = 0,01, this probability is negligible. It can therefore be assumed that the occurrance of errors will always be detected.

The effect of errors on the normal operation must now be considered. An error in a packet causes that packet to be discarded - thus some extra repetition occurs. The packet consists of 32 bits, and three error regions will be considered;

normal, with p = 0,0001
high, with p = 0,001
very high, with p = 0,01

p	P_0^{32}	P_1^{32}	P_2^{32}	P_3^{32}	$\sum_{i=1}^{32} P_i^{32} = P_m$
0,0001	0,997	0,003	-	-	0,003
0,001	0,969	0,030	< 0,001	-	0,031
0,01	0,725	0,234	0,036	0,003	0,275

The probability, under these conditions, that any packet is discarded is then the probability that one or more errors occur, i.e. P_m.

A single extra repetition occurs if one packet is discarded. This can occur in one of two ways; either the packet carrying a message can have errors, and/or that with the acknowledgement can have errors. The probability of an extra repetition, P_d, is then:

$$P_d = P \text{ (message error)} + P \text{ (ack. error)} + \{P \text{ (message error)} \cdot P \text{ (ack. error)}\}$$
$$\simeq 2P_m$$

as the 'go' and 'return' channels here, are assumed independent and identical, this gives the following table for the various error rates.

p	P_d
0,0001	0,006
0,001	0,061
0,01	0,558

For an error rate of 1 bit in 1000 approximately 6% of the transfers will be elongated due to the errors, i.e. SR: $78 < SR_{max} < 118$ messages/sec.

For an error rate of 1 in 100 the effect is much more serious, giving SR: $33^1/_3 < SR_{max} < 50$ messages/sec.

Considering now the possibility that consecutive packets from a given transmitter are in error, e.g. where A = 1st packet, B = second packet and etc., an approximation of the effect can be made by assuming

$$\begin{aligned} \Pr \text{ (A and B both contain errors)} &= P(A_e.B_e) \\ &= P(A_e)\ P(B_e/A_e) \\ &\simeq P(A_e)\ P(B_e) \\ &\simeq P(A_e)^2 \end{aligned}$$

$$\Pr\ (A_e.B_e.C_e \text{ contain errors}) \simeq P(A_e)^3.$$

This leads to the following table.

p	P_m	P_m^2	P_m^3
0,0001	0,003	-	-
0,001	0,031	0,001	-
0,01	0,275	0,076	0,021

Even this second order effect, for an error rate of 1 in 100, is seen to be significant.
The effect of these consecutive errors is again to increase the number of repetitions necessary.

From the above calculations an approximate value can be put on the thoughput of the system:

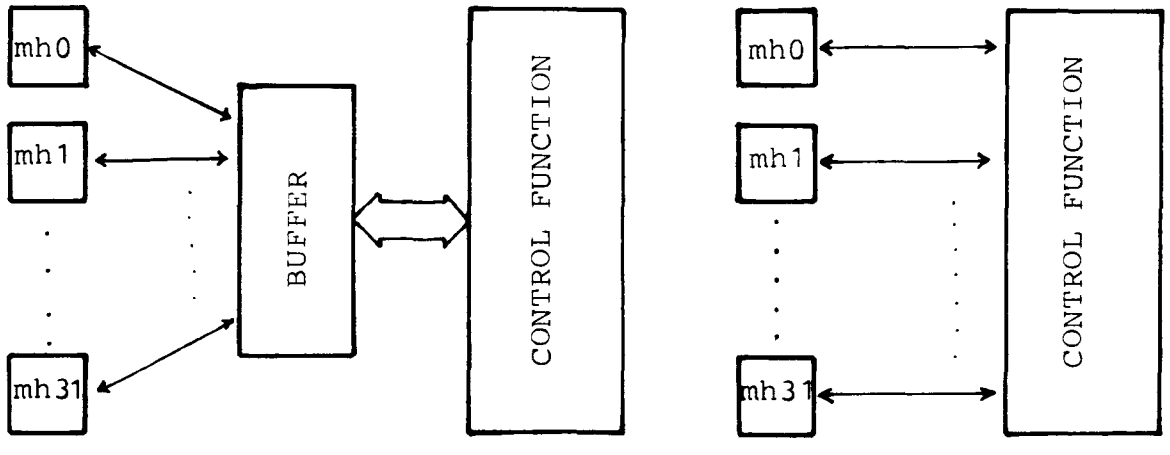
For a system in which the error rate seldomly lies above 1 in 1000 a signalling rate of > 70 messages per second is realistic.

Should the error rate increase above this value the possible signalling rate decreases to the point where, with an average error rate of 1 in 100 the signalling rate lies in the region $\pm$ 25 messages per second.

As it is unlikely that the system will operate continuously under such extreme conditions as this latter, it is reasonable to assume an <u>average</u> throughput of <u>> 50 messages/sec.</u> Comparing this with the 800 messages/sec. for the naked message transfer shows the cost of the error correction supplied by the protocol.

5.2.4. Message Handler - Control Function coupling

Having defined the link protocol which operates over the signalling circuits, the coupling between message handlers and Control Function will now be considered. The message handler for each extension, 32 in total, must be connected, in some way, to the Control Function. (The control function carries out the actual processing of the contents of a message.)
The two basic actions which must be carried out are the transfer of a received (and verified) message from the handler to the control function, and vice versa for a message resulting from the processing within the control function.
Two basic methods of coupling are possible, namely indirect and direct, as shown in fig. 5.13. In this figure the arrows indicate message transfer paths.



A) INDIRECT B) DIRECT

(mhN=message handler for extension N)

Fig.5.13 HANDLER-CONTROL FUNCTION COUPLING

A) Indirect Coupling

A buffer process is inserted between the control function and message handlers. This buffer must collect messages from the handlers, store them, then transfer the stored messages to the control function when it is ready to accept them. By allowing the buffer to collect several such messages before initiating their transfer to the control function, then by causing this bulk transfer to occur quickly, the disturbance to the processing action within the control function can be minimised. In this way the control function spends less time carrying out unproductive work, i.e. collecting the messages. Also, as the direct control of the relatively slow acting message handlers lies in the buffer process, then the workload on the control function is reduced further, i.e. no check on each of the handlers need be made in order to distinguish those which have a new message to transfer. Message transfer in the opposite direction, namely control function to buffer process, could be done using either the same bulk transfer technique or on a single message basis. By giving this transfer precedence over any other which the buffer may have to carry out, e.g. that to or from a handler, leads to the most efficient control function operation. The buffer process would then be responsible for the transfer of these messages to the specified handlers when this was ready to accept the new message. The various methods of carrying out this latter transfer are identical with those for the direct coupling, as in fig. 5.13.b), and will be considered in combination with this direct coupling.

The main advantages in using such an indirect coupling are then:

- the control function workload is reduced: the time consuming coupling with the message handlers being carried out by the buffer;

- flexibility: alterations to any of the constituent elements (handlers, coupling technique, buffer or control function) need have no effect on the other elements not directly connected with the alteration.

Disadvantages are that:

- the component count, and therefore cost, is increased wrt that for a direct coupling;
- delay between message generation and processing of the message is increased.

An initial investigation of the implementation of this buffer technique was carried out. The details of this will be discussed very briefly.

The buffer unit itself, consisted of a 8049 micro-computer, internal RAM and ROM being sufficient to implement the required functions.
The status of each message handler was sequentially scanned in order to check as to whether any messages were ready for transfer. If such a state was detected in a handler the message was collected, i.e. read into the 8049.
Four such messages were collected. The completion of this block lead to an interrupt signal being generated to the control function. On acknowledgement of this signal the block was transferred to the control function (the scanning was neglected during this relatively short period in order to speed up the transfer operation).
In order to reduce any excessive waiting times before a message were transferred, in such a block, to the control function, a time limit was set.
The collection of the first message of a new block lead to a timer being set. If the timer period was exceeded before the block of four messages had been collected then, via an interrupt to the 8049, the transfer of this incomplete block was initiated (a parameter, transferred with every block was then the number of messages included).

Single messages were transferred from the control function to the 8049 on an interrupt basis. These were stored until the specified message handler was ready to accept them. A second scanning routine, interleaved with that mentioned previously, was used to determine whether the specific handler, addressed by a message, was free to accept the message. If this was so the message was transferred to it, otherwise it was held in store until a later time at which the process was repeated.

Although not completed in every detail, this investigation did show that such a system was generally feasible within the constraints of the disadvantages previously mentioned.

The main type of traffic being handled consists of relatively low speed, manually generated messages and reactions to these from the system. This, in combination with the low workload on the processor within the control function, as will be seen in chapter 6, led to a choice being made in favour of the direct coupling technique. The implementation of this buffer was investigated as it is considered to be a real alternative to the direct coupling used.

B) Direct Coupling

Using this technique, as shown in fig. 5.13.b), the control function has direct responsibility for the transfer of messages to and from each message handler.
Two main operations can be distinguished here:

1) The collection of messages from the individual handlers, and the indication by a handler to the control function, that it contains a message which must be collected.
2) The transfer of messages to the specific handler, identified within the message, from the Control Function.

Two basic methods of carrying out this operation are possible, namely polling and interrupt.

1) Polling

The control function scans the status of each message handler in turn, this sequence being repeated cyclically. The status is then checked and if found active (i.e. the handler contains a message to be collected), the corresponding message from that handler is transferred to the control function. This polling routine can be carried out either at fixed intervals or (within limits) when the control function has no other processing to carry out. The routine itself can either be based on a full cycle, i.e. once started all 32 handlers are checked, or applied discretely to one or more handlers at a time, depending on how much time is available.

The advantages here are that as the time of transfer and etc., are under full control, then a well synchronised system is obtained. Also, through the scanning action, the identity of the handler from which the message is to be transferred is directly available. Flexibility is built in as the scanning routine can be altered easily without effecting other system elements.
Being basically a software controlled operation, the time required to carry out this process is relatively long, as is the average time for which an active handler is kept waiting before the message is collected ($\pm \frac{\text{scan period}}{2}$).
A further disadvantage lies in the control function overhead - the scanning must be carried out whether any handler is active or not.

2) Interrupt

In this case a handler which becomes 'active' (i.e. contains a message to be collected) must generate an interrupt signal to the control function. This signal leads, in the control function, to the interruption of any processing being carried out at that time (if this is tolerable). The generating source is then identified, the message collected and the previous processing is resumed where it left off.

The obvious advantage here is that control function overhead only occurs when necessary, i.e. when there is a message to be collected. The interrupt process itself also tends to be of relatively short duration, which means that messages are transferred quickly to the control function, thereby freeing the handler.

The greatest disadvantages are that the particular handler which generated the interrupt must now be identified explicitly, and that differentiation between multiple handlers which activate simultaneously, must be carried out in the interrupt process.

Of these two methods that using interrupt techniques was chosen for the system being considered. This was done for two main reasons, the first of which played the greatest role in the decision made:

1) The speed with which messages are transferred to the control function. The message handler, containing very limited storage capacity, is not able to acknowledge a message received until this has been transferred to the control function - otherwise it could be overwritten by a new message arrival.

Using the polling technique in this case could lead to a serious reduction in the possible signalling rate achievable over the signalling circuit.
This technique of 'acknowledgement after collection' has the advantage that the control function itself can modulate, to some extent, the intensity with which messages are transferred to it. In excessively busy periods a degradation in service can be used to avoid the 'flooding' of the system.

2) The amount of overhead produced in the control function, in order to handle the relatively infrequent message arrivals, using polling, was considered excessive. Such a loading reduces the efficiency with which the actual processing can be carried out. Consideration here was given to the arrival pattern of messages - the activation of the interrupt process costs overhead in that the system state, at that time, must be saved for later resumption. This overhead would approach that for polling only for very frequent arrivals.

C) Message Transfers to Handlers

This transfer is relatively simple. Each message includes an identification of the extension (and therefore handler) which is its destination. This can be used to address the handler concerned. Initially a status check must be carried out as the transmitter in the handler concerned, may be busy with the transfer of a previous message to the extension. If this is so, then the transfer from the control function would be attempted again at a later time. If free the message would be transferred to the handler.

The system, to be implemented, is then based on:

- interrupt signals from a message handler which has a message to be transferred to the control function;
- direct addressing, by the control function, of the handler to which a message is to be transferred.

5.2. . Implementation of the Message Handler - Control Function Coupling

The basic unit used for the transfer of messages to and from the control function is, as considered previously, an I/O Buffer. This supplies the temporary storage facility for messages awaiting collection by the control function or transmission to an extension. As each handler must have this facility, then each will have an I/O buffer built into it, as shown in fig. 5.14. Each message handler is thus considered as an apart peripheral unit to some extent.

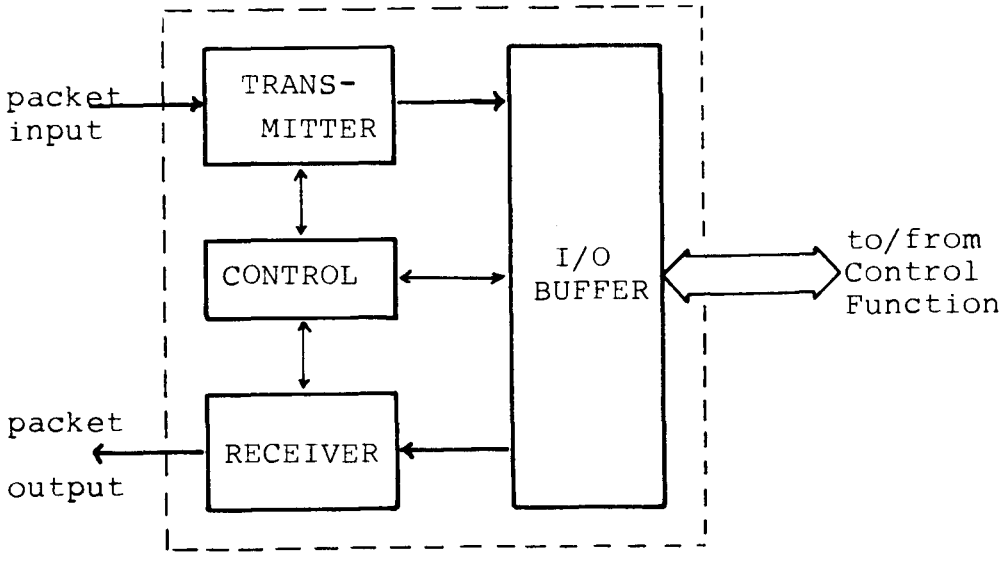


Fig.5.14 MESSAGE HANDLER

In the message handler, the receiver and transmitter portions are, to a great extent, independent units, as expected for operation over a duplex circuit.

These can then be considered separately. The receiver has as output, messages which are to be transferred to the control function, while the transmitter has, as its input, messages from the control function which are to be transferred to the extension.
The I/O buffer must therefore also consists of two semi-independent units, one for each of the two transfer directions.

A) Messages from handler to control function

This then deals with the receiver portion of the handlers plus the corresponding areas of the I/O buffers.
The global configuration is as shown in fig. 5.11.b).
In order to initiate the collection of its message, a handler which activates must generate an interrupt signal to the control function. (The message concerned must also be placed in the I/O buffer ready for transfer to the control function.)

An interrupt signal to the control function must contain two parameters:

- the interrupt indication itself;
- the identity of the interrupting source, i.e. the location from which the message can be collected.

A single interrupt line, common to all handlers, is used to supply the former parameter. This indicates that one, or more, handlers has become active.
The second parameter could be supplied by the control function itself, by initiating a scan of the status of each handler in turn. This would also isolate one handler at a time, which could then be serviced. The waste of control function processing time involved in such a software scan, is one of the main points which led to the choice of coupling via interrupts - some other technique must be used.

There are three requirements, namely:

- a single active handler must be isolated and identified;
- the process must be carried out quickly;
- all handlers must be treated equally, i.e. none have precedence.

The technique chosen to do this is as follows:

One, or more, handlers activating, leads to the signalling of this fact, via a control line, to a central interrupt controller. The controller then isolates a single handler via a type of hardware scan. This consists, as indicated in fig. 5.15, of the generation of a test signal, which is injected just after that handler which was last active (i.e. has already been serviced). This signal is then 'rippled through' from one handler to the next until blocked when contact is made with an active handler.

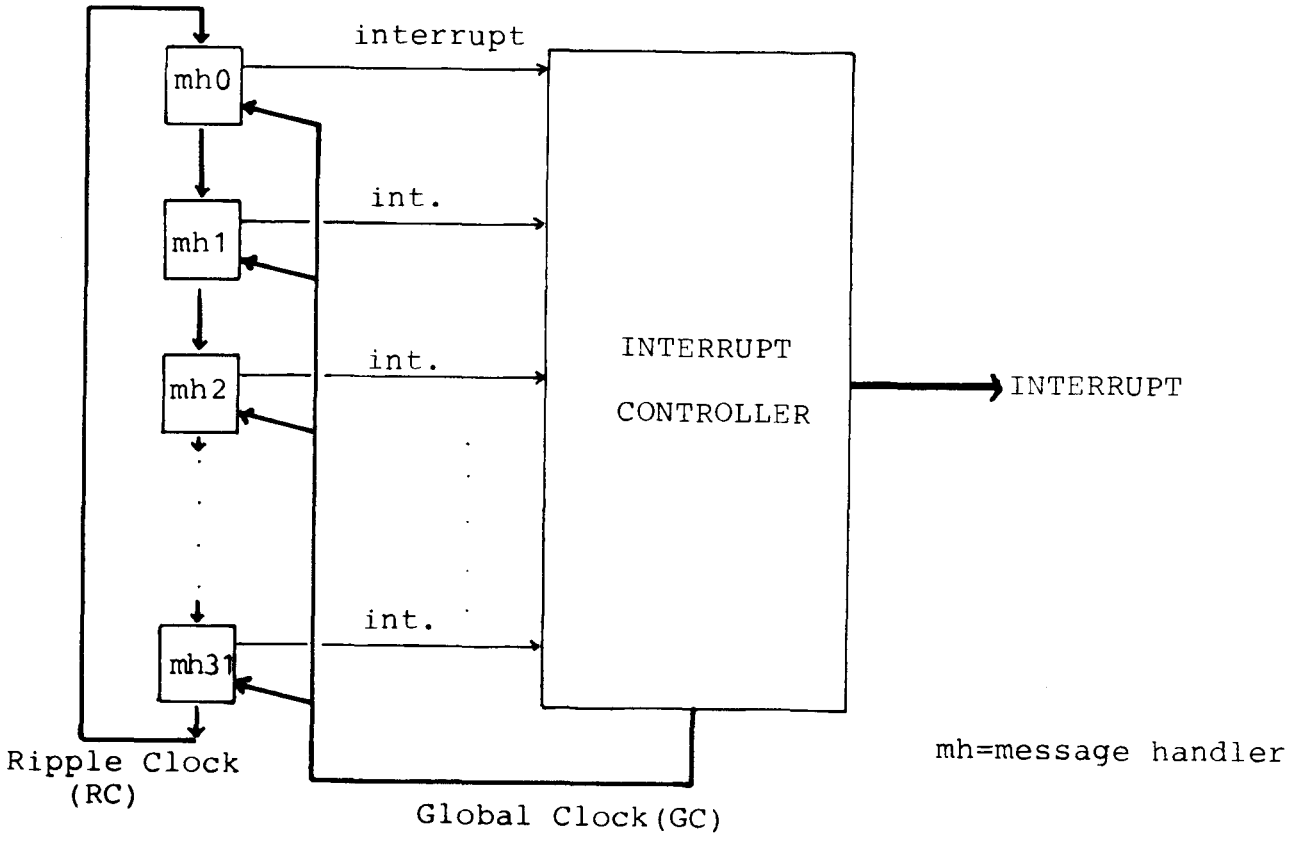


Fig.5.15 INTERRUPT HANDLING

The injection of the test signal, named Global Clock (GC), is carried out via a common line to all handlers. The last active handler then causes this signal to be entered into the 'ripple' chain at the input to the next handler in the chain.
A form of rotating priority is thus created which treats all handlers equitably, and isolates one single handler, as required.
After a delay, during which this isolation takes place, the interrupt controller generates an interrupt signal to the control function.
A bonus created by this system is that after this process is complete, the identity of the isolated handler is known within that handler. Using a single common data bus for message transfer to the control function, as shown in fig. 5.16, and allowing exclusive access to this by the isolated handler, means that the control function need merely read the message from this bus. A 'logical buffer' or 'mailbox' has thus been produced, whose fixed address is known in advance to the control function. This in turn, supplies a relatively quick transfer process with little overhead in the control function.

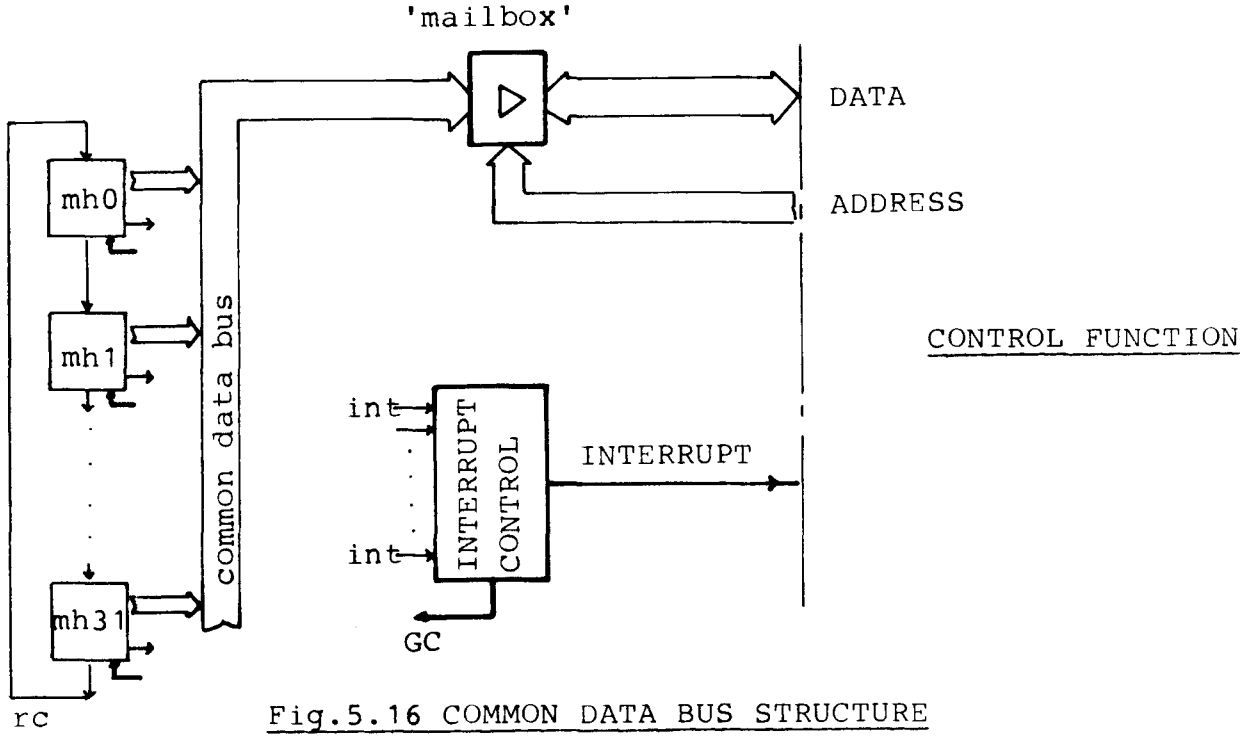


Fig.5.16 COMMON DATA BUS STRUCTURE

A more detailed block diagram of this isolation technique is given in fig. 5.17.

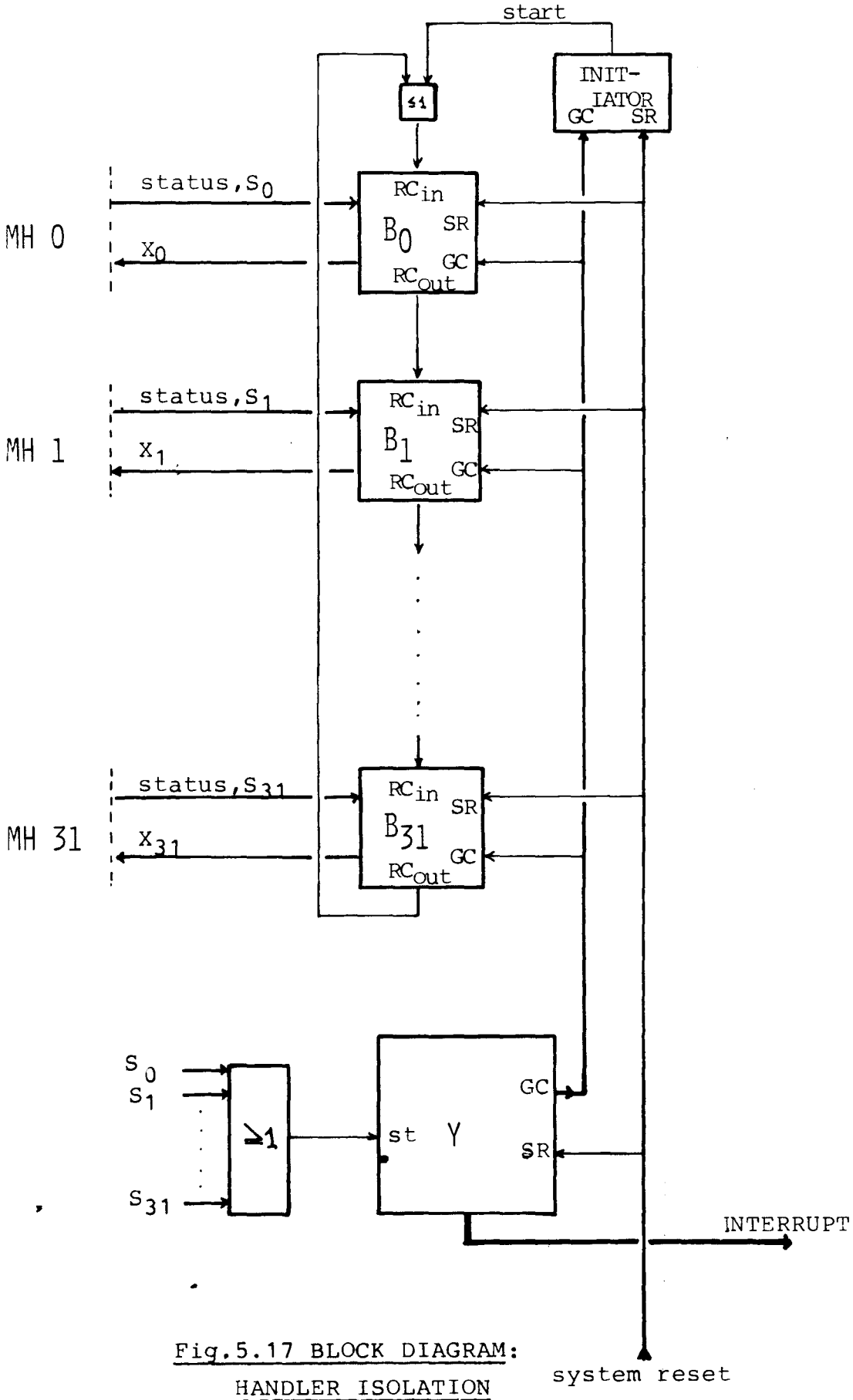


Fig.5.17 BLOCK DIAGRAM: HANDLER ISOLATION

Here the handler status is transferred to blocks B and Y. Y contains the global clock generator and carries out the interrupt procedure with the control function. Block B processes the ripple and global clocks, isolating a single handler, and returns a signal, X, to the handler indicating whether this has been granted access to the common data bus. Also included are the facilities necessary for system initiation on reset, i.e. the initiation of the priority process.

Detailed circuit and timing diagrams of the various units considered are given in fig. 5.18 to 5.22.

Messages from the extension cannot include the internal system extension number, as this is not known in the extension (for reasons of flexibility). This source information must be included in any message transferred internally within the exchange and must therefore be added to the incoming message in some way. To do this in the handler reduces hardware flexibility - a handler would then be completely tied to a given position in the system. A more flexible solution is to use the 'X' signals of fig. 5.17 to produce the identity, i.e. extension number, of the handler from which the message is sourced. This is then supplied to the control function, which combines it with the message being collected to form the complete message. A circuit diagram of this interface subunit is given in fig. 5.23.d.

B) Messages from control function to handler

This is done as described in section 5.2.3.c). Basically the status of the specified handler is tested and if found 'free' then the message itself is transferred into the corresponding area of the handler I/O buffer, for further transmission to the extension.

Fig. 5.22 shows the detailed circuit and timing diagrams of the part of the I/O buffer dealing with this area of message transfer, while the circuitry used to create the specific handler address is given in fig. 5.23b. Again, for these functions a single, common data bus is used.
The bi-directional EXTERNAL DATA BUS being split into two uni-directional buses as shown in fig 5.23a.

Component List:

Figs. 5.19,21 and 22.

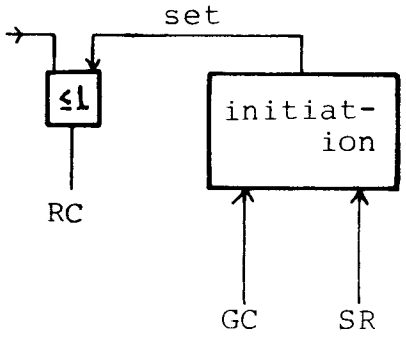
N1	;	74LS 164
N2	;	74LS 374
N3,N13	;	74LS 04
N4	;	74LS 32
N5,N14	;	74LS 08
N6	;	74LS 03
N7,N8,N9,N12	;	74LS 74
N10	;	74LS 165
N11	;	74LS 374
N15	;	74LS 11
N16	;	74LS 27
N17	;	74LS 125

Fig.5.23

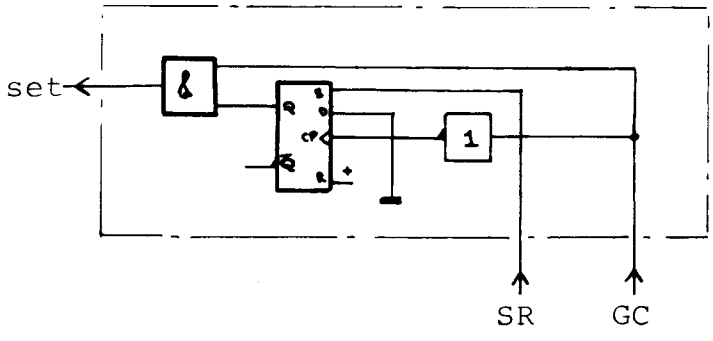
N1,N2	;	74LS 244
N3	;	74LS 08
N4	;	74LS 32
N5	;	74LS 04
N6	;	74LS 02
N7,N8	;	74LS 138
N9,N10	;	74LS 30
N11,N12	;	74LS 148
N13	;	74LS 257
N14,N15	;	74LS 125

Fig.5.20

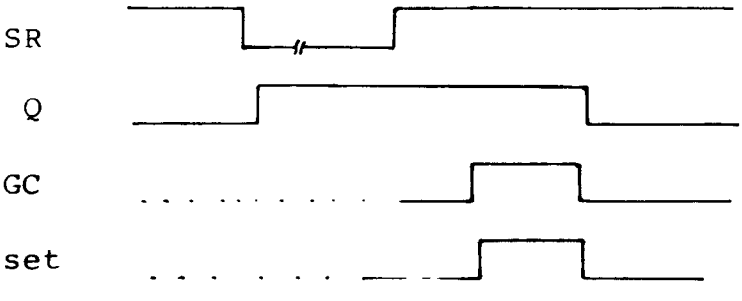
N1	;	74LS 32
N2	;	74LS 11
N3	;	74LS 193
N4	;	74LS 04
N5,N6	;	74LS 74



a) BLOCK DIAGRAM

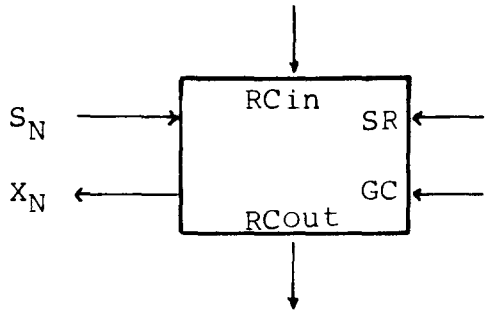


b) CIRCUIT DIAGRAM

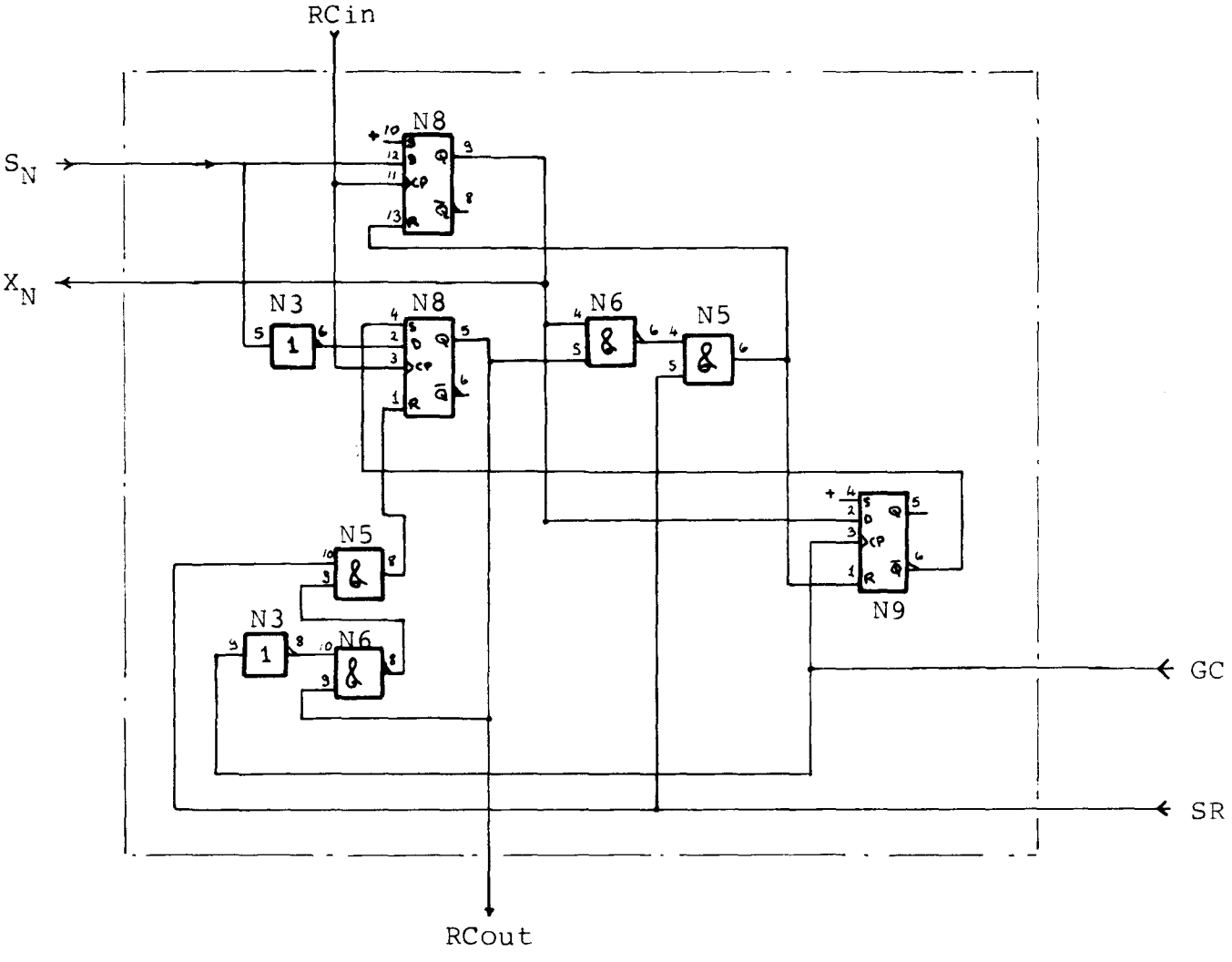


c) TIMING DIAGRAM

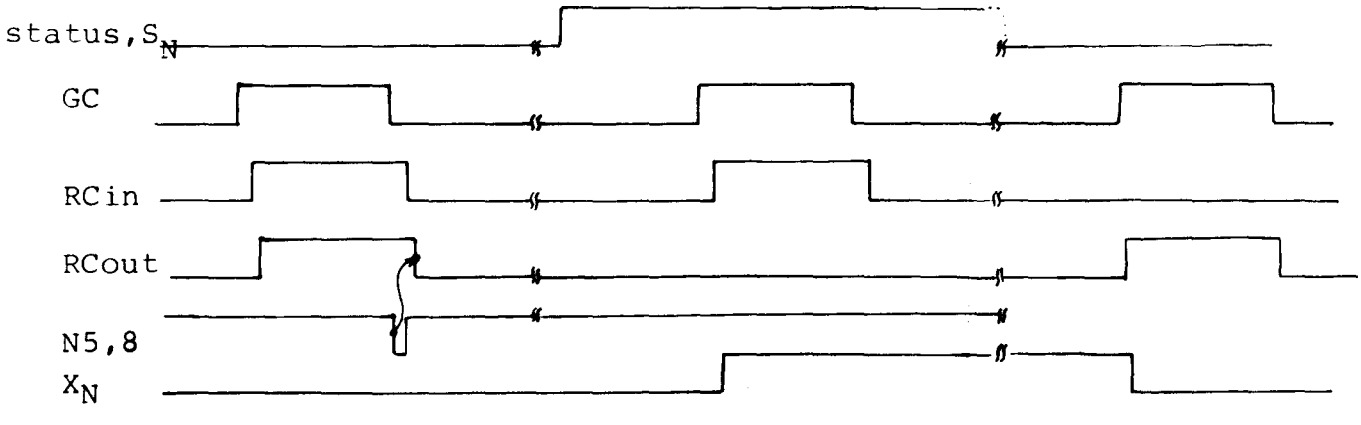
Fig.5.18 INITIATION UNIT



a) BLOCK DIAGRAM

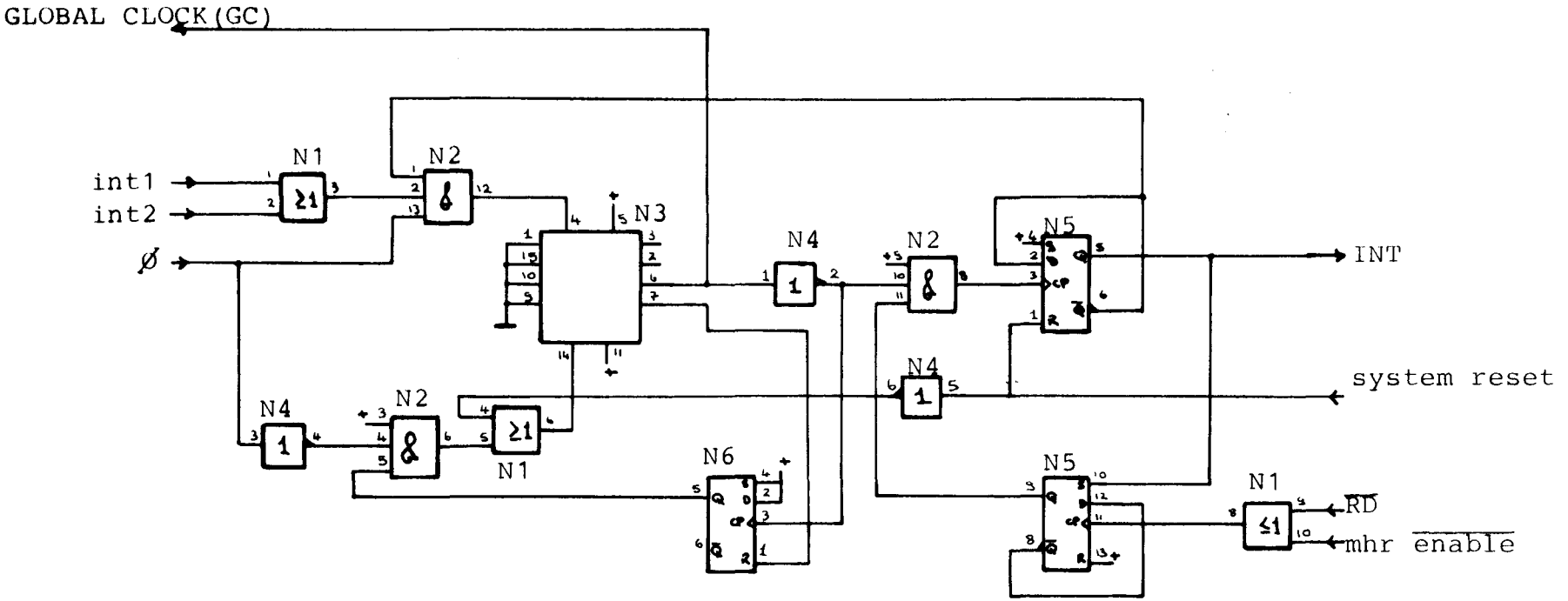


b) CIRCUIT DIAGRAM

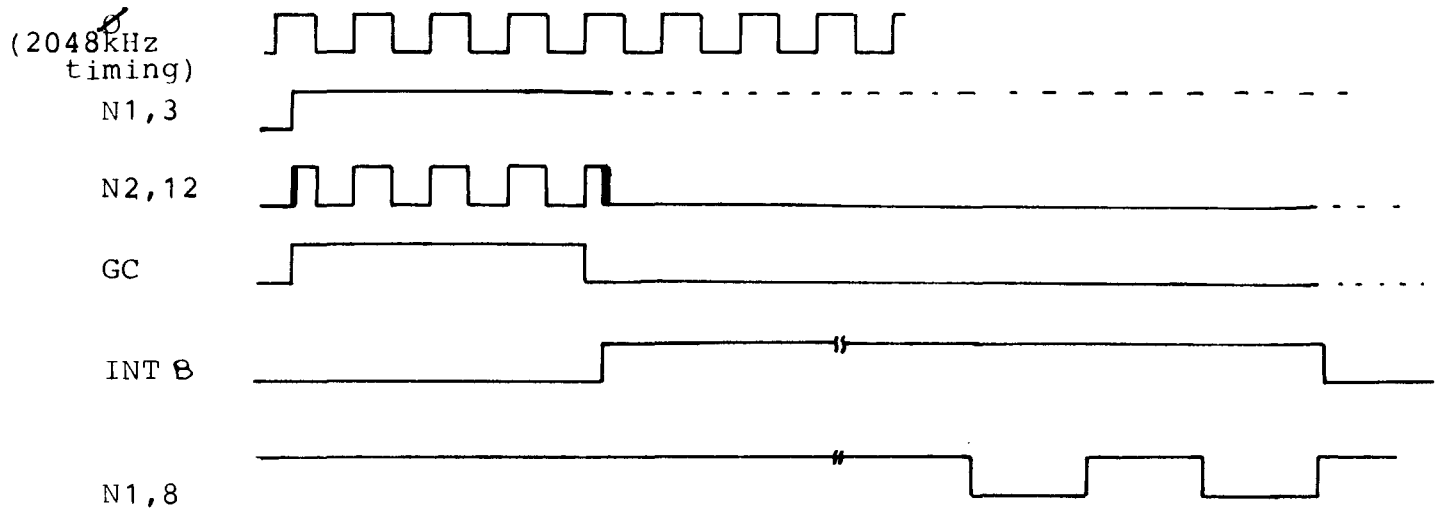


c) TIMING DIAGRAM

Fig.5.19 'B' UNIT



a) CIRCUIT DIAGRAM



b) TIMING DIAGRAM

Fig.5.20 'Y' UNIT

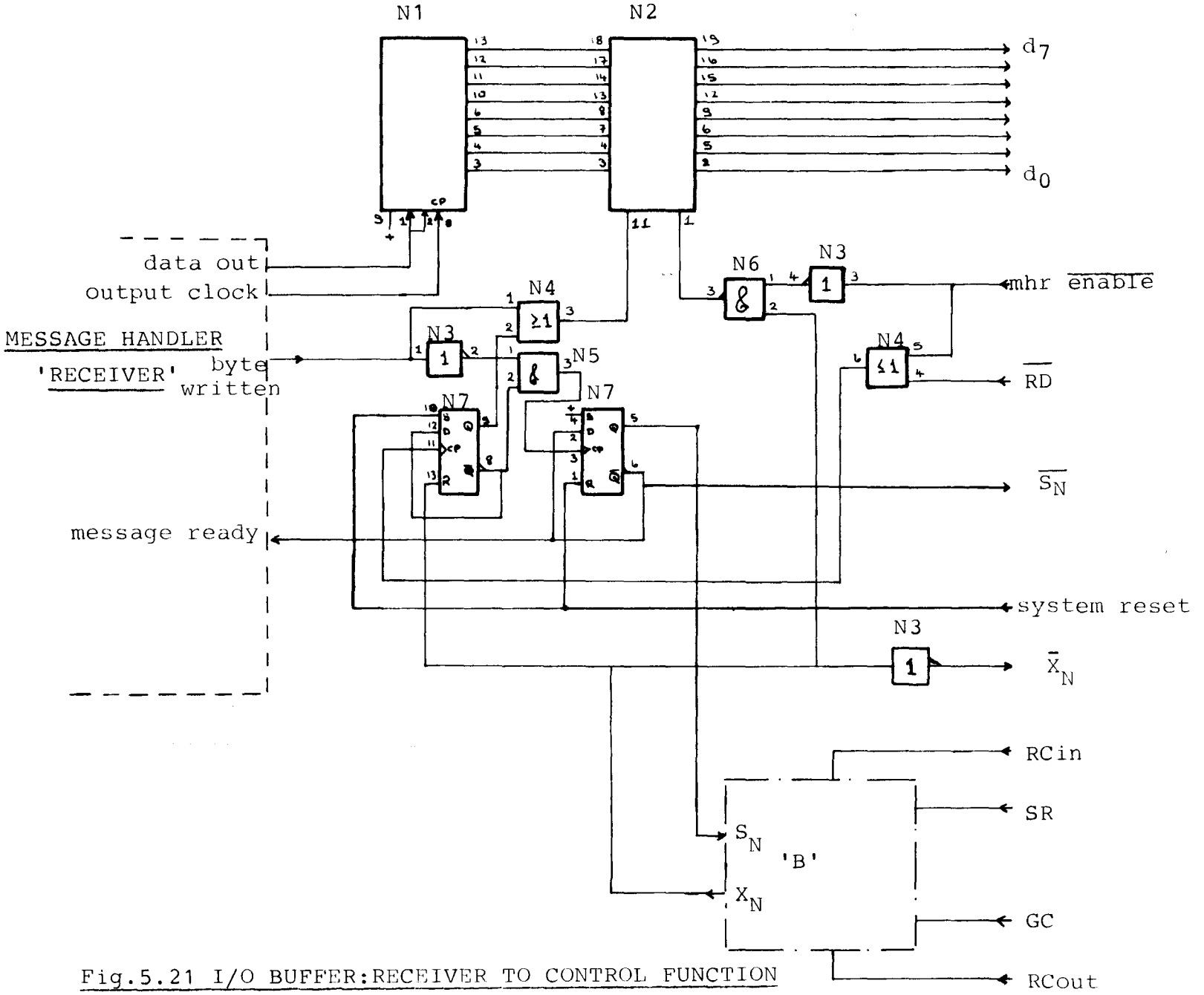
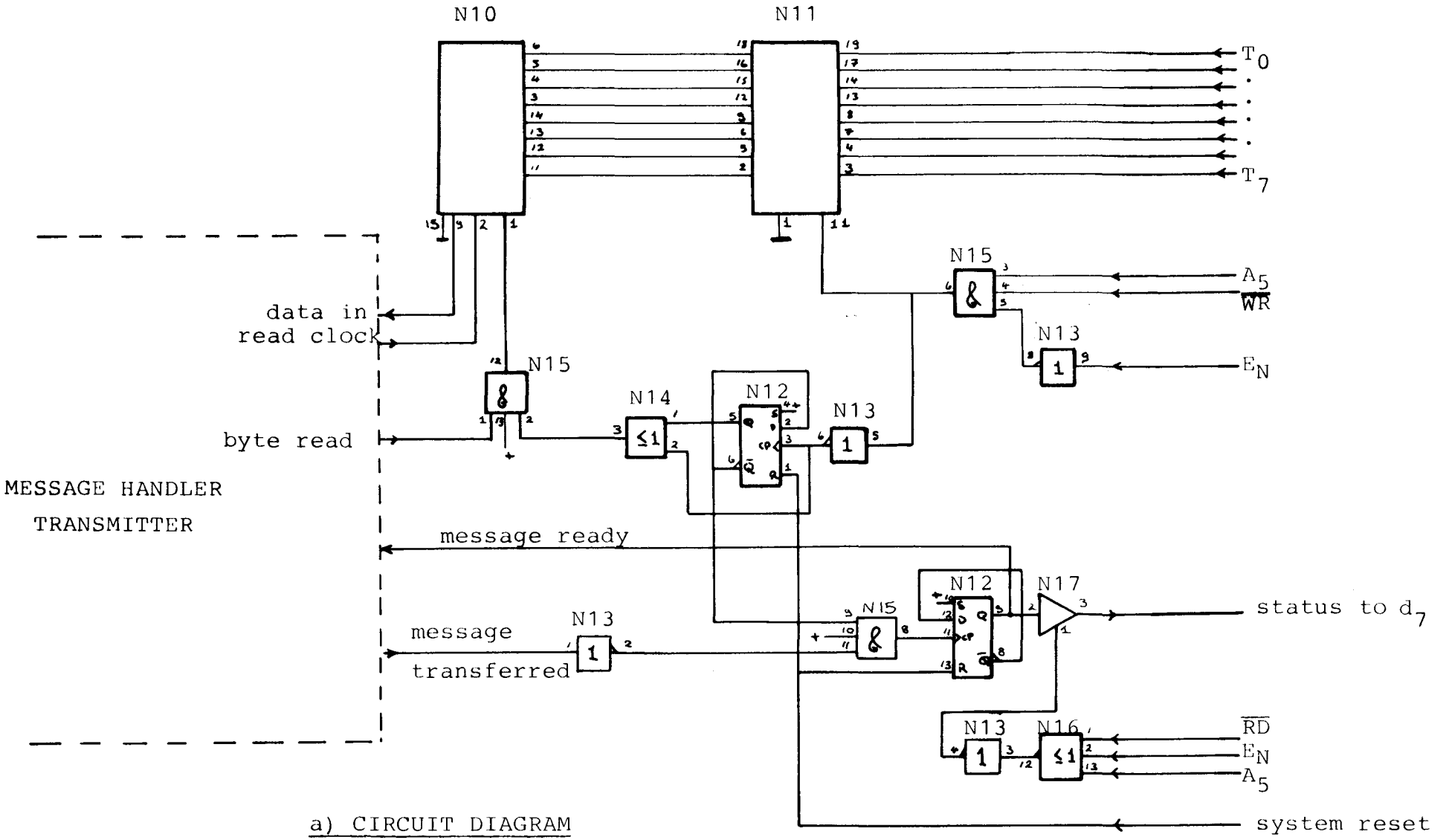


Fig.5.21 I/O BUFFER:RECEIVER TO CONTROL FUNCTION



a) CIRCUIT DIAGRAM

Fig. 5.22 I/O BUFFER: CONTROL FUNCTION TO TRANSMITTER

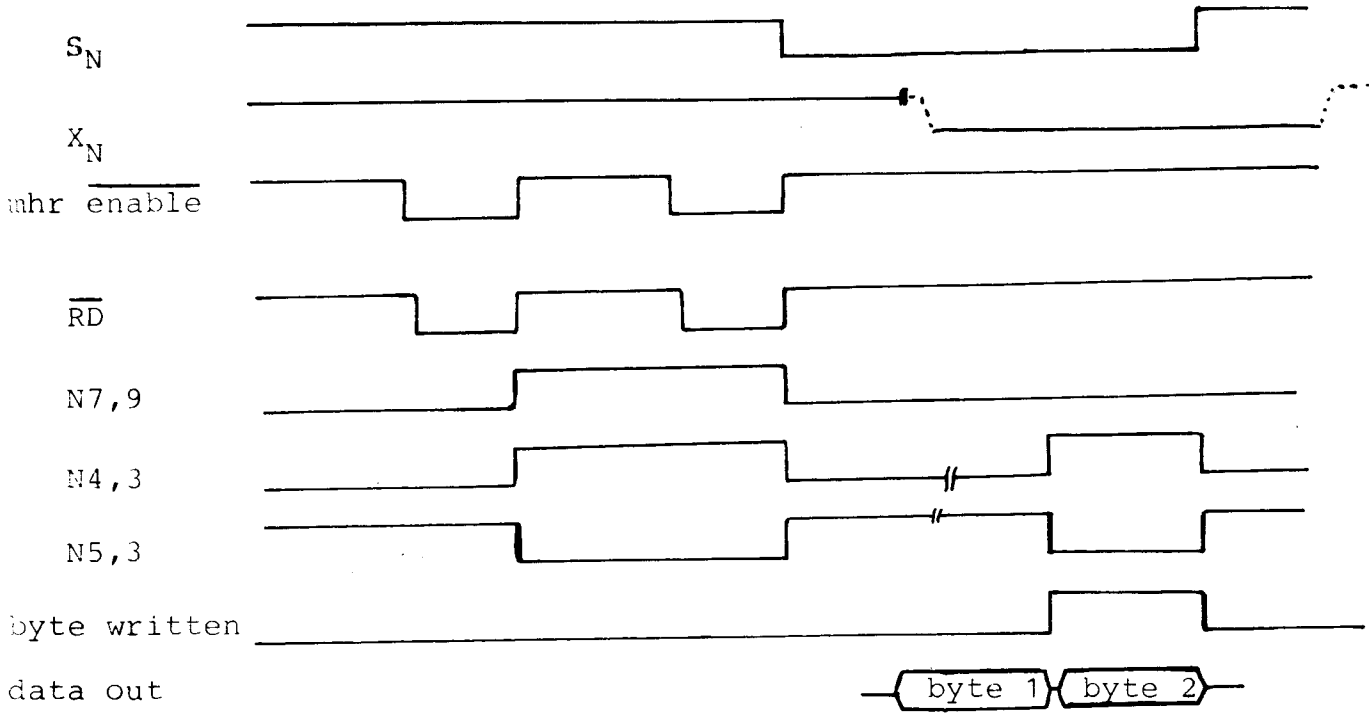


Fig.5.21.b TIMING DIAGRAM

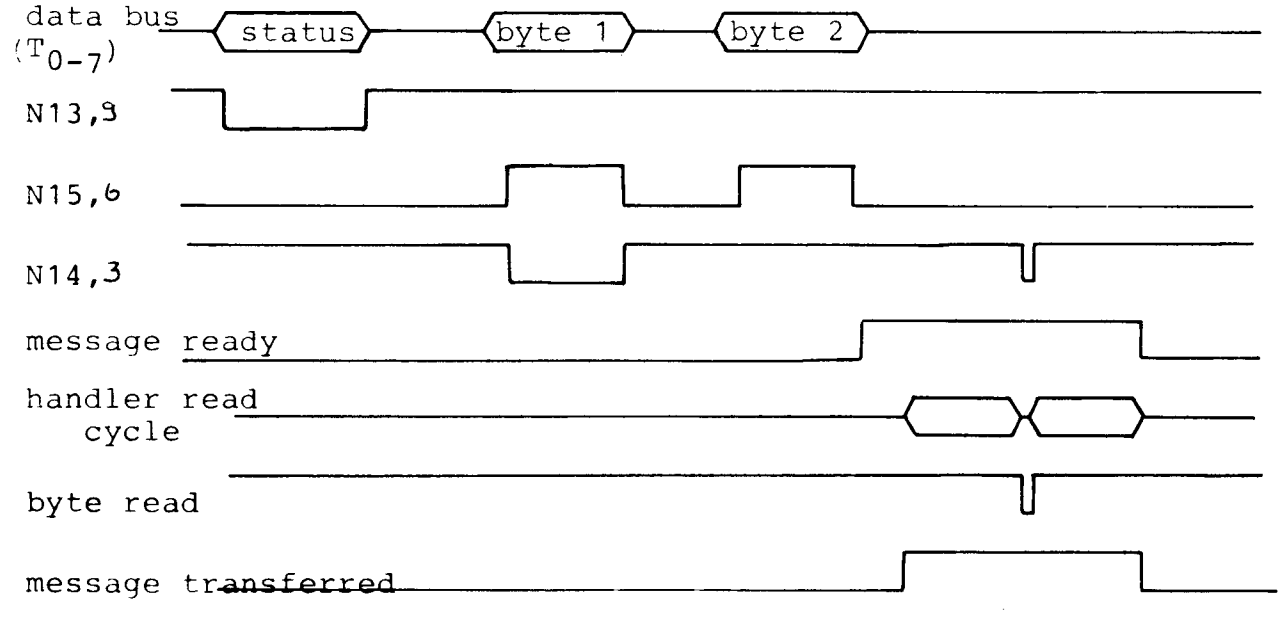


Fig.5.22b) TIMING DIAGRAM

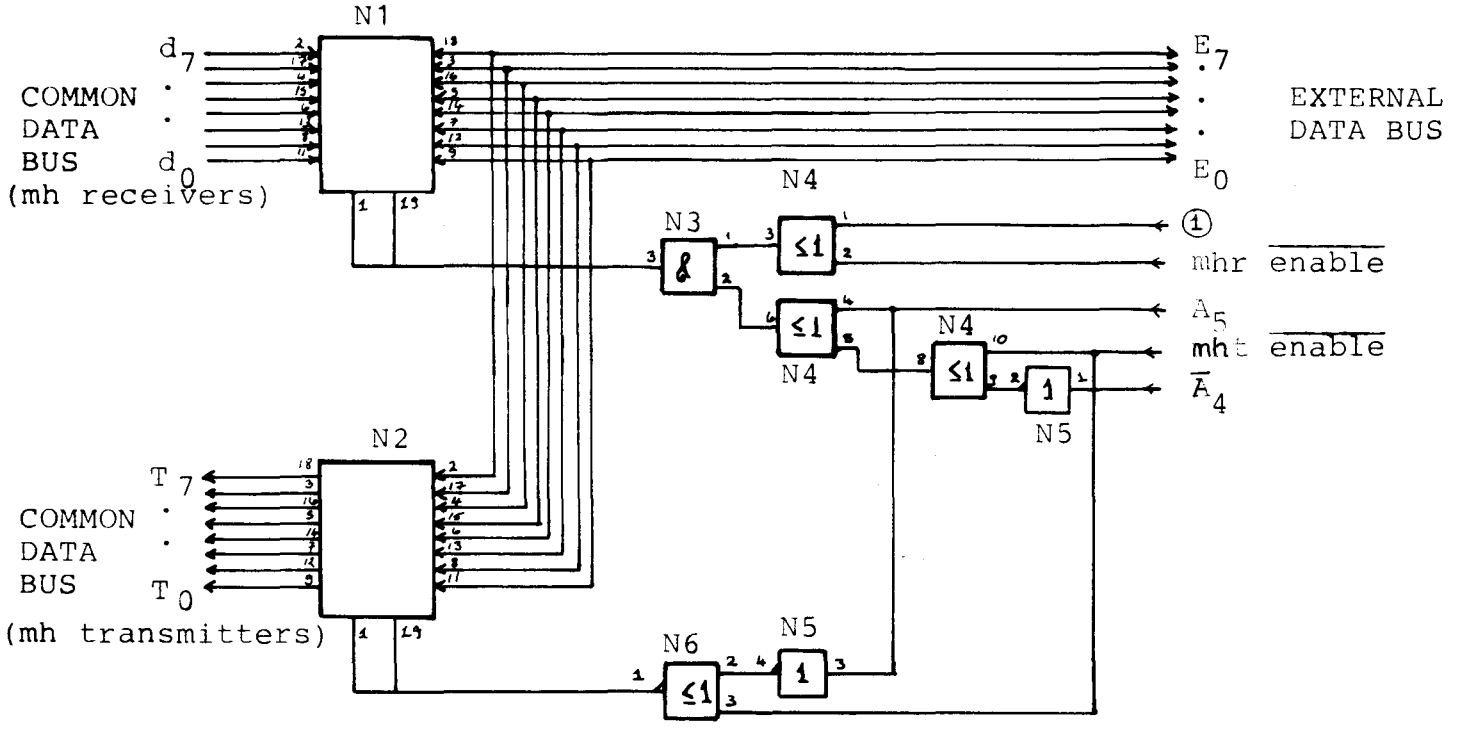


Fig.5.23a) DATA BUS CONNECTION:CIRCUIT DIAGRAM

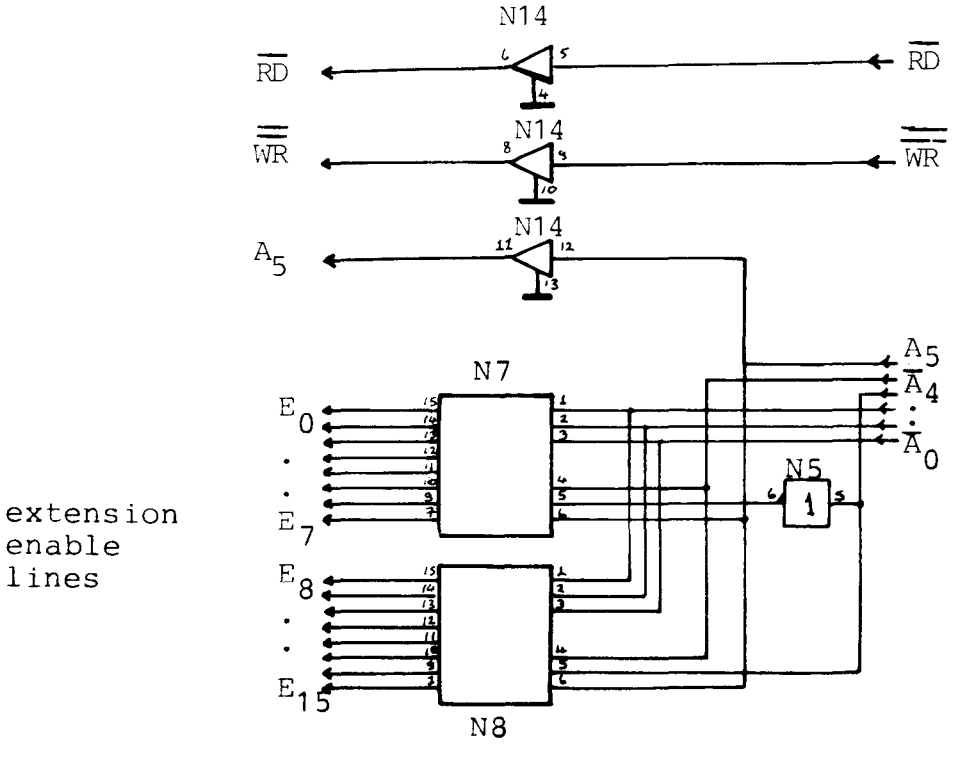


Fig.5.23b) ADDRESS DECODER/CONTROL BUS INTERFACE

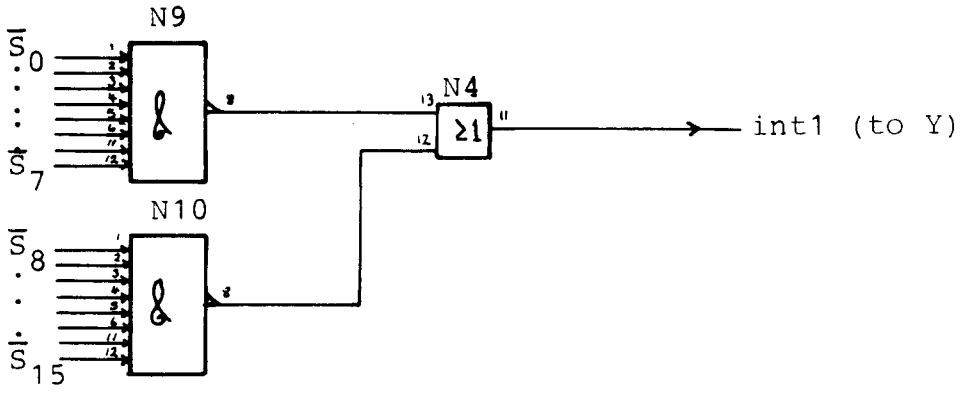


Fig.5.23c) COMMON INTERRUPT GENERATOR

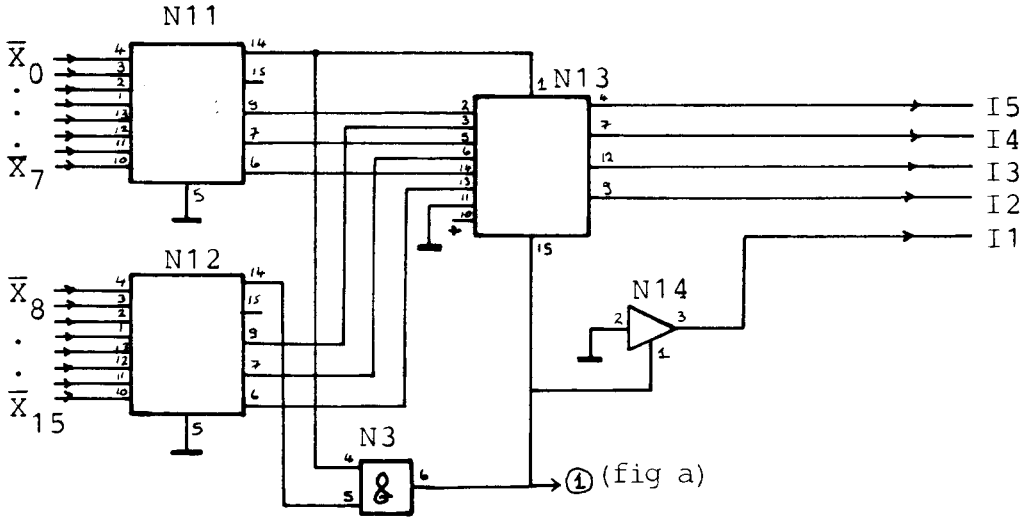


Fig.5.23d) ACTIVE HANDLER IDENTIFICATION UNIT

As will be described in chapter 6, the hardware connections with the control function are:

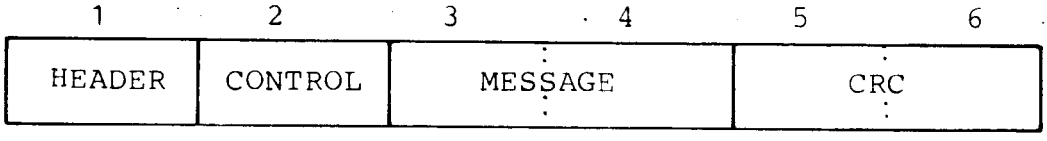
- the internal data bus of the control function extended to the peripheral units (The EXTERNAL DATA BUS);
- address lines in the form of function enable lines plus the low order bits of the address bus (EXTERNAL DATA BUS);
- certain control lines (EXTERNAL CONTROL BUS);
- interrupt and source identification signals.

Due to the volume of hardware used, the system is to be mounted on 3 separate printed circuit cards. Two of these cards (as seen in fig. 7.1) contain subscriber line message handlers, while the third contains the control function. The hardware configuration used to interface these cards with the control function, drive the various connection lines between them, and etc., is shown in figs. 5.23a to 5.23d. Block Y is placed on that printed circuit card which contains the control function, this was chosen, as well as other parts of the interface hardware configuration, in order to keep the two cards containing the 'subscriber circuitry' as identical as possible - with regard to mass production.

5.3. Inter Processor Signalling Circuit

This carries the concentrated message transfer between FEU and CPU in the main body of the exchange.
As the requirements for this transfer are very similar to those for the corresponding subscriber line signalling circuit, the same link protocol will be used. Processing of the messages carried out in the CPU consists, in fact, of the expansion of that carried out in the control function of the FEU.

The packet used to transfer the message is slightly different, as shown below.



PACKET FORMAT (IP)

Here, as the message also includes the extension number, it is 2 bytes long, and a separate byte is dedicated to the control field. Also the integrity of this internal message transfer within the system, should be better than that for the subscriber signalling. To accomplish this a 16 bit CRC is used (in place of 8).

Messages are transferred between control function and the IP message handler directly. The interface between handler and the IIB is driven by an enable signal from the user circuit management function, as described in chapter 4. The signalling circuit consists of a dedicated timeslot in each frame on the IIB - a packet is therefore transferred consecutively in six of these dedicated timeslots.

As in section 5.2.2 an approximation of the message throughput of this circuit must be made:

basic circuit capacity = 64 kb/sec.
message length = 15 bits (message + user identity + ext. number)
Maximum throughput = 4300 messages/sec.

packet length = 48 bits
absolute maximum Signalling Rate; SR_{amax} = 1333 packets/sec.

and, due to repetition effects, SR_{max} ; 444 < SR < 667 packets/sec.

Repeating the calculations of section 5.2.2, regarding the effect of errors on this signalling rate leads to:

p	P_0	P_1	P_2	P_3	P_m
0,0001	0,999	0,005	-	-	0,005
0,001	0,953	0,046	0,001	-	0,047
0,01	0,617	0,300	0,071	0,0011	0,383

here $P_0 = (1-p)^{48}$

$P_1 = \binom{48}{1} p(1-p)^{47}$ etc.

It is seen from this that, due to the longer packet, the effect of errors is somewhat higher than that for the subscriber signalling circuit.

The worst case SR_{max} is found to be > 350 messages/sec. This is considered adequate for the expected loading of the system. Again synchronisation of the transmitter/receiver combinations could be used to increase this, if necessary.

5.3.1. Implementation

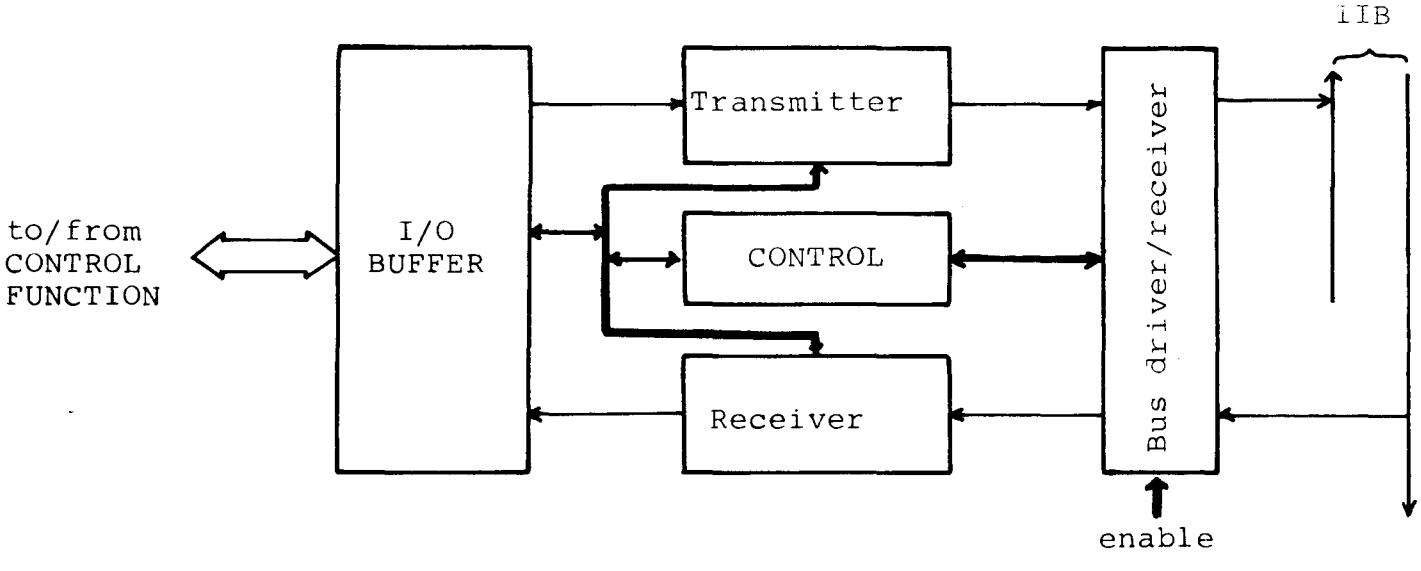


Fig. 5 Inter-Processor Message Handler

This consists, as shown in fig. 5.24, of an I/O buffer for connection to the control function, the typical receiver/transmitter combination for implementation of the link protocol and a bus driver/receiver unit (driven by an enable signal from the user circuit management function). Of these only the I/O buffer will be considered in further detail.

A received message is deposited in the buffer, and an interrupt signal is generated to the control function. This leads eventually to the message being collected. Messages from the Control Function are transferred after a status test of the unit has shown the transmitter to be free.

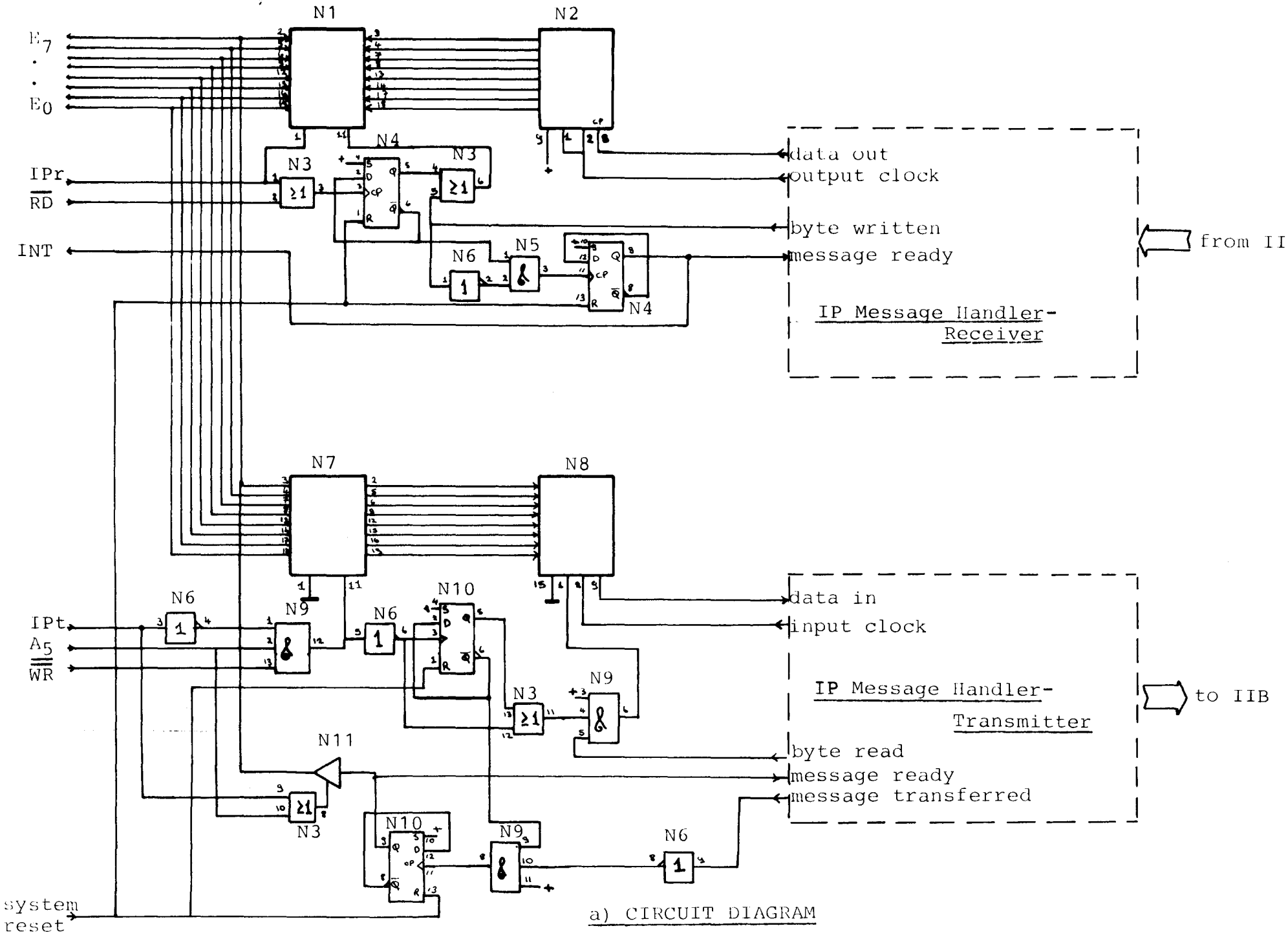
This unit is considered apart from the subscriber signalling circuits due to its intense usage. Also, for this reason, the interrupt/addressing facilities between the unit and control function are implemented separately from the rest of the signalling units. Circuit and timing diagrams of this I/O buffer used, are given in fig. 5.25.

5.4. Integrated Access Protocol

5.4.1 General.

The description of the signalling system used has now been completed. This supplies a medium in which messages can be transferred reliably between user and exchange.
Little consideration has been given to the actual meaning of the events which caused the generation of these messages as this was considered to be the responsibility of a higher layer in the hierarchy.

The object of these messages is to create, maintain and support a connection between users. In order to do this the messages which are transferred must occur in an ordered sequence, e.g. a user requests a connection and only after the acceptance of this request by the exchange does this 'calling party' identify the required 'called party' via dialling ciphers.



a) CIRCUIT DIAGRAM

Fig.5.25 INTER PROCESSOR SIGNALLING:I/O BUFFER

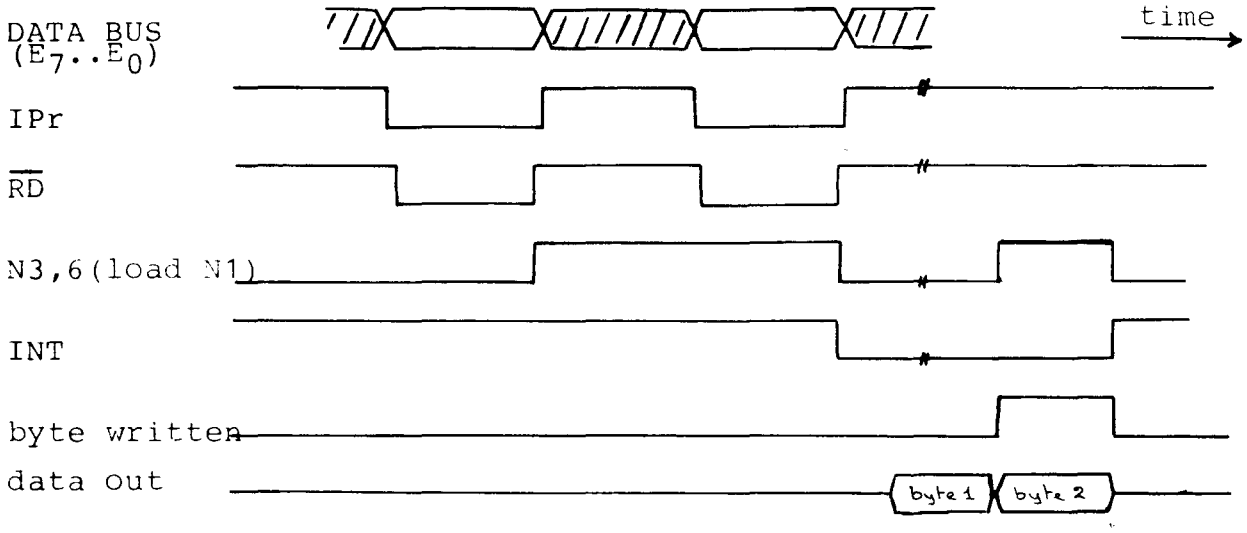


Fig.5.25b) TIMING DIAGRAM:RECEIVER

Timing for the transmitter is identical with Fig.5.22.b.

Similarly, only a given set of all possible messages have any meaning during a particular phase of a connection - in the above identification sequence only such messages as dial ciphers from the calling party are significant.

Basically the third layer of the signalling structure of fig. 5.3 deals with the ordering of these events. Simple handshake techniques are used to accomplish this - in general any given event generated by a user leads to a reply (from a small set of possibilities) from either the exchange or far end user. The users here can be any of the four units described in chapter 3.
An attempt has been made to create a message vocabulary covering all the events which can occur within these user units. Furthermore this single set is applicable in general to each of the units. This results in the messages described by fig. 5.26.
This figure gives an example of the message sequence which can occur in a typical connection. Here two users, A and B, are connected with an exchange, or chain of exchanges, X-X'. The direction of message flow over the signalling circuits, A-X and B-X' respectively, are indicated by arrows. A message generated by a user penetrates the system, directly, no further than the first exchange. This in turn leads either to a reply being transmitted to the user or some corresponding message to the far end user. To this message from the user, several possible replies can be made and in general the handshake, i.e. message/reply, is completed. The message sequence shown is not exhaustive, e.g. a 'clear request' message can occur at any time, from either party.

As is seen from fig. 5.26 three distinct phases occur:

I) System activation/deactivation: this is purely concerned with the state of the subscriber line and common equipment connected to it.

II) Circuit build up/break down: this is concerned with the creation, or etc., of the required connection between one user and the other.

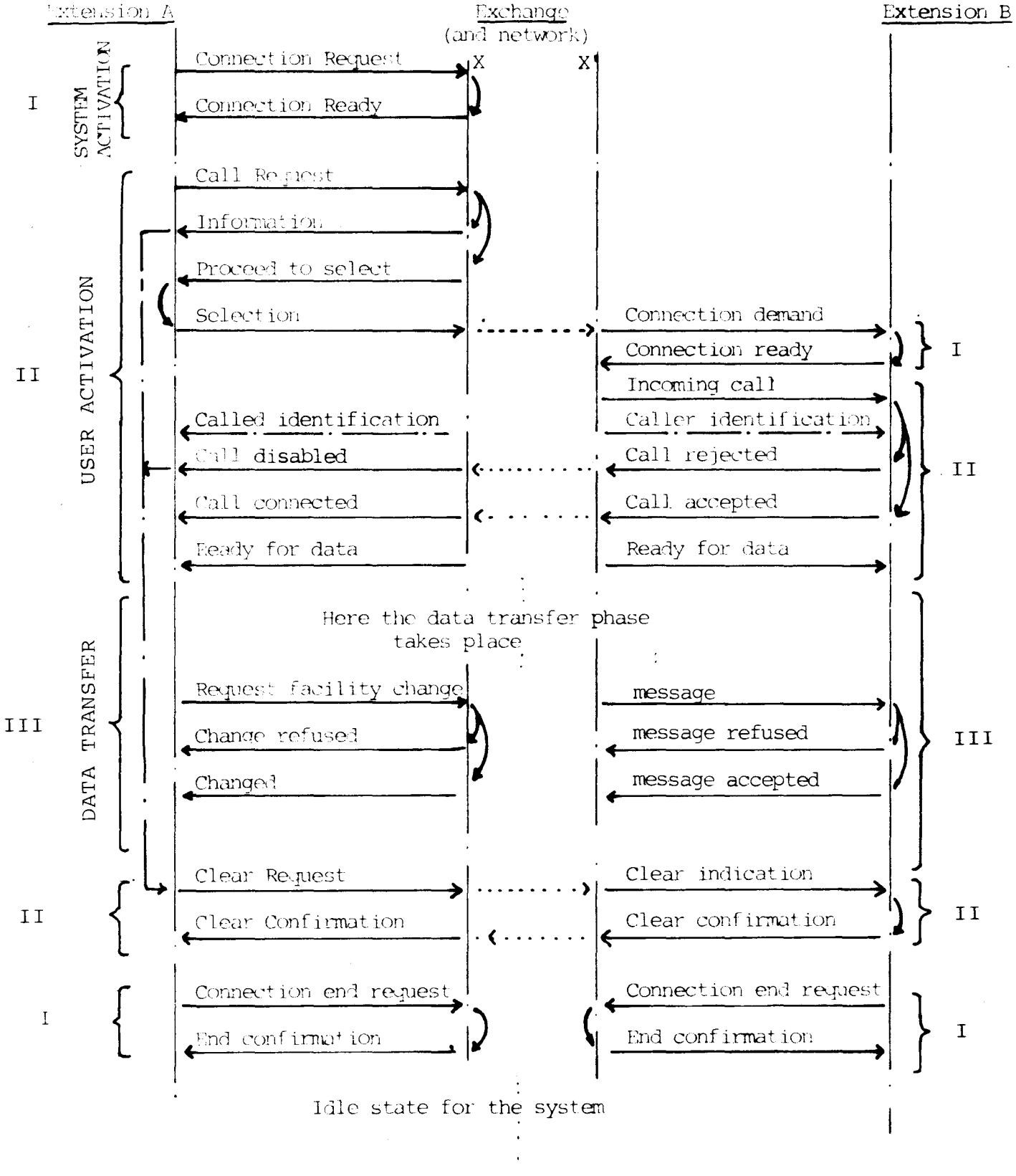


Fig.5.26 INTEGRATED ACCESS PROTOCOL:
GLOBAL MODEL.

III) Data transfer phase: in which the actual end to end information transfer (via the user circuit) between users takes place.

The detailed actions, arising from any given message, in the FEU will be dealt with in chapter 6, these actions leading to the various Application Programmes described there. In general one of two actions are taken by the FEU after message processing:

A) A reply is transmitted to the user. This is typical in the system (de)activation phase, where the message is concerned only with the user-FEU region.

B) A derived message is transferred to the CPU in the main body of the exchange, for further processing.

This latter leads eventually to some reply from the CPU, which is then processed in the FEU - after which a derived message is transferred to the specified user.

5.4.2. Handshake Techniques

This reply to a message generated by a user, completes the handshake procedure used to order the occurrance of events, the next message may then be transmitted. This process is shown for cases A and B in fig. 5.27.

As is seen from these diagrams the delay occurring between generation of a message and receipt of a reply, especially in case B, is relatively long. As a user should, under normal conditions, have received a reply within manual reaction times, i.e. ± 65 msec., (corresponding to 15 messages/sec.) this leads to stringent time limits within the system (in particular the main body in B).
This difficulty is most likely to occur during the dialling period of the connection build up phase. If necessary it would be possible to consider all of these dialling ciphers as a single 'complex event'.

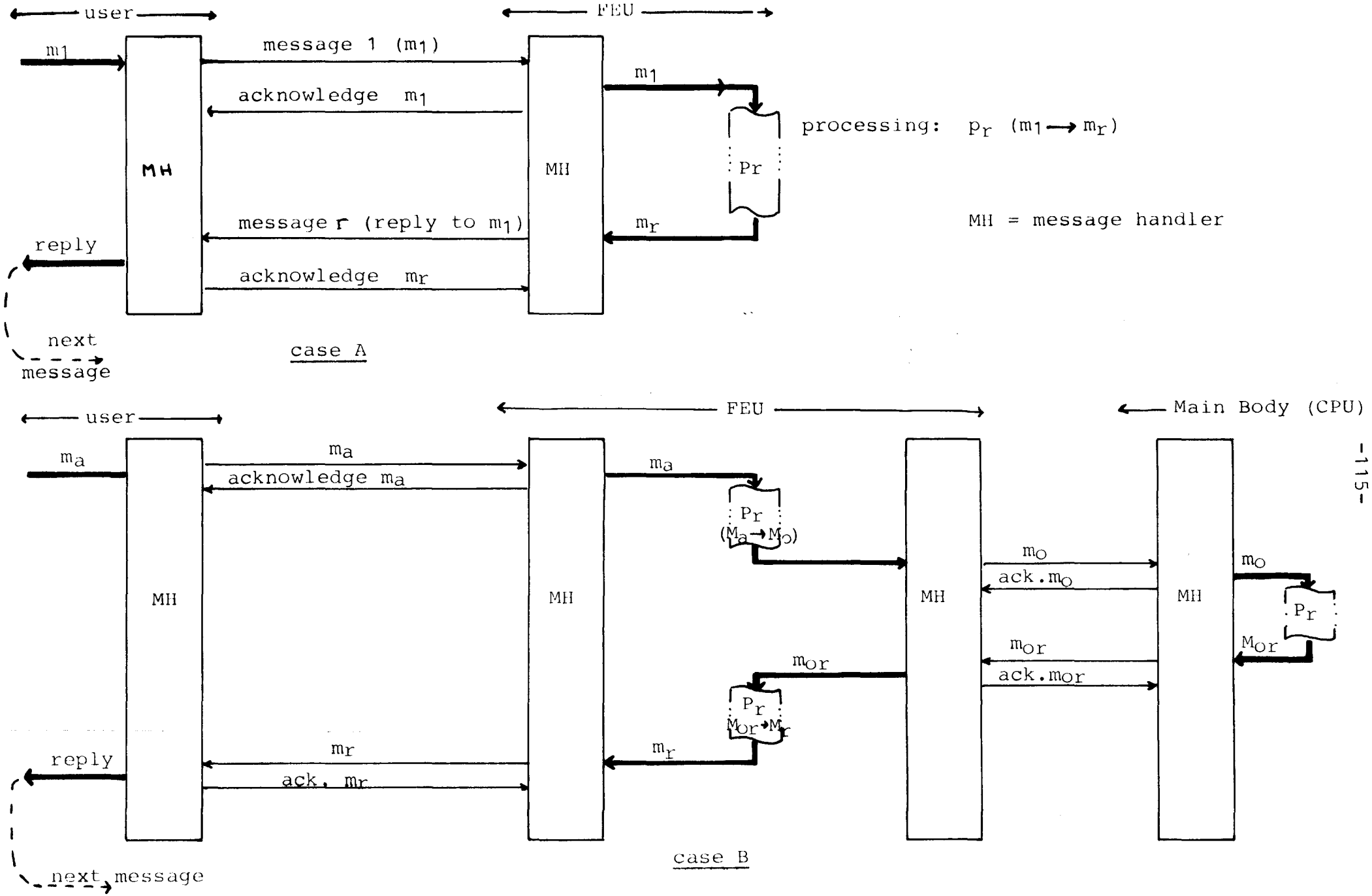
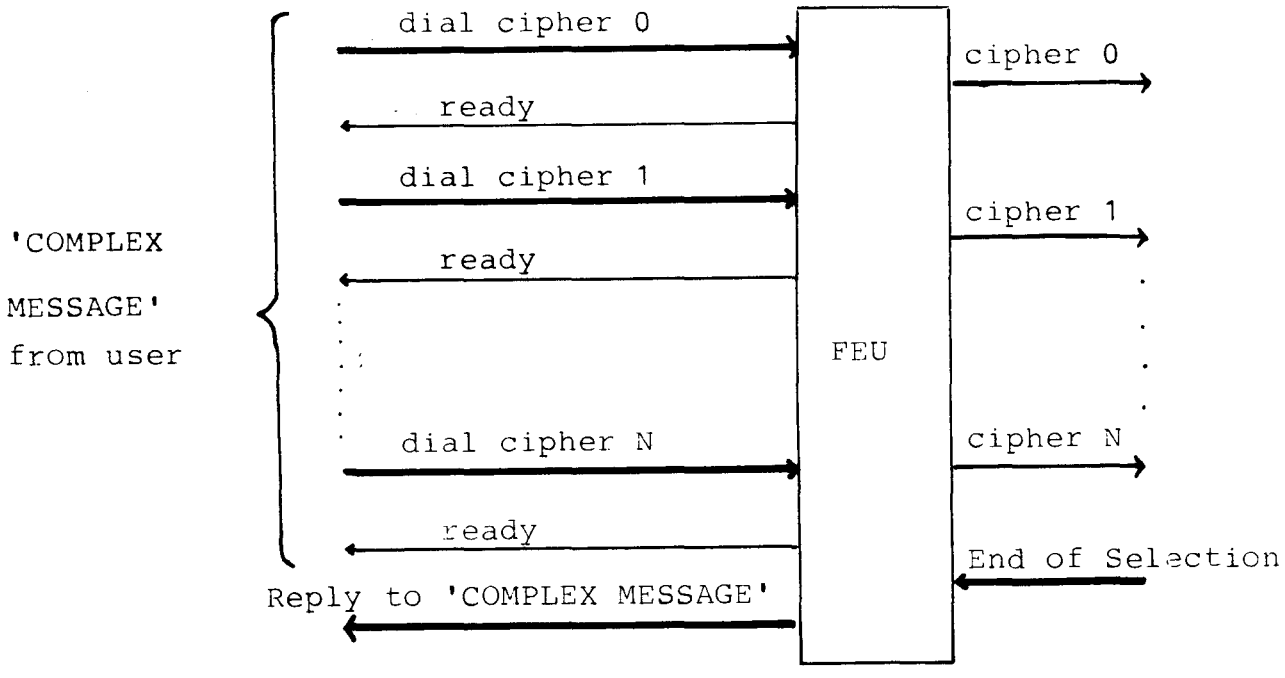


Fig. 5 HANDSHAKE PROCEDURE

Each individual cipher could be answered by the FEU with a 'ready' indication - allowing the next cipher to be transmitted while, from the main body of the exchange, a reply to the completion of the dialling phase could be made.



In this way the FEU would serve the required 'quick reaction' processes while the main body would serve the slower reaction requirements. The disadvantage here, is that the system has no longer full control over its operation, and special measures might have to be taken to overcome any possible difficulties which could arise, e.g. by the numbering of messages. The implementation will be based on the former method but should circumstances dictate the latter could be used.

This then completes the consideration of the signalling system. Messages can now be transferred between the elements of the system in a reliable and ordered manner. The actual coding of these messages will be considered in combination with the software of the system in the following chapter.

6. CONTROL FUNCTION

6.1 INTRODUCTION

The control function has as its main task, the support and management of the user circuit connections occurring in the system. In order to fulfil this requirement, the control function must:

- support the message exchanges with subscriber station and the CPU;
- process the messages received from these sources, derive the sequence of actions specified by the message and implement these actions;
- co-ordinate and implement the various elements of the FEU.

The Integrated Access Protocol, discussed in chapter 5, describes globally, the sequence of messages which lead to, and arise from, these connections. The detailed actions made by the control function, for each message, are described in the subsection on application software, while the general methods of message input to, and output from, this control function are described in the section covering system software. The hardware used to implement these actions at a physical level is considered first.

6.2. Control Function Hardware

6.2.1 Configuration

Fig. 3.6, repeated in fig. 6.1, indicates the basic hardware requirements of the control function namely, a processor, a memory unit, a means of interfacing with the peripheral units (I/O buffers) and the necessary connections between these elements.

This specific, conventional, configuration is not ideal for the system being considered due to the large number of I/O buffers occurring - which also leads to a large number of interconnection lines. Although the principal of operation remains the same, the actual implementation chosen was the result of the sequence of conversions shown in figs. 6.2 to 6.5. This leads to a reduction in the amount of interconnecting and decoding hardware necessary.

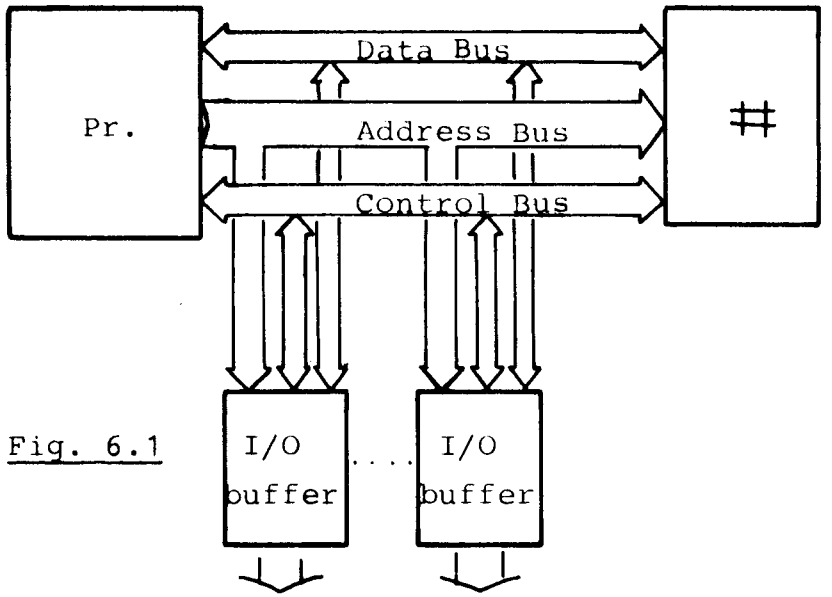


Fig. 6.1

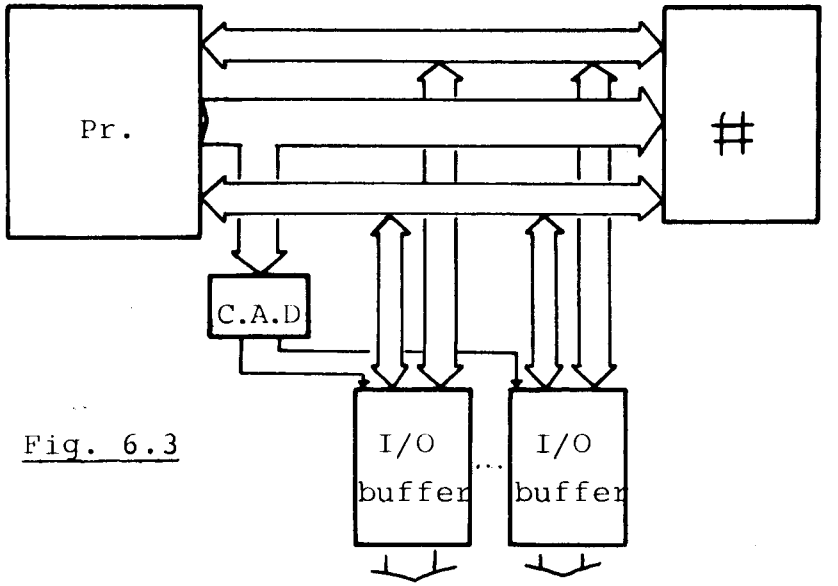


Fig. 6.3

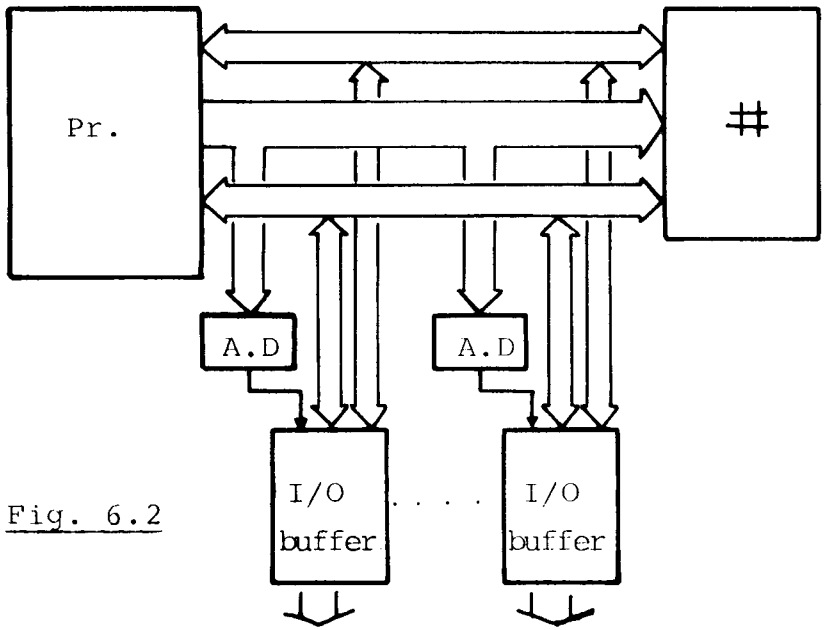


Fig. 6.2

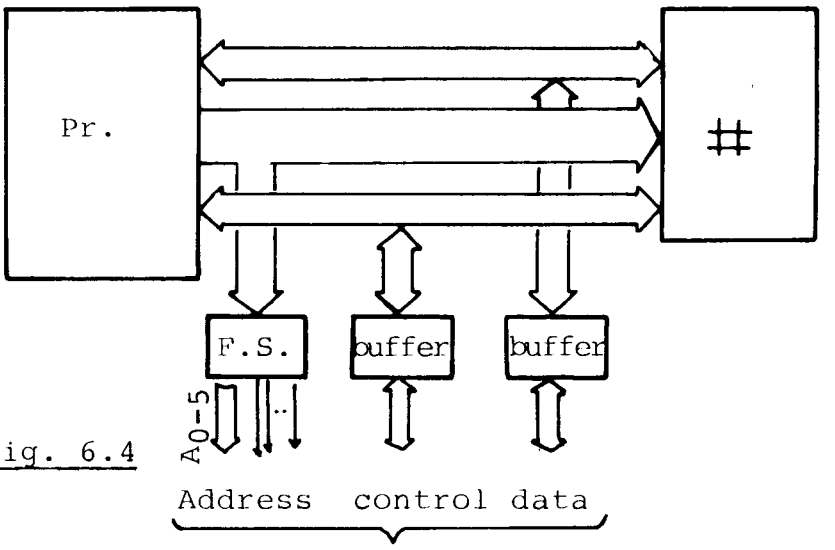


Fig. 6.4

FS = Function Selector
Pr = Processor
# = memory modules
(C)AD = (Combined) Address Decoder

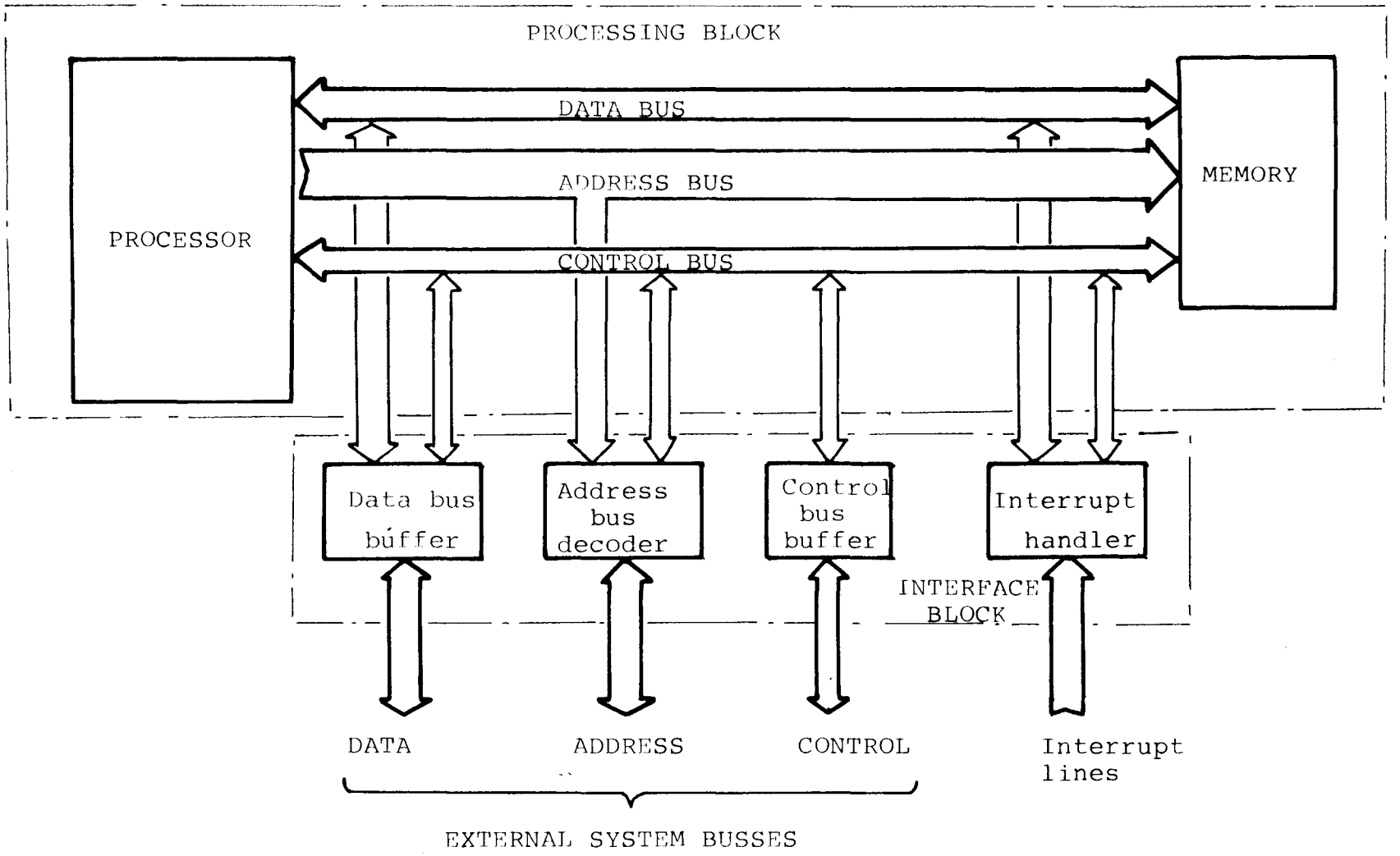


Fig. 6.5 CONTROL FUNCTION

The first step, as shown in figs. 6.2 and 6.3, was to extract the address decoders from the I/O buffers and to combine these into a single unit - a function selector. The I/O buffers were then shifted from the control function into the peripheral units themselves. Messages transferred to and from the processor occur via the Data Bus - this must then be extended, via a driver/receiver, to the peripheral units, as must the processor control lines. All peripheral units are then connected to these common Data and Control busses while access is obtained via the select signals, from the address decoder in the control function (each buffer has an output line dedicated to it). This results in the control function configuration of fig. 6.4.
The peripheral units must have a means of communicating with the processor in order to synchronise the message transfers between them. As described in chapter 5, this is done on a interrupt basis. To support this communication an interrupt handling unit has been included in the control function/peripheral interface block. This, and the previously considered units, is shown in fig. 6.5 - the block diagram of the control function hardware.

6.2.2. Implementation

The hardware can be seen to consist of two main regions, namely:

A) Processing block.
B) Peripheral Interface block.

The implementations of these will now be considered.

A) Processing Block

This consists of a Z80 microprocessor, operating with a 2048 kHz clock, plus the RAM/ROM memory modules used for data respectively programme storage. Connections between these and the interface block are made via the normal Z80 address, data and control busses.

The timing of the various actions within the hardware of the control function is therefore largely dependent on that of the Z80 itself, as described in reference .

Memory mapped I/O techniques are to be used throughout, i.e. each peripheral unit is seen as an area of memory. This area can consist of several actual memory locations, depending on the I/O buffer construction - separate locations being dedicated to input, output and status buffers.
A memory map for the function is given in fig. 6.6. As is seen from this, those locations concerning peripheral devices are distinguished from those within the memory module by the most significant bit of the address (A_{15}). This means that one half of the basic 64 kbyte address space of the Z80, has been dedicated to these peripherals. As the expected programme memory requirements are unlikely to be excessive this is the simplest solution. Of the remaining 32 kbyte address space only 10 kbyte has been used (2k RAM, 8k ROM), the rest being available for future software expansion.
Fig. 6.7 includes the actual realisation of this block. As is seen from this figure, selection logic is used to activate the specific memory chip within which the required address is situated. As the Z80 bus driving capabilities are extremely limited, bus drivers have been included in this configuration.

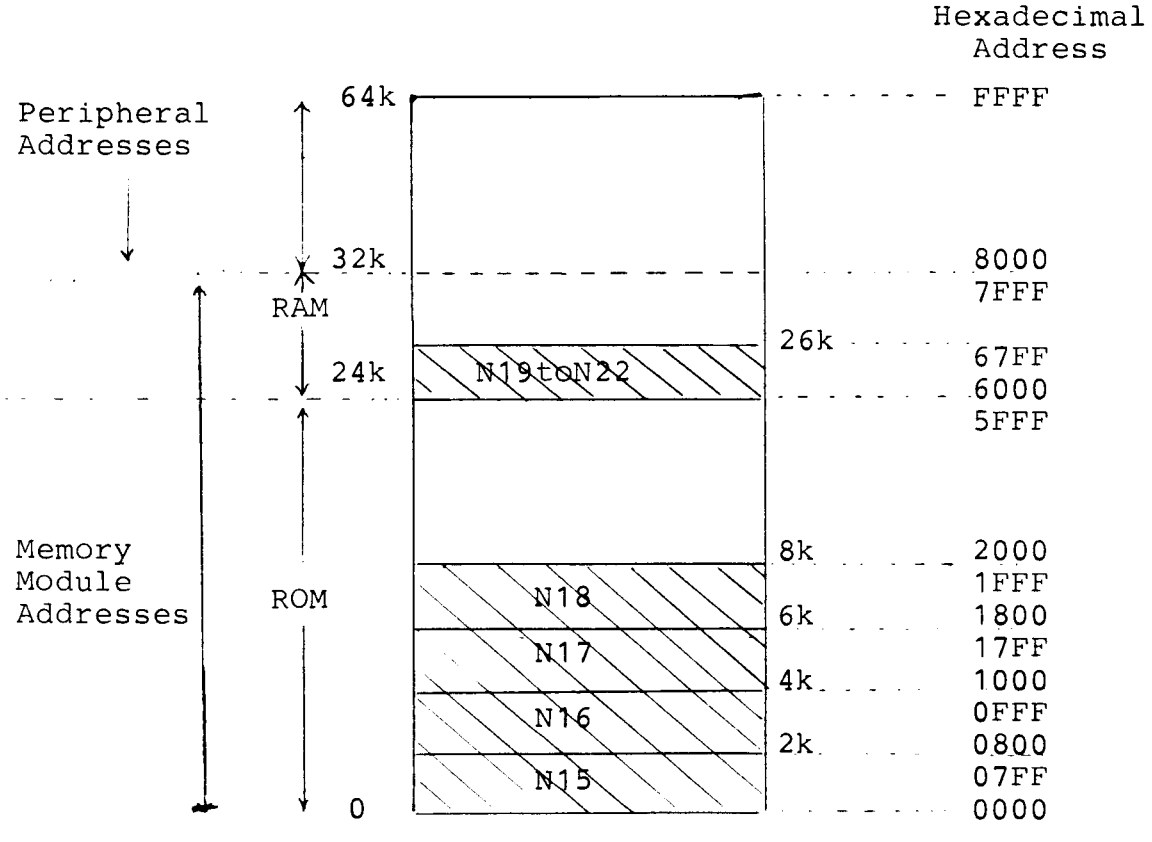
Within the processing block of the control function there is a need for various timing processes. This need arises from the timers described in connection with the application programmes. These timing processes can occur concurrently, i.e. simultaneously.

Fig.6.6 MEMORY MAPPINGS

ADDRESS LINE												PERIPHERAL ADDRESSED	
A_{15}	A_{14}	A_{13}	A_{12}	.....	A_5	A_4	A_3	A_2	A_1	A_0	Function	Module
1	1	1	1		x	x	x	x	x	x	Subscriber Line Message Handlers	rec.
1	1	1	0		0	0	0	0	0	0		ext.0 TRANS. STATUS REGISTERS
		.										
		.										
1	1	1	0		0	1	1	1	1	1		ext.31
1	1	1	0		1	0	0	0	0	0		ext.0 TRANS. INPUT REGISTERS
		:										
		:										
1	1	1	0		1	1	1	1	1	1		ext.31
1	1	0	1		x	x	x	x	x	x	IP Message Handler	rec.
1	1	0	0		0	x	x	x	x	x		trans. status reg.
1	1	0	0		1	x	x	x	x	x		trans. input reg.
1	0	1	1		x	x	x	x	x	x	Real Time I/O Buffer	rec.
1	0	1	0		0	x	x	x	x	x		trans. status reg.
1	0	1	0		1	x	x	x	x	x		trans. input reg.
1	0	0	1		0	x	x	x	x	x	IM I/O Buffer	status reg.
1	0	0	1		1	x	x	x	x	x		input reg.
1	0	0	0		x	x	x	x	x	x	Interrupt Reg.	———

a) PERIPHERAL ADDRESS MAP

reg=register
rec=receiver
trans=transmitter
ext=extension
x=don't care
IM=Interface Management
IP=Inter Processor



b) MEMORY MAP

▨ on card

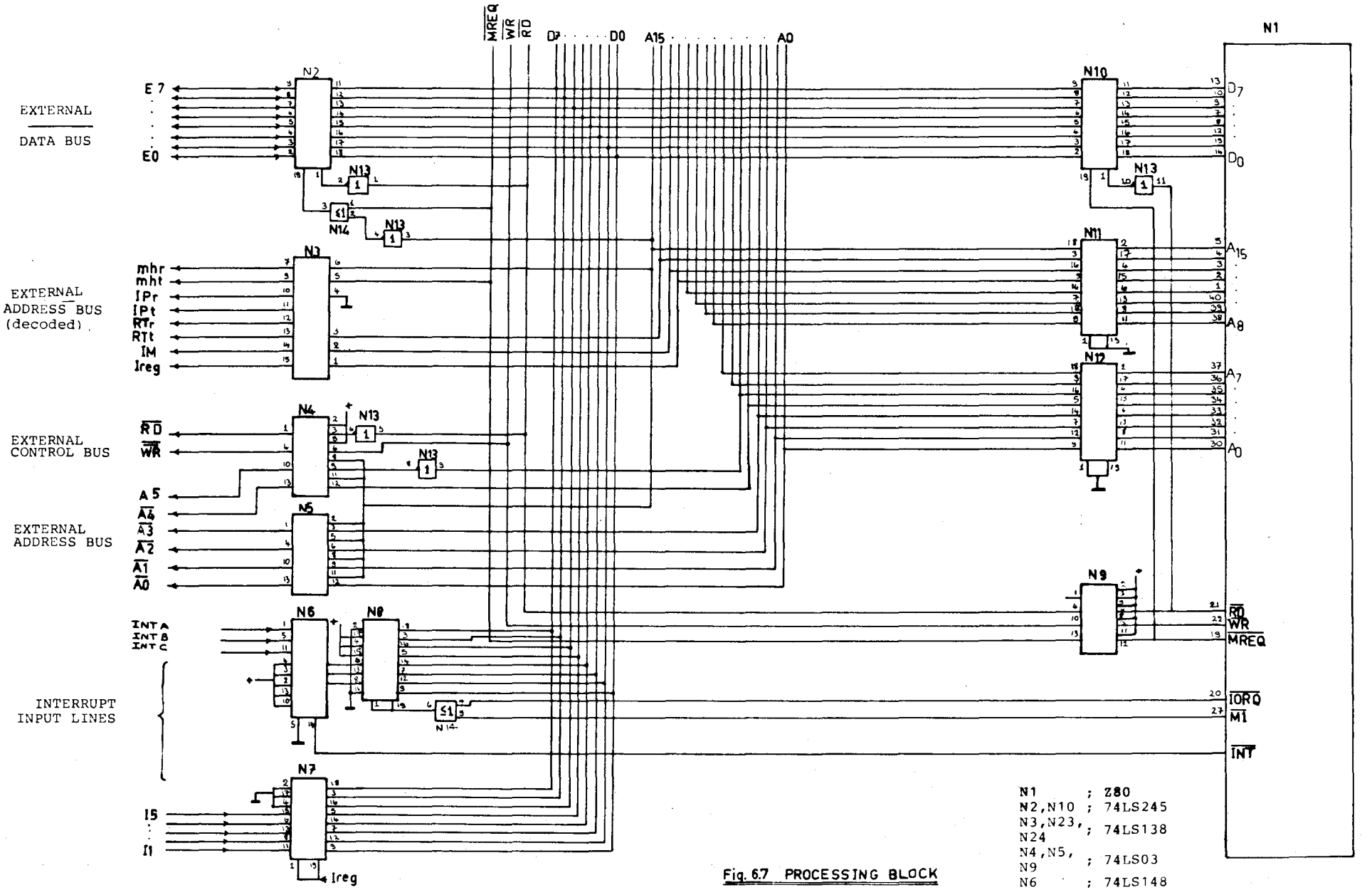
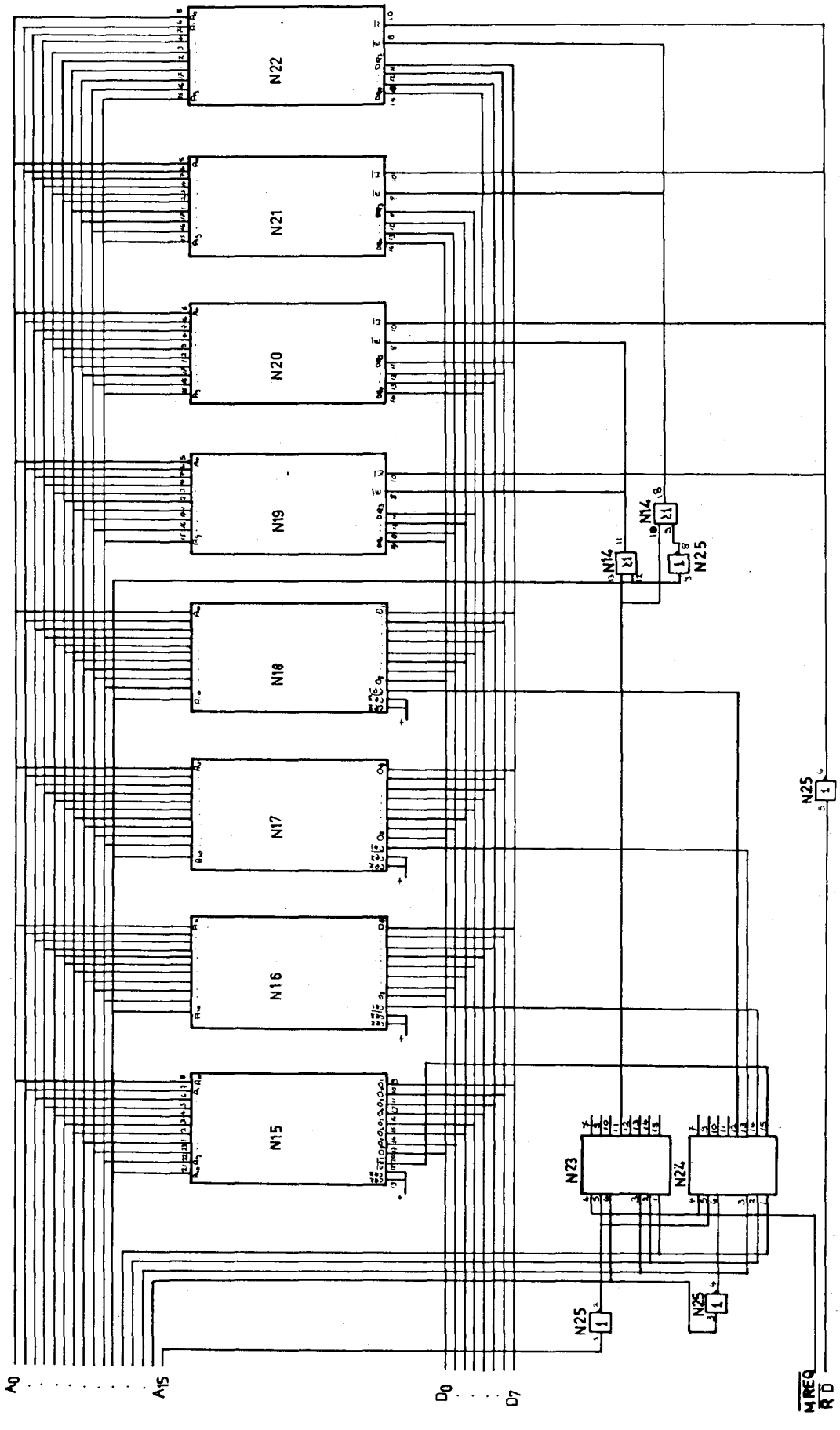


Fig. 6.7 PROCESSING BLOCK

a) PROCESSOR & INTERFACE

N1	;	Z80
N2,N10	;	74LS245
N3,N23, N24	;	74LS138
N4,N5, N9	;	74LS03
N6	;	74LS148
N7,N8		
N11,N12	;	74LS244
N13,N25	;	74LS04
N14	;	74LS32
N15,N16, N17,N18	;	82S191
N19,N20, N21,N22	;	HM6514-9



b) MEMORY UNIT

In order to reduce the workload on the Z80 processor these timing processes will be implemented in a slave unit which, although operationally an element of the processing block, will physically occur outwith this block. The necessary information transfer between the processor and this Real Time Function, as the collection of these timing processes will be called, will occur via the standard message transfer techniques used. As far as this transfer is concerned it is thus seen as an independent peripheral unit.
The function itself, as shown in fig. 6.8, consists of:

- an I/O buffer,
- a 8049 microcomputer with on board memory,
- a counter.

The hardware and software of this function will be discussed in appendix C.

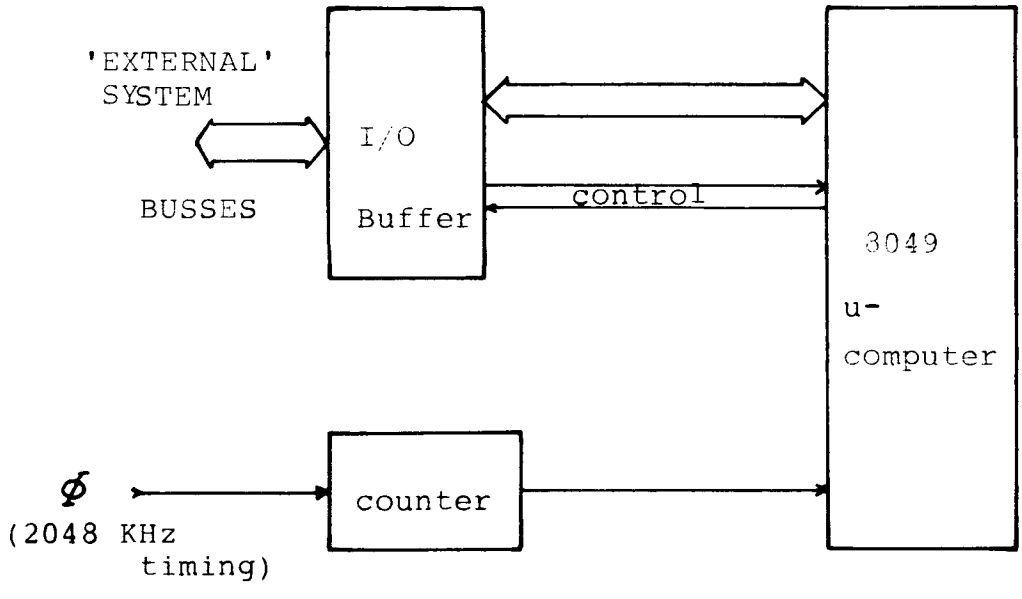


Fig. 6.8 REAL TIME FUNCTION

B) Peripheral Interface Block

This must supply the physical connections between processor and peripheral units. As the I/O buffers are situated in the peripheral units themselves, this connection can be seen as simply the extension of the processor data, address and control busses to these buffers. As shown globally in fig. 6.5, in order to limit the decoding circuitry and etc. required in the buffers, a certain amount of address decoding takes place in this interface block. Also the data and control busses are buffered, and the necessary interrupt handling circuitry in included. The hardware realisation of this is shown, along with that of the processing block, in fig. 6.7, while the various elements of the interface block are described in the following:

1) Data Bus Buffer

Messages transferred between a memory location corresponding to a peripheral unit and the processor, require that the processor data bus be connected to that unit. Due to the number of circuits connected to this bus, a distinction is made between the above transfers, which occur via an 'external' data bus (external to the processing block), and those concerned with the memory unit - which occur via the normal data bus. This 'external' data bus is connected to the 'normal' bus only when a transfer occurs relating to a peripheral unit (i.e. $A_{15} = 1$). The connection of the two busses occurs via a transceiver, whose mode of operation is dependent on whether the peripheral is being written to or read from.

2) Address Bus Decoding

The processor operates using a 16 bit address bus. Instead of connecting this to each peripheral unit a certain amount of address decoding is implemented in the interface block.

A peripheral unit being addressed now leads to the activation of an 'enable' signal to that unit.
Transmit, receive and status buffers in the unit are activated by separate 'enable' signals from this decoder.

In the case of the transmit buffers of the subscriber line message handlers, only partial decoding takes place. This is due to the number of 'enable' lines which would otherwise be required. Here a 'function enable' signal is generated by the decoder, while the low order bits of the address bus – identifying the specific transmit buffer concerned – are also given as output. Decoding of these low order address bits, i.e. enabling the specified unit, occurs on the printed circuit card containing these handlers. As this secondary decoding can be done using a central decoder on the card, the saving in connecting lines more then justifies the small amount of extra hardware (decoders) required.
The external address bus to the peripheral units therefore consists of a collection of 'enable' signals and 'address' lines.

3) Control Bus Interface

This consists merely of the transfer of the Read and Write control signals from the processor to the peripheral units, via the external control bus, and the driving of these lines.

4) Interrupt Handling Circuitry

The peripheral units indicate, to the processor, that a message is ready to be collected by the generation of an interrupt signal. The processor can thus receive an interrupt request from:

A) the inter processor message handler,
B) any one of the 'subscriber line' message handlers
or C) the 'Real Time' function.

On occurrence of one or more of the above, the interrupt handler must generate the actual interrupt signal to the processor.
On acknowledgement of this signal it must then supply the identity of the interrupt source.
So that simultaneous interrupts can be handled one at a time, a priority is given to each of the above sources; A having the highest priority, C the lowest. The source identity is supplied in the form of an interrupt vector, obtained in the interrupt handler via a priority encoder, i.e. the identity of the highest priority 'active' source is given.
In the case of an interrupt occurring from B above, the processor will also be supplied, on demand, with the identity of the specific message handler which is active. This information, supplied from the cards containing the message handlers via the 'identification' lines (I_1 to I_5 in fig. 6.7), is transferred to the processor from the interrupt handler via the 'Interrupt Register'.

6.3. Control Function Software

6.3.1. Introduction

This supplies the overall control of the FEU, via the system hardware. From messages received from the various peripheral units, the software of the control function derives a sequence of actions to be carried out by the system. These actions are the physical support used to maintain a connection between users.

Two main levels of software can be distinguished, as seen in fig. 6.9:

- Input and Output routines, which order the transfer of messages to and from the control function. These also supply the software/hardware interface within the control function.
- Message Processing, which translates the received messages into a specific sequence of actions and initiates these (via the I/O routines). This area is, to a great extent, hardware independent.

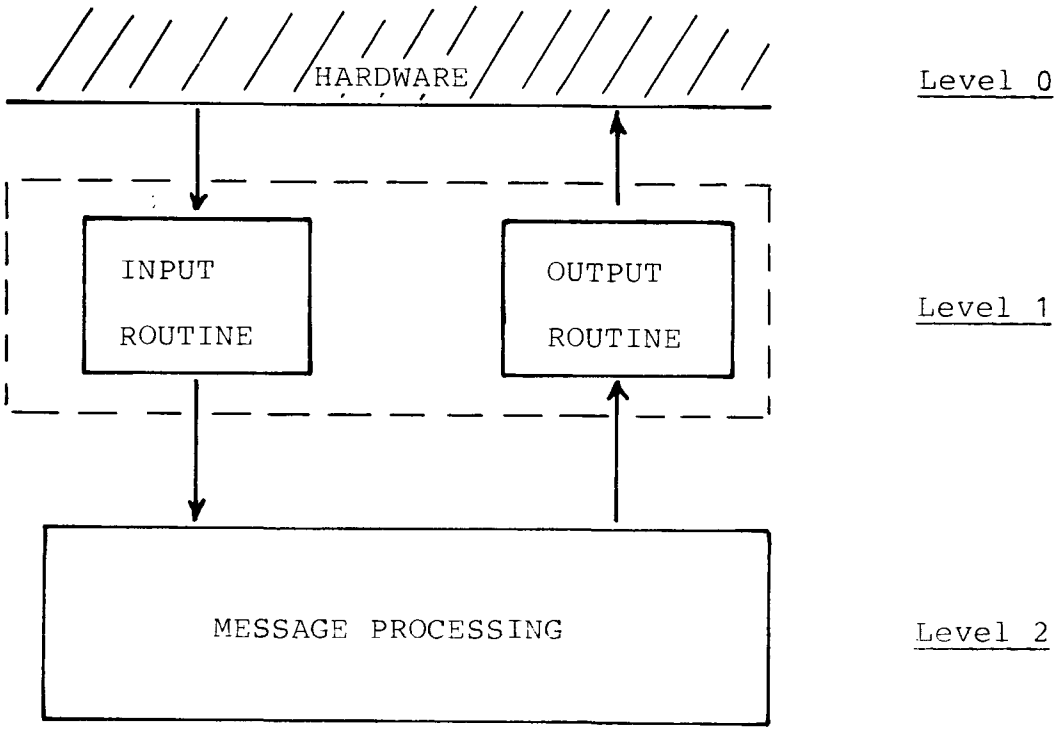


Fig. 6.9 **BASIC SOFTWARE STRUCTURE**

The general structure will be discussed in this section while, in section 6.4, the detailed realisation of the various elements will be considered.

6.3.2. Input Routines

As has already been discussed in chapter 5, message input to the control function is to occur on an interrupt basis. This frees the source handler quickly.

The input routines must then, cause the processor to be interrupted in an orderly manner, so that the interrupted processing can be resumed at a later time. The interrupt source must then be identified and the message collected. Lastly, the message must be presented to the processing function and the processing restarted.
Two main methods of message presentation to the processing function are possible:

- directly, in which case the new message is processed immediately. Any message processing being carried out at the time of interrupt is then delayed until the new arrival has been serviced. This, in fact, creates a Last In First Out queue (LIFO).
- via a buffer. The arrival is entered at the tail of a queue. The processing function is then re-instated at the point reached before interrupt, and continues from there. On completion, the next message in the queue is processed. This is a First In First Out queue(FIFO).

The latter is chosen, as this gives a more equitable manner of handling the various sources. (A priority has already been given to the incoming transfer requests, as described in 6.2). The input routines then, enter message arrivals into a queue to await processing.

Consideration must be given to the required capacity of this input queue. This is dependent on the message flow in the system, which will now be considered briefly.

In the busy hour approximately one quarter of the user population (0.25 Erlangs traffic/user) will be active simultaneously, i.e. 16 users neglecting the relatively small load generated by telemetry and the common equipment in the subscriber station.

The message generation will have its peak value during the build up phase of a connection, in which the dial ciphers are generated. Not all active users are in this phase - it will be assumed that one half are in this peak activity phase simultaneously. Typically a maximum rate of message generation, from a manually driven source, is ± 15 messages/sec. This leads to a total of (8 x 15) + (8 x 2) = 136 messages/sec. (in which active users outwith this 'peak' activity phase are assumed to generate ± 2 messages/sec). Allowing a small margin for the other user equipments, leads to a generation intensity of ± 140 messages/sec. from the subscriber stations.

Considering the system operation, it is seen that a message from a user normally leads to a derived message being transferred to the CPU. Due to the handshake technique used, a reply from the CPU, and a corresponding message to the user, are also part of the normal procedure.
Assuming that the CPU/FEU message transfer also includes a certain amount of purely system orientated information, leads to a typical average message flow for the system as shown in fig. 6.10, i.e. an input of ± 340 messages/sec. to the FEU can be expected.

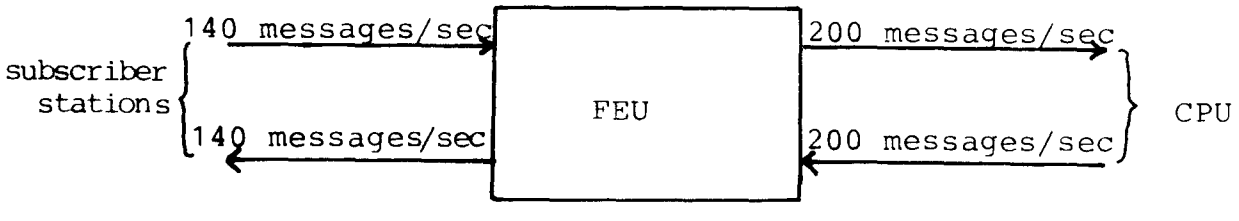


Fig. 6.10 MESSAGE TRANSFER INTENSITY

Each message must be processed inthe control function. This leads to a certain utilisation factor, ρ, for the processor which is a measure of the fraction of time during which the processor is busy. This is tabulated, in table 6.1, for various values of processing time required per message, τ, and the number of messages per second which must be processed, I.

ρ (msec.)	I (messages(sec.)				
	200	250	300	350	400
0,5	0,1	0,125	0,15	0,175	0,2
1,0	0,2	0,25	0,3	0,25	0,4
2,0	0,4	0,5	0,6	0,7	0,8
3,0	0,6	0,75	0,9	> 1	> 1
4,0	0,8	1,0	> 1	> 1	> 1

(ρ > 1 indicates an overloaded system)

$$\rho = I \times \tau$$

Table 6.1

A realistic value for τ would be $\pm$ 1 msec. (allowing $\pm$ 200 instructions, using a Z80 with 2 MHz clock, to be carried out). This leads to a system loading of $\pm$ 35% (ρ = 0,35).

Assuming further, that the system can be represented by an M/M/1 structure (see ref. 4) leads to a probability of finding k messages in the queue at any time of:

$$P(\underline{n}_q = k) = \rho^k(1-\rho).$$

This is tabulated in Table 6.2.

k	$P(\underline{n}_q = k)$
0	0,650
1	0,228
2	0,079
3	0,028
4	0,010
5	0,003
6	0,001
7	-
8	-

Table 6.2

The expected queue length, $E(\underline{n}_q) = \frac{\rho}{1-\rho} = 0{,}54$ messages, while the probability of having a queue of more than 6 messages is negligible.

6.3.3. Output Routines

Typically the message processing phase will lead to the generation of a message which must be transferred to a peripheral unit. The output routine implements this transfer.
The identity of the specific peripheral is included in the message, so that direct addressing can be used.
A problem which arises is that the relatively slow speed of operation of these peripherals means that the output routine may encounter a 'busy' peripheral, i.e. one busy with a previous transfer. In this case, as the system cannot, in general, be allowed to wait until this unit becomes free, the message must be put into a queue and a further transfer attempt made at a later time.
The length of this output queue is dependent on the number of message output transfers made and the probability of encountering a 'busy' peripheral.
A very rough approximation can be obtained by considering each peripheral as a processing element and assuming an M/M/1 structure (as for the input routines) for each of these systems separately. A more reliable figure could be obtained by simulation if necessary.
Three main sub-systems are to be considered:

a) Inter-processor message handlers.
b) Subscriber signalling circuit message handler.
c) Real Time function.

a) IP

actual message rate = $\pm$ 200/sec.
maximum message rate = $\pm$ 350/sec. (from calculations in chapter 5)

$\rho = \frac{200}{350} = 0{,}571$; $E(\underline{n}_q) = \frac{\rho}{1-\rho} = 1{,}33$ messages

$P(\underline{n}_q = k)$ as shown in table 6.3.a).

b) Single Subscriber

actual message rate = ± 17/sec. (15 + 2)
maximum message rate = ± 50/sec.

$\rho = \frac{17}{50} = 0{,}340$; $E(\underline{n}_q) = 0{,}5$ messages

$P(\underline{n}_q = k)$ as shown in Table 6.3.b).

For the 8 stations in this 'peak activity' phase $E^T_{(\underline{n}_q)} = 4$ messages, while the effect of those outwith this phase is negligible.

c) Real Time Function

This deals mainly with message outputs from the control function used to start and stop timing processes. It will be assumed, considering the speed of operation of the processor included (8049), that on average ½ msec. is necessary to free the message handler included, i.e. maximum message rate = 2000/sec. Assuming further that, in the steady state, a 'timing' message is created for, on average, 1 in 2 of the messages processed in the control function, i.e. actual message rate $= \frac{340 \times 2}{2} = 340$ messages/sec.

$\rho = \frac{340}{2000} = 0{,}170$; $E(\underline{n}_q) = 0{,}2$ messages

$P(\underline{n}_q = k)$ is shown in Table 6.3.c).

k	$P(\underline{n}_q = k)$		
0	0,429	0,660	0,830
1	0,245	0,224	0,141
2	0,140	0,076	0,024
3	0,080	0,026	0,004
4	0,046	0,009	0,001
5	0,026	0,003	-
6	0,015	0,001	-
7	0,009	-	-
8	0,005	-	-
9	0,003	-	-
10	0,001	-	-
	a)	b)	c)

Table 6.3

From these calculations is it seen that an output queue length of (10 + 8 x 6 + 4) = 62 would seem sufficient for normal operation. It must be stated again that, due to the approximations and assumptions used, this gives only an order of magnitude for the queue length.

It is seen that, due to the intensive usage, any output routines must be of relatively short duration.

The system can now be seen as in fig. 6.11.

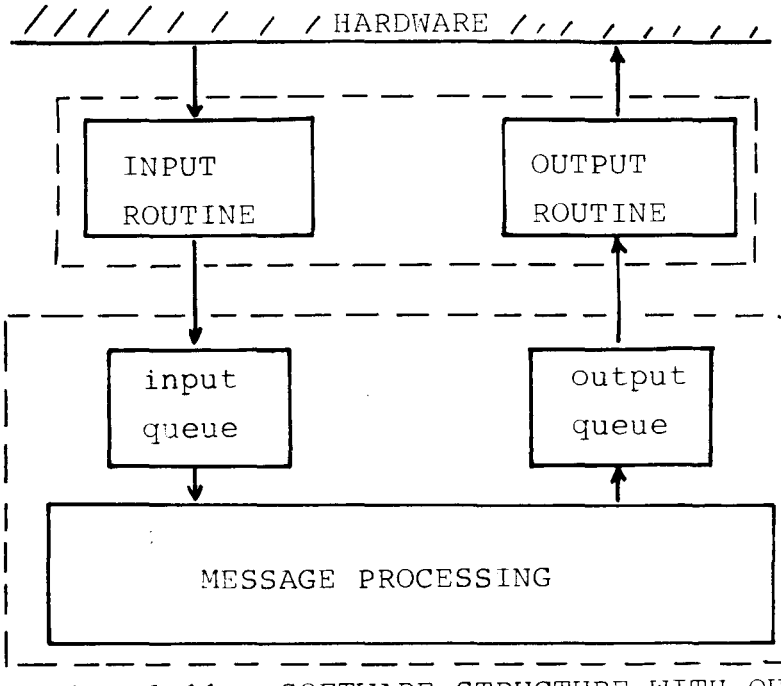


Fig. 6.11 SOFTWARE STRUCTURE WITH QUEUES

6.3.4. Message Processing

This area of the software consists of:

- input and output queue handling routines;
- Main Programme.

The Main Programme carries out the translation of an input message into the required series of actions which must be carried out by the system, and initiates these actions.
The specific actions for each message are described in a separate routine-an Application Programme. The Main Programme must then, from a message, choose the single Application Programme concerned, i.e. carry out a 'Case' statement, and cause this programme to be activated — the Application Programme then continues with the initiation of the various actions specified.

These application programmes can therefore be considered as a further layer in the software structure, as shown in fig. 6.12.

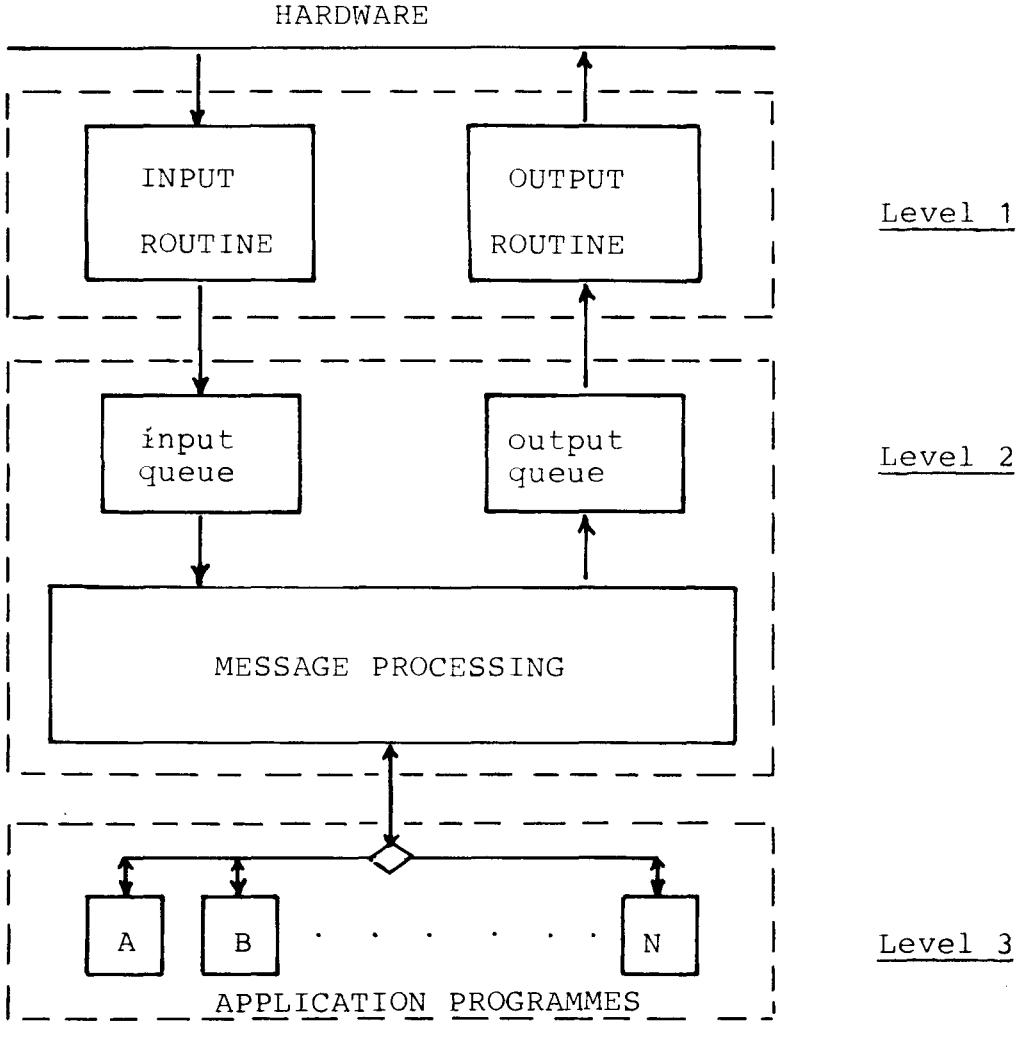


Fig. 6.12 DETAILED SOFTWARE STRUCTURE

Certain system routines, such as time slot management, are also present in this third layer. These will be described at a later stage of the discussion.

This model can be simplified to some extent by the combination of the in- and output queues into a single processor queue. As each message has a specific code dedicated to it, as described previously, this simplification of the necessary software can be made, but at the cost of system flexibility (i.e. the number of possible different messages is reduced). This will be done in the system being considered as the maximum number of different messages (256) is considered sufficient to serve the system requirements.
This then, leads to the system operation as shown in fig. 6.13, in which the arrows indicate message transfers.
The input routine transfers a message into the queue.
The message processing collects a message from the queue, translates this into the identification of an application programme and starts this programme. Typically, this results in one or more messages which must be transferred to a peripheral (i.e. initiate a given event). This transfer is carried out by the output routine, if possible, otherwise the message is entered into the queue and the transfer attempted again at a later time.

6.4. Software Implementation

In this section the implementation of the system operation, described in section 6.3, will be discussed and a realisation of the software will be proposed.

6.4.1. Input Routines

The sequence of actions leading to the collection of a new message by the processor is explained below and shown in fig. 6.14.

The occurrence of a new message (1 in fig. 6.14) in the peripheral unit, causes a local interrupt signal to be generated (2). This is transferred, via a priority encoder, to the maskable interrupt input of the processor (3).

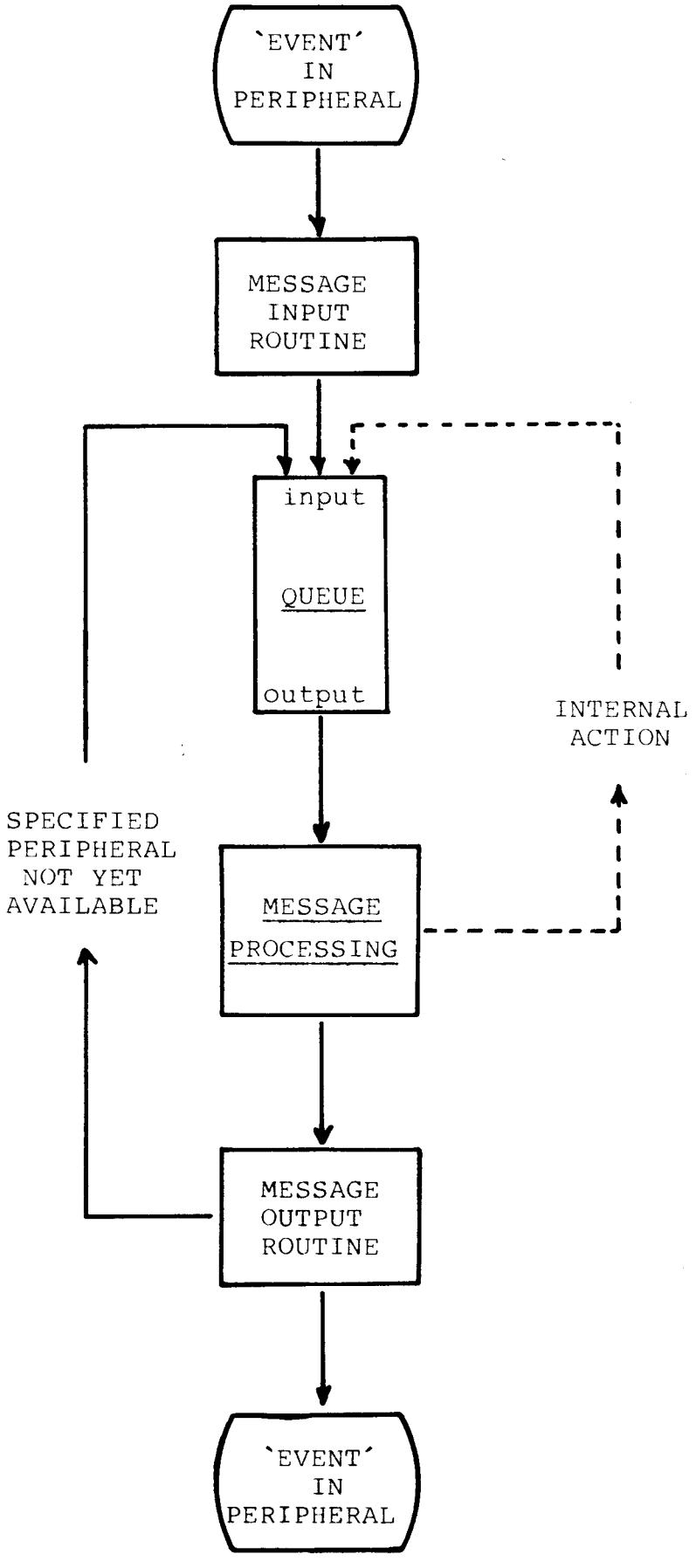


Fig.6.13 SYSTEM SOFTWARE OPERATION

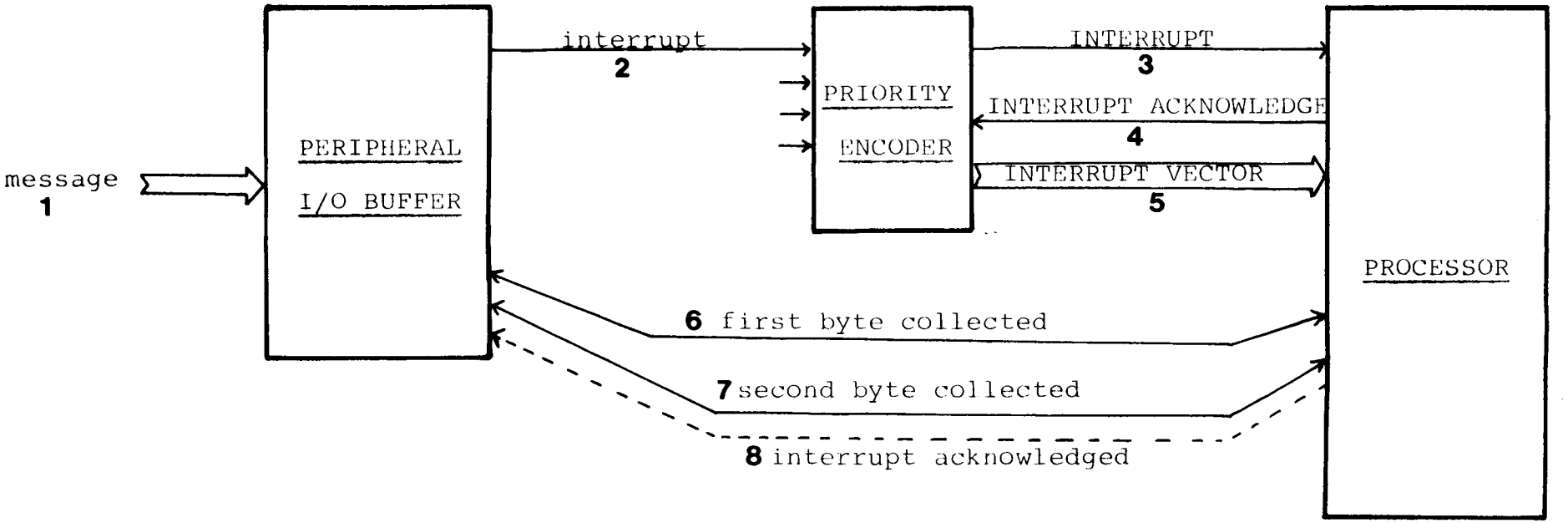


Fig.6.14 MESSAGE INPUT

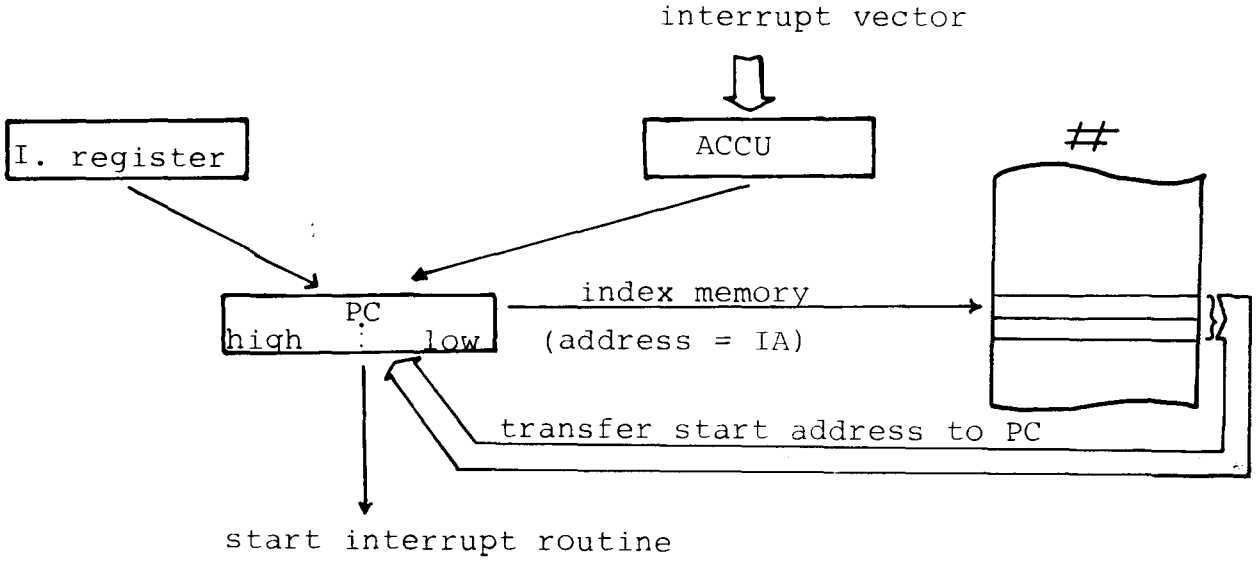
Each peripheral unit which can cause message transfers has a separate input line connected to this priority encoder. When the processor is ready to service this request, an interrupt acknowledge is generated (4). On receipt of this, the encoder transfers an interrupt vector to the processor (5). This vector identifies the source of the interrupt. From this information the processor causes the two byte message to be collected from the specified source (6 and 7). The collection of the second byte is also seen as an interrupt acknowledge to the peripheral unit (8), i.e. the peripheral is freed for further use. Protection against multiple (near) simultaneous interrupts occurring is supplied by the priority encoder - which processes the highest priority interrupt first, and by the processor itself - which will handle interrupts strictly in sequence of arrival. A queue of interrupt requests can therefore form at the encoder input.

The input routine can thus be split into three phases:

1) Recognition and acknowledgement of an interrupt, followed by the collection of the interrupt vector.
2) Translation of this vector into the source location and collection of the two byte message.
3) Entry of this message into the queue.

As is seen in reference 3 , using Mode 2 of the Z80 interrupt responses, the first phase is automatic. The maskable interrupt input is used due to the occurrence of critical sections in the software - namely operations on the queue variables. An interrupt occurring while this input is disabled causes a delay in servicing of the request. Otherwise the processor generates a $\overline{IORQ}/\overline{M1}$ pulse combination (i.e. interrupt acknowledge). This in turn causes the encoder to transfer the interrupt vector, via the data bus, to the accumulator.

After pushing the present Programme Counter (PC) contents onto the stack, the accumulator contents, in combination with that of the I register, are used as a pointer to a location where the start address of the corresponding service routine is found. This start address is entered into the PC and a jump to this location is made. This sequence of actions is shown below.
Phase 1) above is thus completed.



Control is thus transferred to a specific interrupt service routine.
Three separate interrupt sources are possible:

a) inter-processor message handler;
b) subscriber circuit message handlers;
c) Real Time function.

Each of these will lead to a separate service routine. The common purpose of these routines is the entry of the source message into the work queue. The routine for actually entering the message into the queue, QIN, is common to all the above service routines (this is in fact phase 3).

Phase 2 consists of supplying QIN with the necessary parameters. This can be done in one of two ways:

1) where the message is collected from the corresponding peripheral and entered into a fixed temporary storage area. QIN is then called and transfers the contents of this area into the queue itself,

or 2) by transferring the address of the location where the message can be collected to QIN. This again calls for a temporary storage area where the address is found.

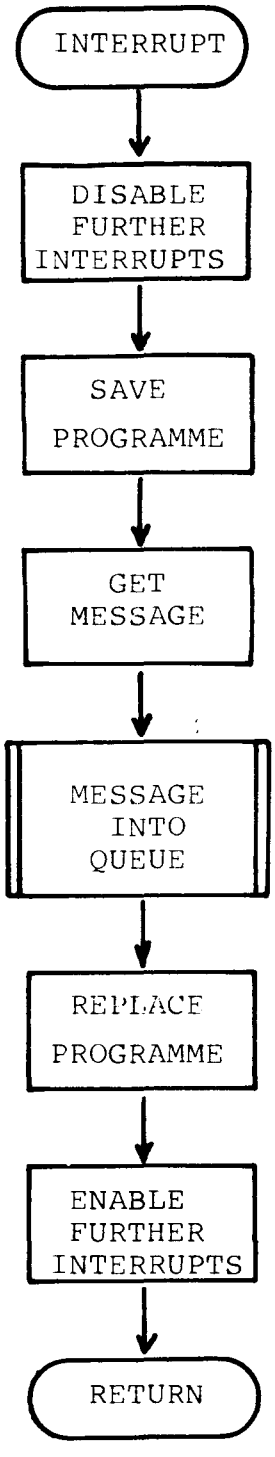
The former method is chosen as it frees the peripheral somewhat more quickly although, due to the double transfer, the total processing time will be higher than would be the case with the latter method.
As memory mapped I/O is used the messages can be collected directly (a given routine handles a specific peripheral, while the interrupt vector indicates the specific routine to be used). Registers BC of the processor are used to supply the temporary storage necessary.
At the end of the routine the system is returned to its original state.

A flow chart for the global routine is given in fig. 6.15. The realisation of this, for the various sources, is given by the routines INT1, 2, 3 and QIN in the programme listing at the end of the chapter.

For the subscriber line message handlers, the interrupt routine (INT2) must also identify the specific handler and combine this information with the message - for reasons considered previously.

6.4.2. Output Routines

As is seen in fig. 6.13 any output message resulting from the 'Application' processing is not entered into the queue directly.



Interrupt generated

Save present parameters which could be altered during this routine

Identify source, collect message and enter into temporary store

Call queue input routine which transfers the message into the proper position in the queue

Return the saved parameters

Re-instate the interrupted programme

Fig 6.15 INTERRUPT ROUTINE : FLOW CHART

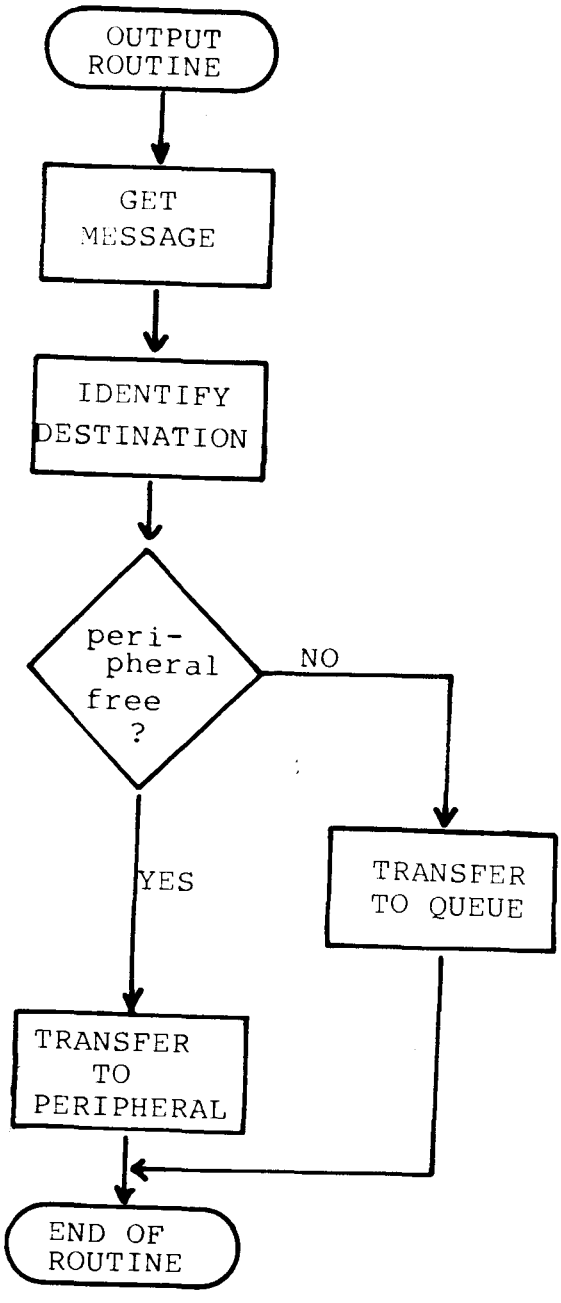
An output routine is called and an attempt made to transfer the message to the specified peripheral. Should this transfer be blocked by a busy peripheral, then the message is put into the queue for a further transfer attempt at a later time.
This deviation from the basic structure (which would require the message to be entered into the queue directly) is made in the interest of efficiency. The message flow is speeded up and the length of queue required is reduced.

An output routine (called from either an application or the main programme) receives, as parameter, the message to be transferred. Four main destinations are possible, namely:

1) inter-processor message handler;
2) real time function;
3) interface management controller;
4) a subscriber line message handler.

For each of these a slightly different routine is required, but all have the same goal, as described above. The specific routine required is identified by the calling process, as this is indicated implicitly in the output message.
A flow chart for the global output routine, along with a brief description of its operation, is given in fig. 6.16. The realisations of the four routines (MOUT1 to MOUT4, corresponding to the above destinations) is given in the programme listing at the end of the chapter.

As seen from this listing MOUT1 to 3 can address the peripheral directly - this having a fixed address (memory mapped I/O). For MOUT4, which serves all 32 of the subscriber line message handlers, an extra operation must be carried out, namely the identification of the specific handler, before the transfer can be attempted.



CALL specific output routine

Transfer message tothis routine

Test status register of the specific peripheral

If peripheral is busy then return the message to the queue for a later transfer.
(by calling QIN routine)

If peripheral is free then transfer the two byte message to it

Return to normal processing

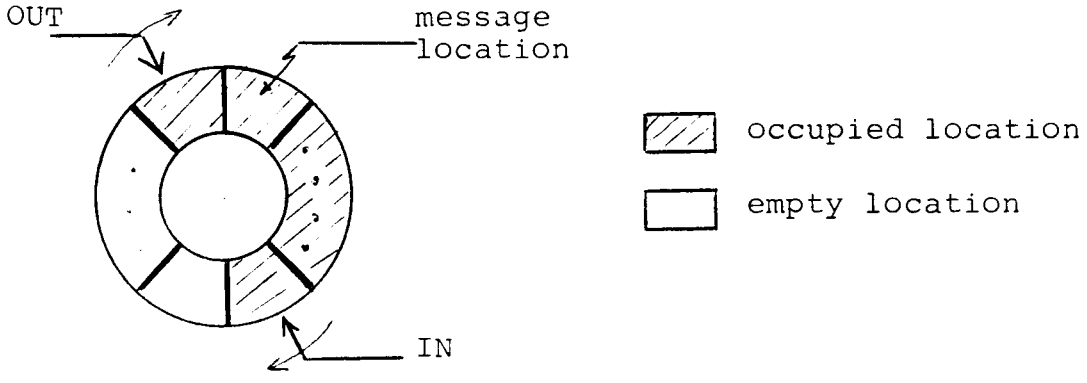
Fig. 6.16 GLOBAL OUTPUT ROUTINE

6.3.3. Message Processing

a) Queue and Queue Handling Routines

Queue

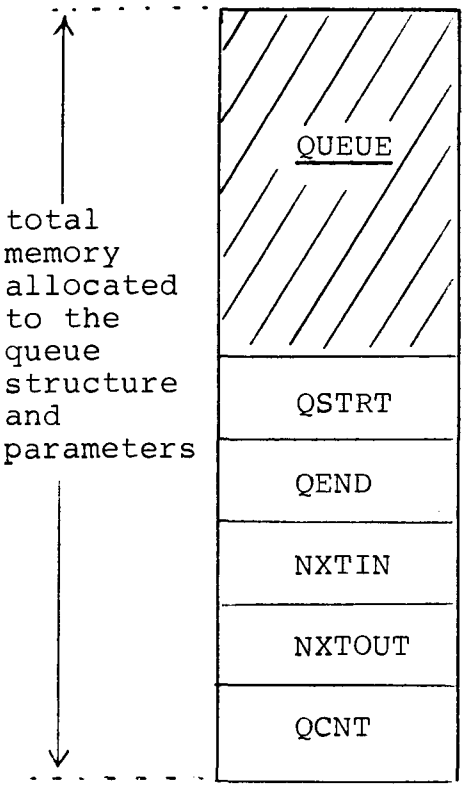
This data structure is used for the storage of messages awaiting processing.
A cyclic (round robin) structure is used to implement this, as indicated below.



QUEUE

A new message is entered at that location indicated by the IN pointer. This is then rotated in a clockwise direction to point to the next input location. Similarly, the OUT pointer indicates the next message to be collected from the queue. This also rotates in a clockwise direction. In order to avoid overwriting, protection must be supplied to avoid the condition in which the two pointers indicate the same location.

This structure is to be realised using the linear memory space available (RAM). In order to obtain the cyclic operation, a number of queue parameters must be available. The queue structure thus created is shown in fig. 6.17.
From the calculations of previous sections it is seen that a queue capacity of $\pm$ 70 locations would be sufficient.



QUEUE : the collection of memory locations allocated for the storage of messages.

QSTRT : a pointer to the first memory location allocated to the queue structure.

QEND : a pointer to the last memory location allocated to the queue structure

NXTIN : a pointer to the location in the queue into which the first byte of the next input message is to be written.

NXTOUT: a pointer to the location in the queue from which the first byte of the next output message is to be used.

QCNT : total number of messages in the queue at the present time.

Fig. 6.17 QUEUE STRUCTURE

A pointer is simply a 16 bit address and therefore encloses two bytes in the data structure, as do the messages.

Operations on these parameters are seen as critical programme sections, no two processes may access the parameters simultaneously (e.g. nested interrupts), as this could lead to malfunction.

The two basic operations on this queue are then the input respectively output of messages.

Queue_Input_Routine

This causes input messages to be deposited in the two sequential locations starting at NXTIN, if the queue was not already full. These input messages are placed, by the calling process, in a temporary storage area before the input routine is called. The queue parameters must then be updated.
A flowchart of this routine, QINP, is given in fig. 6.18, while the realisation is given in the programme listing at the end of the chapter.

Queue_Output_Routine

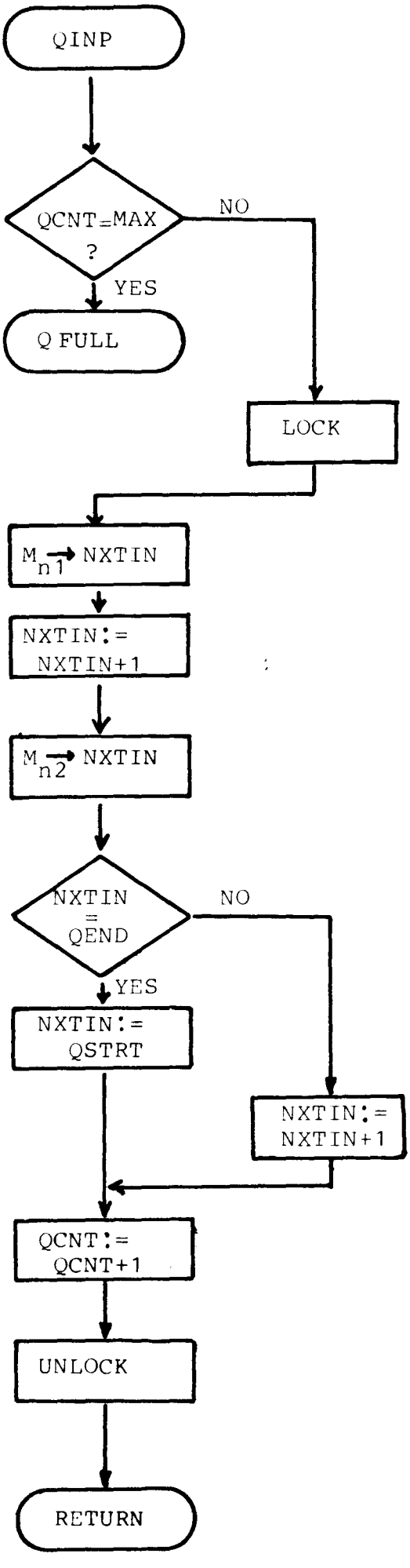
This operates in a similar manner to the above input routine but now results in the entry of a message into the temporary storage area for transfer to the calling process.
A flowchart for this routine, QOUTP, is given in fig. 6.19, while the realisation is given in the programme listing.

For these routines the temporary storage area consists of the BC register pair of the processor. This is chosen, as opposed to an external RAM area, to speed up the operation - register transfers operate more quickly than external memory accesses.

As is seen in fig. 6.18, due to the finite queue length, a condition can occur in which a message arrival meets a full queue, i.e. no further input is possible. Two solutions to this condition could be used:

1) by allowing the queue to overflow, i.e. discarding the new arrival;
2) by blocking any further inputs to the system until space has been created in the queue.

This latter would lead to a 'graceful' degradation in the service offered by the system.



Message input routine

Test if queue is already full

If so go to QFULL routine

If not block any further external operations on the queue and continue

Transfer first byte of message (in register B) to location indicated by NXTIN pointer

Increment NXTIN

Transfer second byte of message (in register C) into queue

Check if end of queue area has been reached

If so set NXTIN pointer back to the start of the queue area

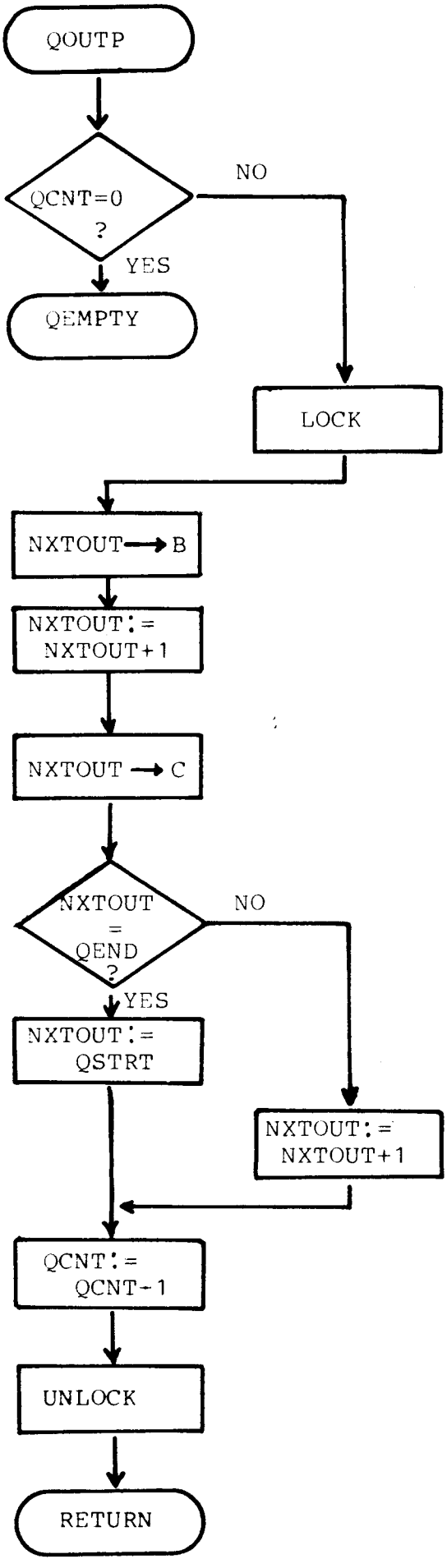
If not set NXTIN to next input location

Increment queue count

Unblock the system

Return to the calling process.

Fig. 6.18 QUEUE INPUT ROUTINE



Message output routine

Check if queue is empty

If so go to QEMPTY routine

If not block any further external operations on the queue and continue

Transfer first byte of message- in location NXTOUT- to register B

increment NXTOUT pointer

Transfer second byte to register C

Check if end of queue area has been reached

If so set NXTOUT to start of queue area

If not increment NXTOUT

Decrement queue count

Unblock the system

Return to the calling process

Fig. 6.19 QUEUE OUTPUT ROUTINE

But, as message processing leads normally to the generation of one or more output messages, the problem is not completely solved.
A solution based on the former is used. In this case the maximum queue capacity is increased to 255 messages. As can be seen from the calculations regarding the probable queue length, the probability of a full queue occurring, and therefore a message being lost, is now negligible.

These queue handling routines can be called by any process which requires access to the queue. QINP serves Message Input and Output routines and can be called by the main programme, while QOUTP is accessed exclusively by the main programme.

b) Main Programme

This consists of the continuous cycle:

- get the next message from the queue,
- process this message,
- output the result.

Processing the message consists of the identification of the application programme specified by a particular message, followed by the activation of this programme. Parameters transferred to the latter consist, very simply, of the originating message itself.
The flowchart of fig. 6.20 indicates the basic operation of this main programme (PROC), while the realisation is given in the programme listing.

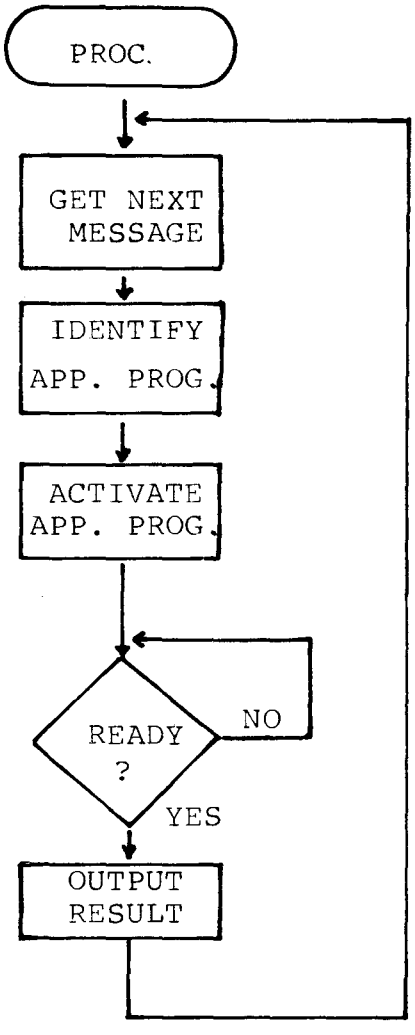
Message Translation

Many methods are available for the translation of a given message into the identification of a specific application programme (i.e. the implementation of a 'case' statement). The simplest of these, and that which will be used in this system, is based on the user of a Lookup Table.

The entries in this table are the start addresses of the various application programmes. The actual contents of a message (byte 2) is used to index this table - this being the displacement wrt the base of the table, as shown in fig. 6.21. The entry at this address is then the start address of the specific application programme required.
Each entry in the table consists of two bytes of memory, giving a total area of 512 bytes for the 256 message combinations. More complex translation structures could be used to reduce this area (see ref.5) but at the cost of extra processing time.

Message_Coding

Consideration must be given to the coding of the messages. Each message has a specific codeword (8 bits) dedicated to it. Identical messages being transferred over different trajects of the systems, e.g. subscriber to FEU and FEU to CPU, have different codewords. The list of all messages used is given in table 6.4. These can be found in the descriptions of the various application programmes of the next section. The specific codewords allocated to these messages are shown in table 6.4. The unused codewords are available for future use wrt expansion of the message vocabulary.



Main programme

Take next message from the queue

Translate the message into the identity of an application programme

Activate this programme

Wait until the programme has been completed

Transfer any result to the specified peripheral

-and go back to the start

Fig.6.20 MAIN PROGRAMME

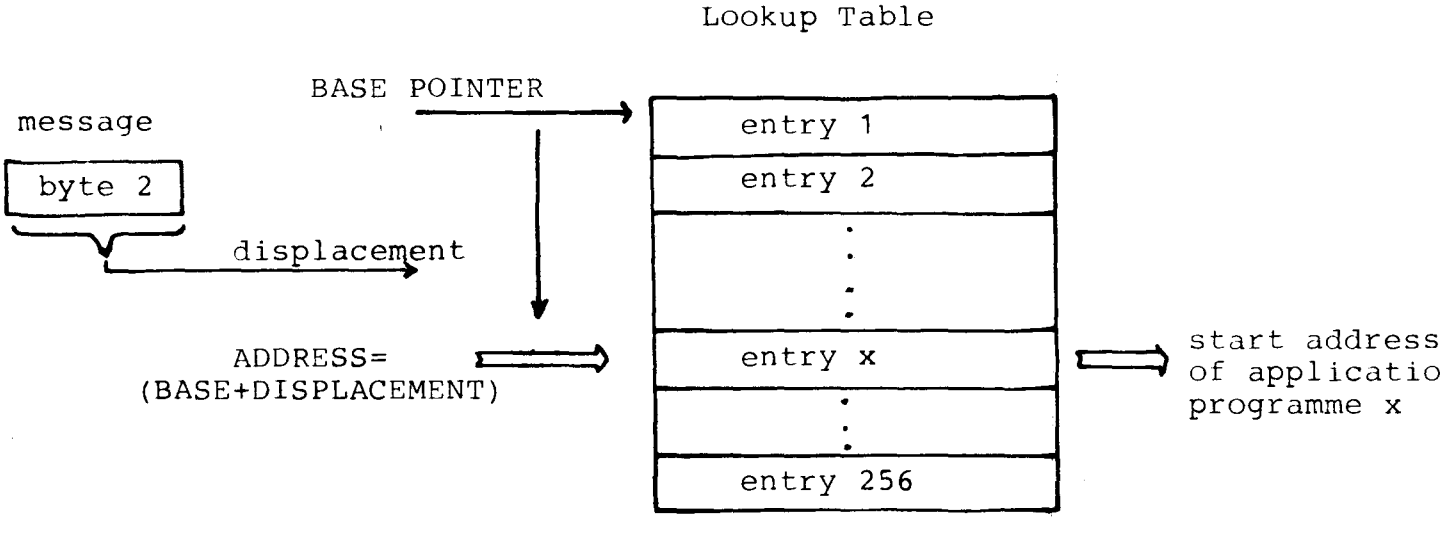


Fig.6.21 MESSAGE TRANSLATION

A \ B	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	CON REQ	CLL REQ	SEL CAR1	CLR CEN	conn	alloctd tm slt0	" 16	cllreq tm slt0	" 16		sel car1	clr req				
1	CON RDY	INFO 1	SEL CAR2		contd	" 1	" 17	" 1	" 17	info 1	sel car2	clr cen				
2	CEN REQ	INFO 2	" 3		strt tmr 1	" 2	" 18	" 2	" 18	info 2	" 3					
3	END CON	INFO 3	" 4		strt tmr 2	" 3	" 19	" 3	" 19	info 3	" 4					
4	MSG ACC	INFO 4	" 5		strt tmr 3	" 4	" 20	" 4	" 20	info 4	" 5	msg acc				
5	MSG REJ	PROC TOSEL	" 6		strt tmr 4	" 5	" 21	" 5	" 21	proc tosel	" 6	msg rej				
6	MSG' ACC	RDY	" 7		stp tmr	" 6	" 22	" 6	" 22	rdy'	" 7	msg' acc				
7	MSG' REJ	CLL DIS	" 8		dcon	" 7	" 23	" 7	" 23	cll dis	" 8	msg' rej				
8	MSG 1	CLL CON	" 9		dcond	" 8	" 24	" 8	" 24	cll con	" 9	msg' 1				
9	MSG 2	INC CLL	" 0		TMOUT	" 9	" 25	" 9	" 25	inc cll	" 0	msg' 2				
A	MSG 3	CLL REJ	" A			" 10	" 26	" 10	" 26	cll rej	" A	msg' 3				
B	MSG 4	CLL ACC	" B			" 11	" 27	" 11	" 27	cll acc	" B	msg' 4				
C	MSG' 1	CLR IND	" C			" 12	" 28	" 12	" 28	clr ind	" C	msg 1				
D	MSG' 2	CLR CON	" D			" 13	" 29	" 13	" 29	clr con	" D	msg 2				
E	MSG' 3	CONG	" E			" 14	" 30	" 14	" 30	cong	" E	msg 3				
F	MSG' 4	USRLKT	" F			" 15		" 15		lkout	" F	msg 4				

sub. station↔FEU

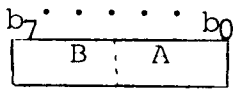
internal FEU

FEU ↔ CPU

msg=message
MSG=MESSAGE
tm slt=time slot

blanks not used

TABLE 6.4 MESSAGE CODING (HEXADECIMAL)



message format

6.4.4. Application Programmes

These describe, and initiate, the specific sequence of actions to be carried out by the system for a given message, and as such, are the detailed expansion of the events described in the Integrated Access Protocol of section 5.4.
The specific programme required from the set, is identified in the message processing function as described previously.
In order to support a connection between two users four global areas of message transfer exist:

- between subscriber station and FEU
- between FEU and CPU
- between CPU and the rest of the network
- within the network but outwith the exchange being considered.

The application programmes to be described deal with the first two of these areas.
The FEU, in this configuration, acts as a buffer for message transfers between subscriber station and CPU, and carries out a pre-processing function on certain messages from these sources. This processing leads to a number of actions being carried out by the FEU.
In the area being considered there are, thus, three regions in which messages are transferred:

A) between subscriber station and FEU
B) internal in the FEU itself
C) between FEU and CPU.

These regions will be indicated in the figures used to described the various message transfers involved, in a given application programme.

The general sequence of actions occurring during a connection consists of three phases, as described in fig. 5.26
A block diagram of this sequence is given in fig. 6.22

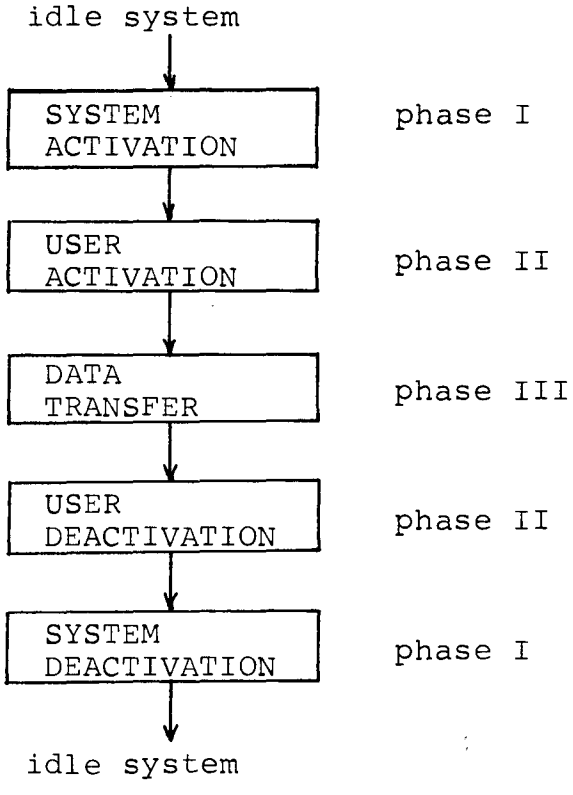


Fig. 6.22 GENERAL CONNECTION SEQUENCE

A user activation can only occur on an active system. This means that should any user attempt to create a connection, while the subscriber station is in the idle state, the build up of the end to end connection is preceded by a system activation phase.
Similarly, at the end of a connection, should the case arise in which no active user is present in a subscriber station, then the system is returned to the idle state via a system deactivation phase.

In the following sections these phases, and the basic application programmes available within them, will be considered separately.

As will become obvious, two software structures, common to a variety of programmes, are necessary. These system management structures are:

a) Time Slot Management - which controls the allocation of time slots on the IH.

b) Subscriber Station Descriptors - which are data structures containing all relevant information regarding the states of an extension and the user equipments on that extension.

a) Time Slot Management

This routine, called by an application programme, schedules the use of the 32 circuits, i.e. time slots, on the IH.
The allocation occurs dynamically, as all but time slot number 31, which is dedicated to inter-processor signalling, are available for use by any user for the connection period.
The input parameter to this routine is either a command to allocate a time slot to a specified user or to deallocate a specified time slot (i.e. at the end of a connection). Two possible parameters can be returned to the calling process, either a message indicating the time slot number to which a user has been scheduled, or a message indicating that, as no free time slot was available, the user has not been allocated a time slot (a congestion condition).
It is assumed that the time slot number specifies both 'go' and 'return' paths on the IH.

Implementation

A file structure as shown in fig. 6.23 is kept, in which each time slot has a corresponding entry. A status bit is used to indicate whether a particular time slot is already allocated, and a count is kept as to the total number of allocated time slots at any time.

The function therefore consists of two subroutines, one for the allocation of time slots (ALLOC), and one for the deallocation (DELOC). The particular routine required is called from the application programme.

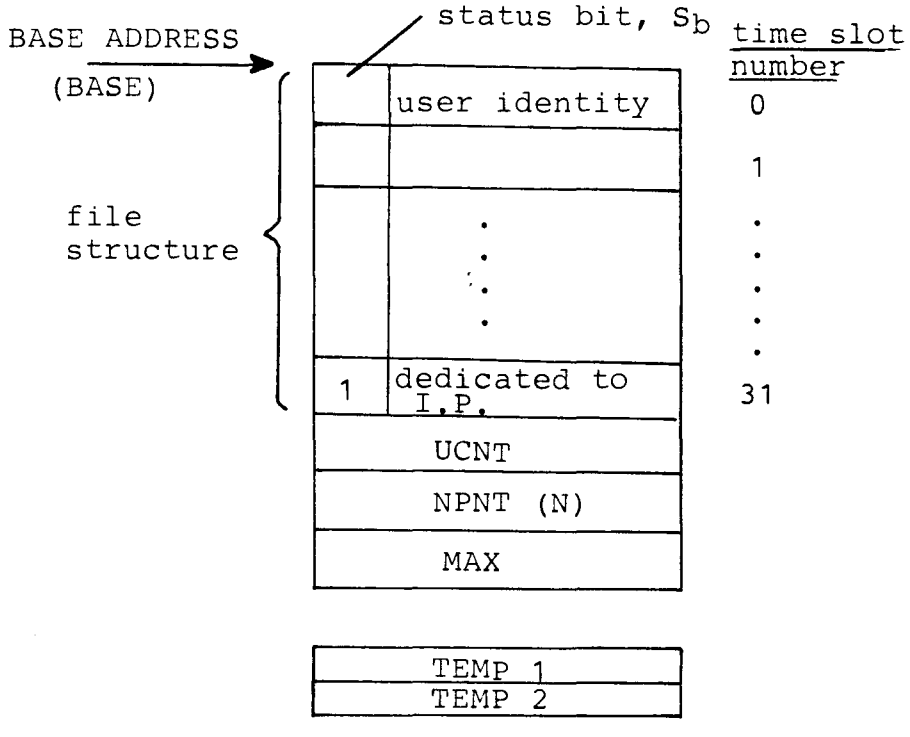


Fig. 6.23

The parameters connected with this file are:

UCNT : this is the present number of allocated time slots ($0 \leq$ UCNT ≤ 31; time slot 31 is not included)

NPNT : this is a pointer (i.e. address) to the present "time slot" entry being considered

MAX : last file location address

Two temporary registers, TEMP 1 and 2, are used for the message transfers to and from the calling process.

A description of the operation of these routines is given in combination with the flow charts of figs. 6.24 and 6.25. (at startup time all parameters must be set at their initial values), while a realisation is given in the assembler listing at the end of this chapter.

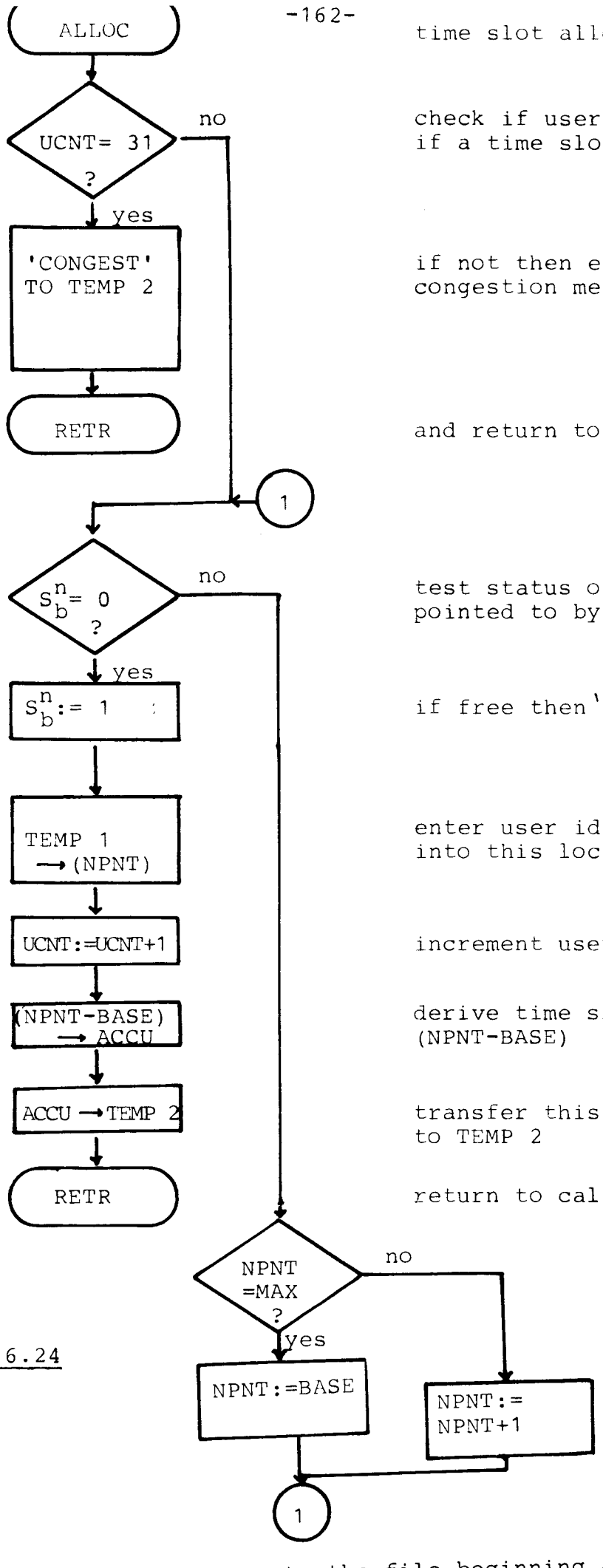
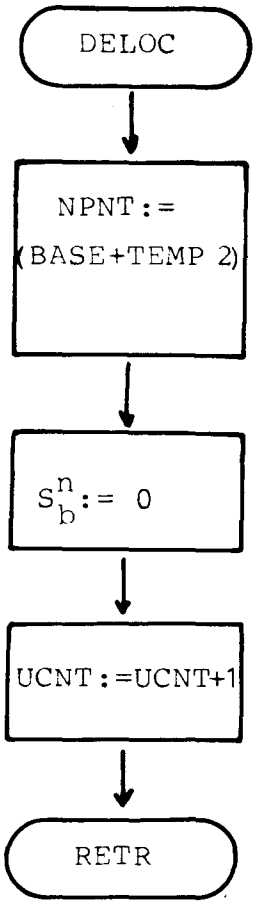


Fig. 6.24

By not setting the pointer to the file beginning at each search, the clustering of active entries is avoided.



set pointer to the required time slot i.e. location (BASE + time slot number contained in TEMP 2)

free the time slot

decrement the user count

and return to the calling process

Fig. 6.25

After a deallocation, N points to a 'free' time slot - which improves the search time when the next allocation occurs.

b) Subscriber Station Descriptor

This consists of a 7 byte data structure per subscriber station, as shown in fig. 6.26. The various elements of this structure are then considered.

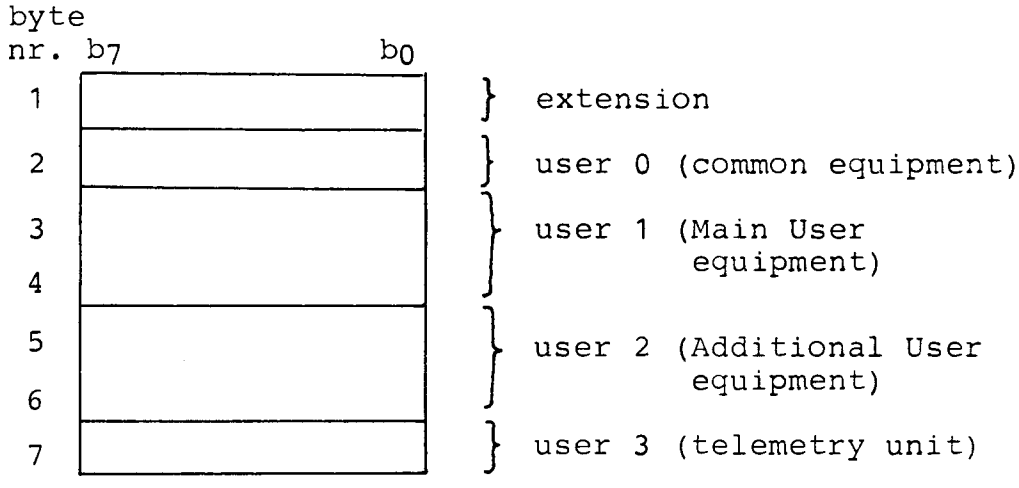
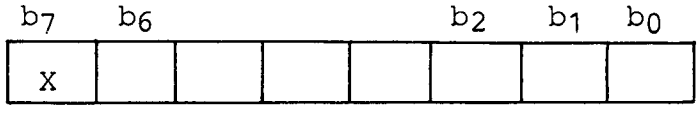


Fig. 6.26 Subscriber Station descriptor

extension descriptor



b_0 : active
b_1 : passive
b_2 : locked out
b_3 : activating
b_4 : deactivating
b_5 : extension initiated activation
b_6 : exchange activated activation
b_7 : X

Main User descriptor

byte 1

b_0 : active
b_1 : passive
b_2 : locked out
b_3 : activating
b_4 : deactivating

byte 2 b_0 ... b_4 : time slot allocated to this user.

The descriptor for the Additional User equipment is identical with this.

The remaining two descriptors are not considered further, as no reference is made to these descriptors in the application programmes - they are included for completeness.

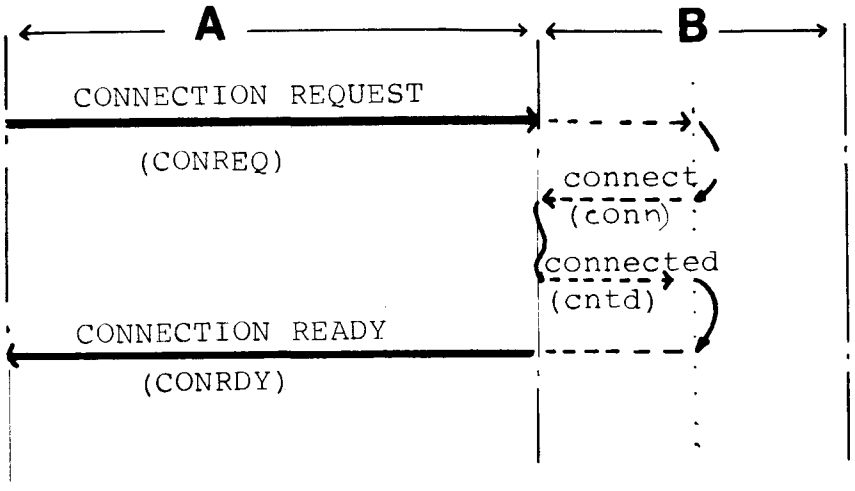
Phase I: System (de)Activation

This deals with the state of a subscriber loop, i.e. whether it is in the active or idle state. In order to create a connection with a user this must be active and, when no users are active within the particular station, this loop will be forced to enter the idle state.
The activation of the loop will consist of the energisation of the power supply in the subscriber station followed by the synchronisation of the transmission system between station and the FEU. Due to the fact that the precise details of how these two items are implemented are not known (several possibilities exist), the sequence of actions describing these occurrences must be kept global. Messages then, will be transferred to and from the specific message handler concerned(the precise methods by which the power supply activation and synchronous operation is achieved will not be considered in detail, although they are assumed to occur within the LTU of the particular extension thus addressed).
In the following diagrams the three regions in which messages are transferred are:

A: between subscriber station and FEU
B: internal to the FEU
C: between FEU and CPU.

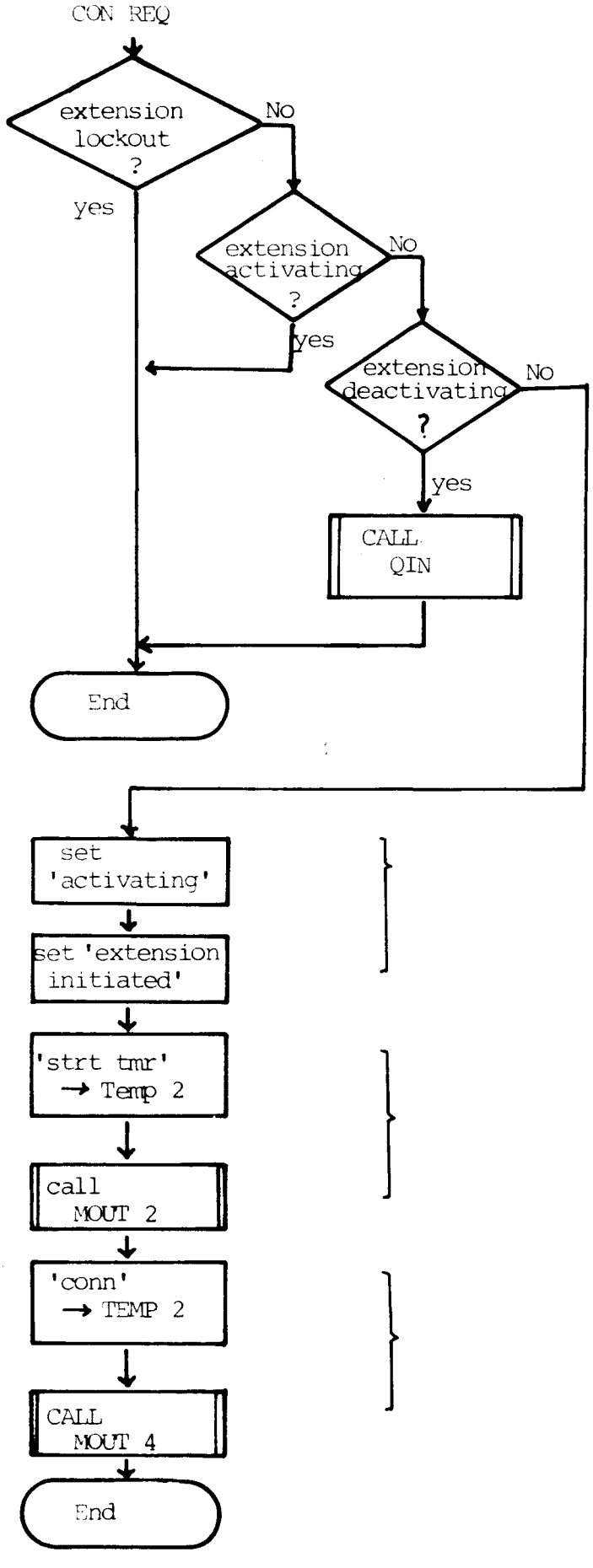
a) SYSTEM ACTIVATION

1) Initiated by the extension



Here a connection request is generated in some way from the subscriber station. This leads to the power supply being activated and synchronisation of the transmission system being carried out by the LTU - via a connect command from the control function. On completion of these processes the connection ready indication is given to the control functio which then transfers a CONNECTION READY message to the subscriber station. Phase II may then be entered.
The actions required are shown globally in the flow charts of fig. 6.26

As is seen from these flow charts, a coupling between the independent application programmes used to process each message is produced via the 'timer' used. These timers are described in appendix C.
This technique is also used in the flow charts occurring in later sections.



routine for handling a 'connection request' message

test if extension is in the lockout state or if it is already activating

if so ignore the message

if not check whether it is deactivating

if so return the message (CON REQ) to the queue, for further handling at a later time

end of routine-return to main programme

if none of the above conditions are met then the extension is passive

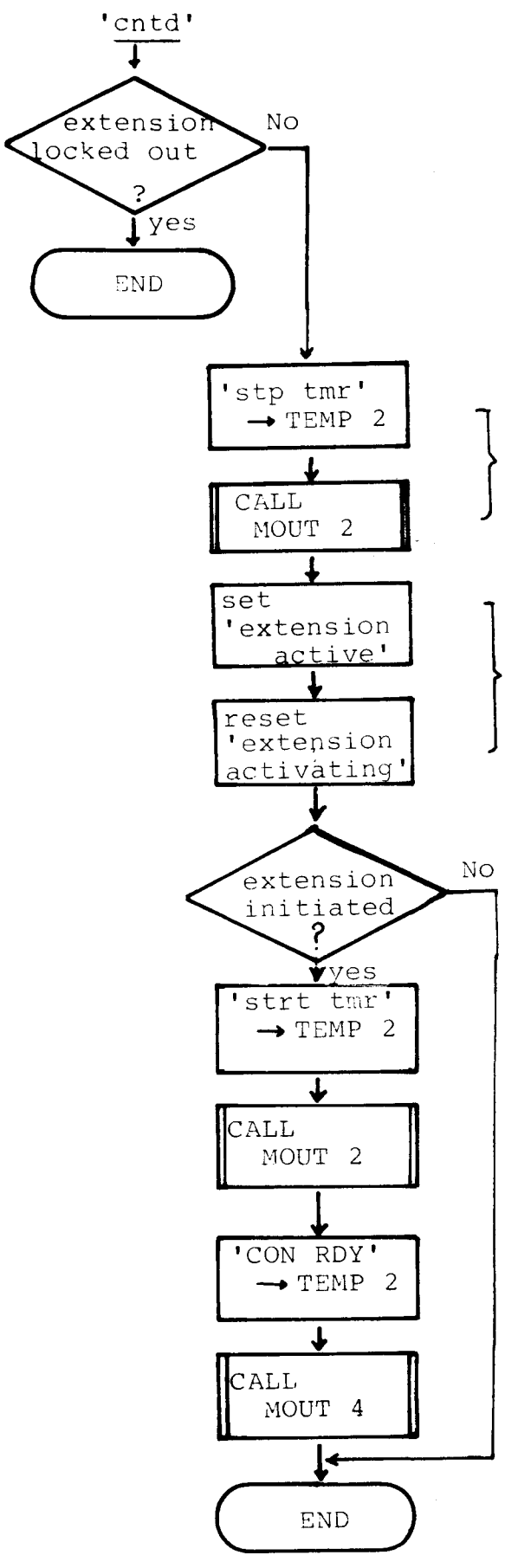
update the extension descriptor

start the timer (class. O) i.e. transfer 'strt tmr' message to Real Time Function

transfer 'connect' message to specified message handler

end of routine - return to the main programme (i.e. wait for reply to 'conn')

Fig. 6.26a



routine for 'connected' message

check if an extension lockout condition is active

if so ignore the message

stop the timer

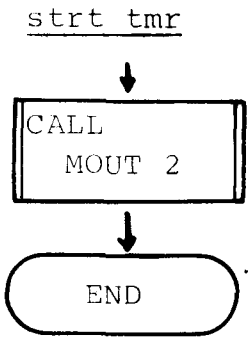
update the extension descriptor

start the timer for the next phase (class 1)

transfer a 'connection ready' message to the extension concerned

end of routine

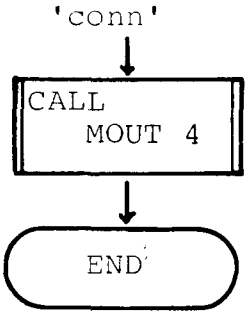
Fig. 6.26b



transfer 'strt tmr' message to Real Time Function

end the routine

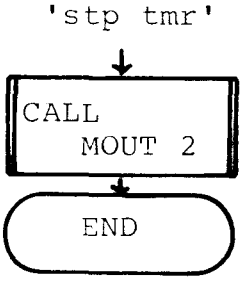
Fig. 6.26 C.1.



transfer 'conn' message to message handler

end the routine

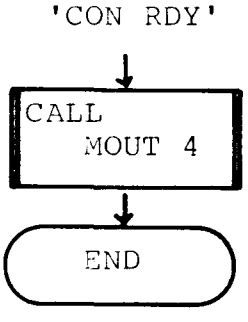
C.2.



transfer 'stop timer' message to Real Time Function

end the routine

C.3.

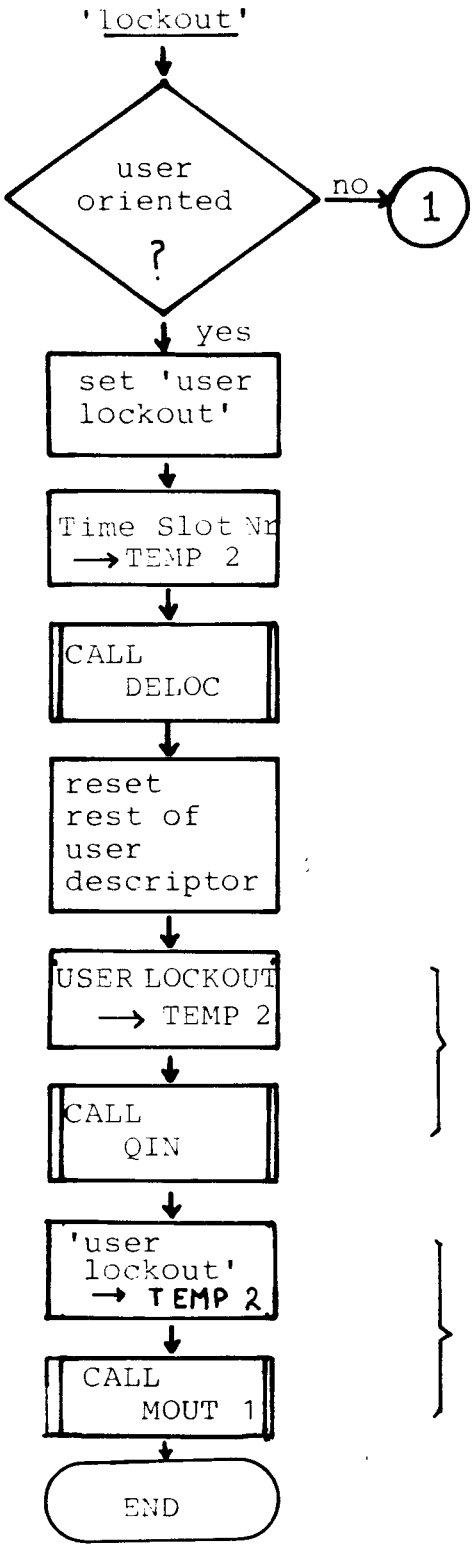


transfer 'connection ready' message to specific extension

end of routine

C.4.

These messages may have been returned to the queue by the corresponding output routine, and may therefore have to be processed again at a later time.



routine for handling 'time out alarm' from the Real Time Function

is the alarm message related to a specific user or to the extension

set lockout bit in user descriptor

fill TEMP 2 with the time slot (in byte 2 of descriptor) allocated to this user

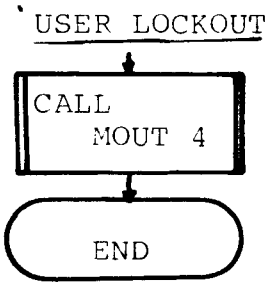
free the time slot

reset the descriptor variables

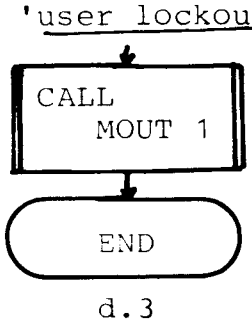
enter a 'USER LOCKOUT' message in the queue for transfer to the user equipment at a later time

transfer a 'use lockout' message to the CPU

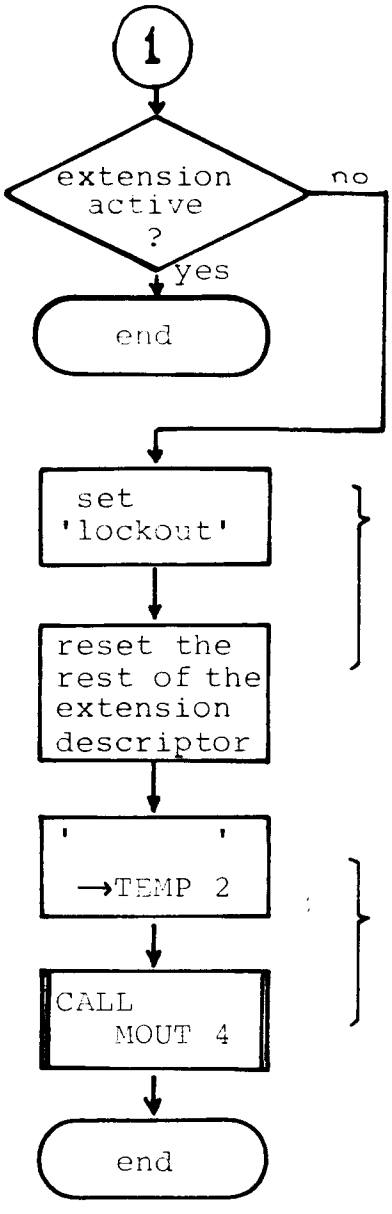
Fig. 6.26d.1



d.2



d.3



check if the extension has become active

if so ignore the message and end the routine

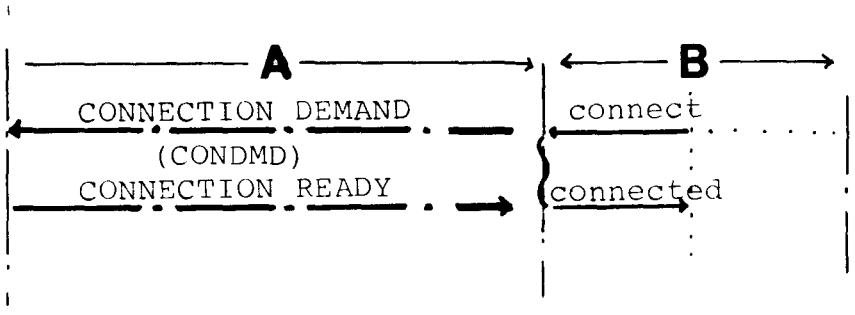
update the extension descriptor

transfer a Reset message to the message handler

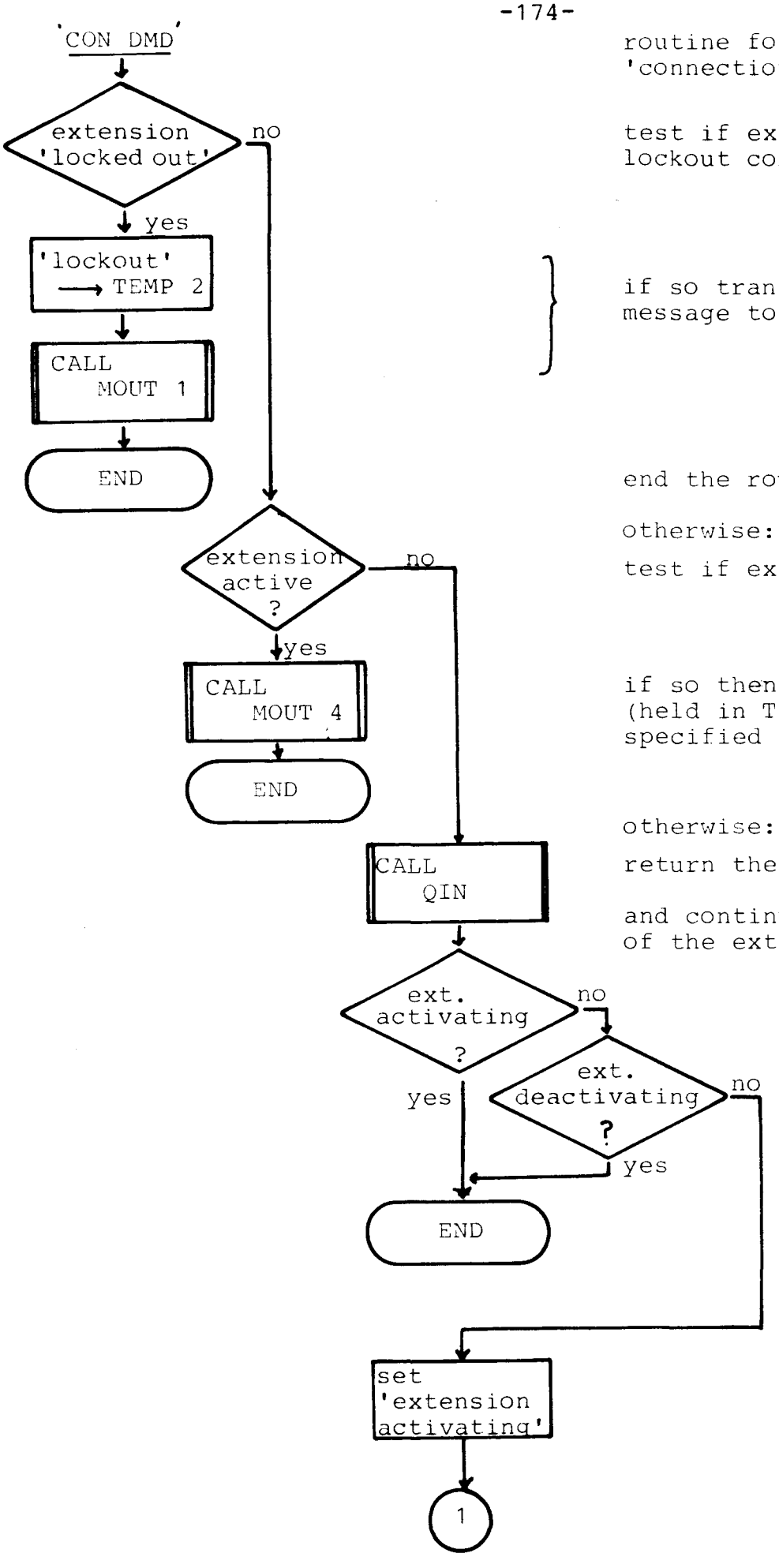
end the routine

Fig. 6.26d4

2) Initiated by exchange



Here a connection is to be created on command from the CPU (the actual messages involved from the CPU are discussed in combination with Phase II).
As in the previous case, the LTU is forced, via a connect message, to activate the connection.
This is equivalent to an abstract 'CONNECTION DEMAND' message to the subscriber station (this could of course be carried out as an independent message transfer). When the connection has been activated a connection ready message is transferred to the control function (equivalent to 'CONNECTION READY'), and the next phase may be entered.
A flow chart for this is shown in fig. 6.27.



routine for handling a 'connection demand' message

test if extension is in the lockout condition

if so transfer a 'lockout' message to the CPU

end the routine

otherwise:

test if extension is active

if so then transfer the message (held in TEMP 1, 2) to the specified user

otherwise:

return the message to the queue

and continue with the activation of the extension;

if the extension is already in the (de)activating state end the routine

otherwise:

set extension activating bit in the descriptor

Fig. 6.27a

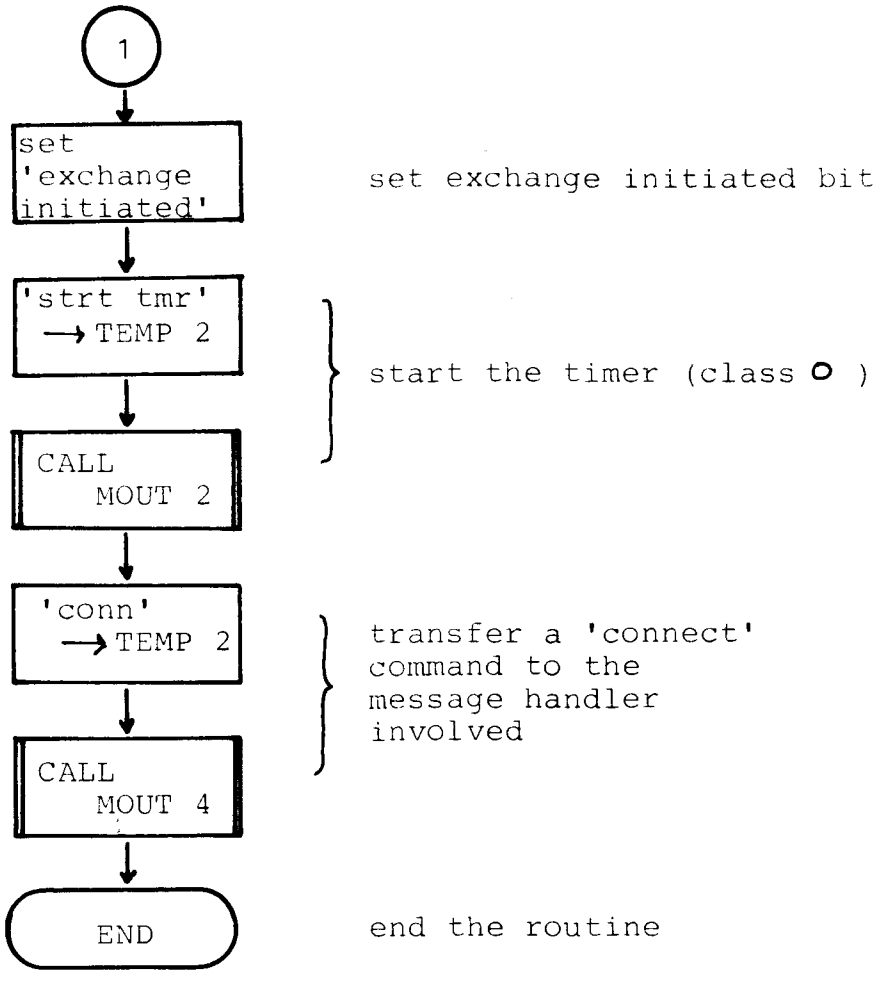
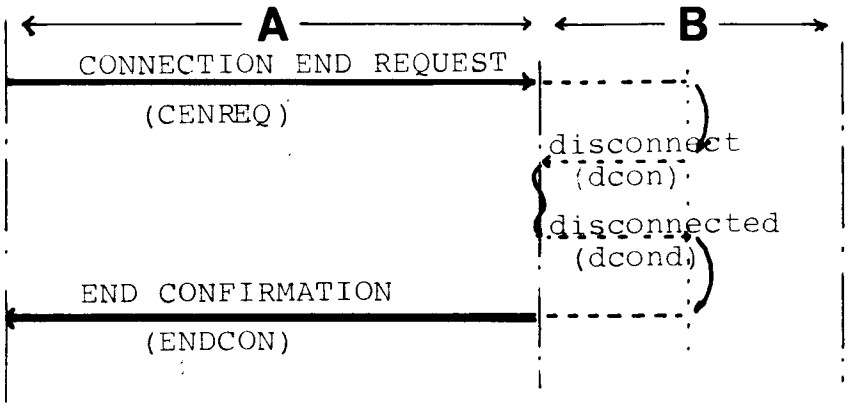


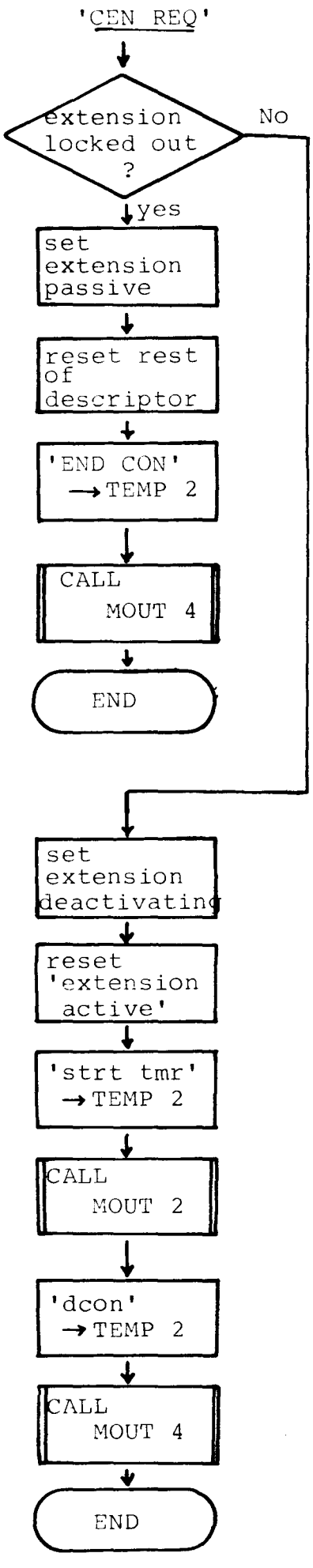
Fig. 6.27a (continued)

b) SYSTEM_DEACTIVATION

In general this is always initiated by the subscriber station (although a lockout condition can be created in the FEU).



A CONNECTION END REQUEST message is transferred from the subscriber station (e.g. after the last use or it has gone inactive).
This causes the control function to give a disconnect command to the LTU, which then deactivates the subscriber loop and indicates the completion of this by a disconnected message. The END CONFIRMATION (which may be abstract and only used to reset the LTU) is then transferred to the station and causes system reset.
This action is described in the flow chart of fig. 6.28.



routine for handing a 'connection end Request' message

check if extension is in the lockout condition

if so set extension passive,

and update the rest of the descriptor

transfer an 'End Confirmation' signal to the corresponding handler - indicating a reset condition

end the routine

otherwise:

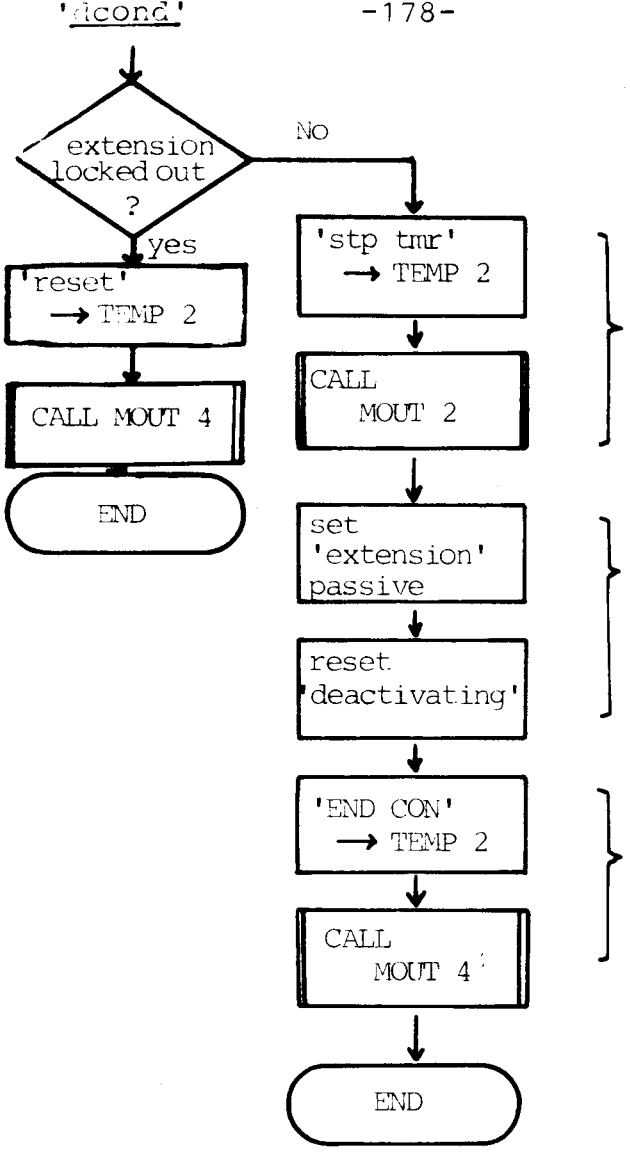
update the extension descriptor

start the timer (class o)

transfer a disconnect command to the message handler

end the routine

Fig. 6.28a



routine for handling a 'disconnected' message

check if extension is locked out if so give a 'reset' command to the message handler

otherwise:

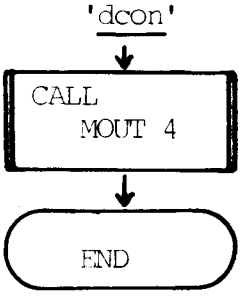
stop the timer

update the extension descriptor

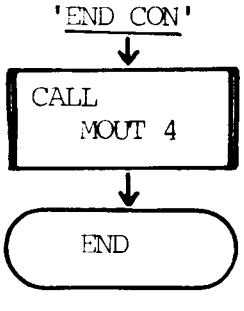
transfer an 'End Confirmation' message to the message handler

end of routine

FIG. 6.28b



C.1.



C.2.

Fig. 6.28c

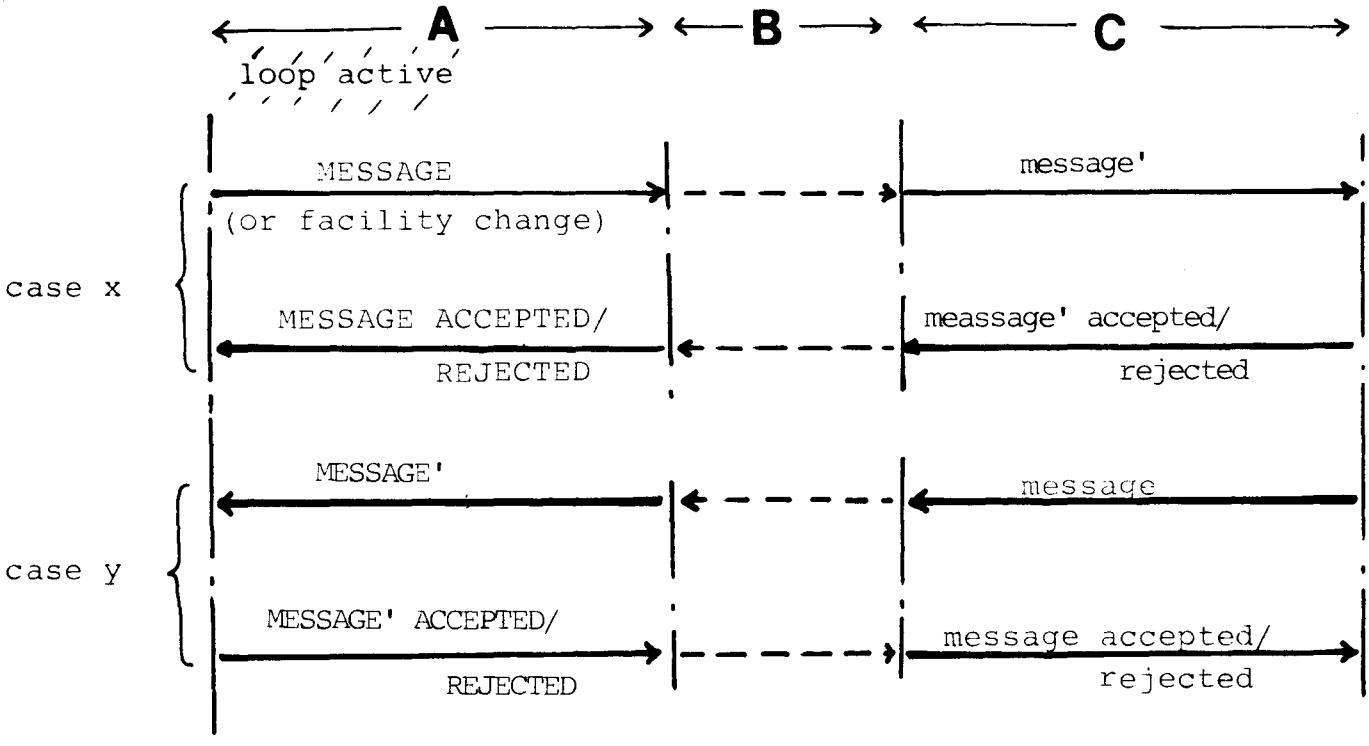
Phase II: User (De)Activation

This phase deals with the (de)activation of the specific user equipments and the creation of the connection with a particular far end user.
This phase may commence only when an active subscriber loop is, or has been made, available.
The activation phase leads, typically, to the build up of an end to end connection over which information (data or speech) may be transferred.
The deactivation phase leads to the break down of this connection.
Two basic sets of message transfers can be distinguished:

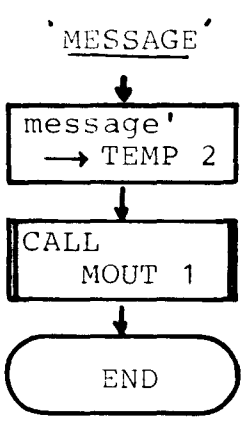
1) in which the second user (the called party) is the CPU of the exchange. This is distinguished in that no time slot on the I.H. is required, as no user information is transferred. Single messages are transferred, indicating a given state in the corresponding equipment or a facility change request from a user. The common equipment and telemetry units are assumed to operate exclusively in this mode, while the other two user equipments may operate in this mode. The complete message transfer consists, in this case, of a message from the source unit and eventually a reply message from the CPU.

2) in which a full end to end connection (and thus time slot on the I.H.) is required. This occurs exclusively for the Main and Additional user equipments and leads to the full build up sequence, shown in fig. 5.26 , of call request, selection (dial) information and etc.

Mode 1: Simple Message Transfer



In this case the FEU must simply cause a corresponding message to be transferred to the CPU, if the received message source was a user equipment in the subscriber station, or to the specified user if the CPU was source.
The flow charts for these two operations, in which the 'message' can be one of a variety, are shown in figs. 6.29 and 6.30.
As is seen in case y, the subscriber loop may have to be activated before the message can be transferred to the required unit in the subscriber station. In order to achieve an orderly operation, the original message is returned to the queue in this case, and the activation of the subscriber loop initiated (CON DMD). The message may be processed several times after this, before a definite loop state is found (active or lockout) – this extra overhead is accepted in order to keep the system operation as simple as possible.



translate the message into the corresponding output form and transfer this to the CPU (for a & b) or message handler for c & d)

Fig. 6.29a

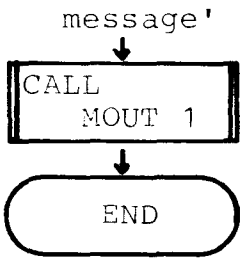


Fig. 6.29b

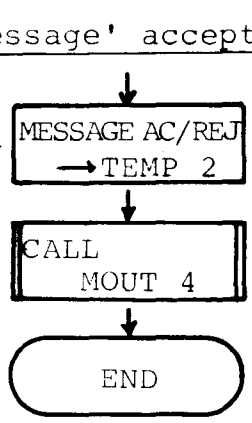


Fig. 6.29c

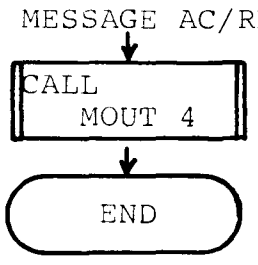
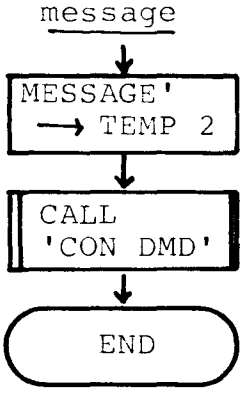


Fig. 6.29d

Fig. 6.29 shows the routines for case x, i.e. where an active loop was available



put corresponding message into TEMP 2

call 'connection demand' routine to carry out the message transfer to a message handler

Fig. 6.30a

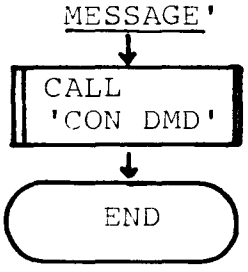


Fig. 6.30b

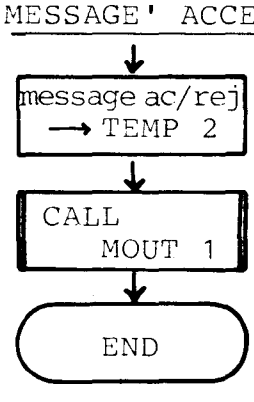


Fig. 6.30c

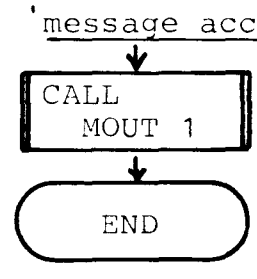


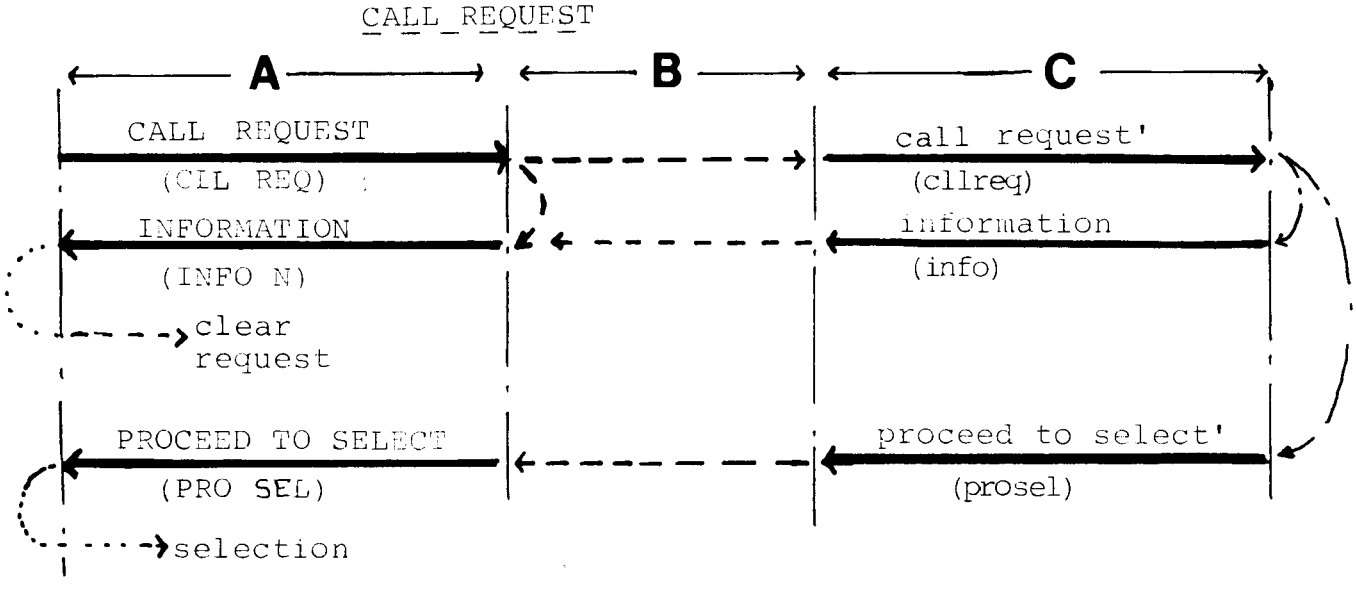
Fig. 6.30d

Fig. 6.30 shows the routines for case y, i.e. initiating from a message from the CPU and which may therefore cause an extension activation

Mode 2: Full connection (de)activation

This case leads to the full end to end connection build up procedure, a typical sequence of which was shown in fig. 5.26 The various messages involved will be considered in the following sections, along with the sequences of actions initiated by them.

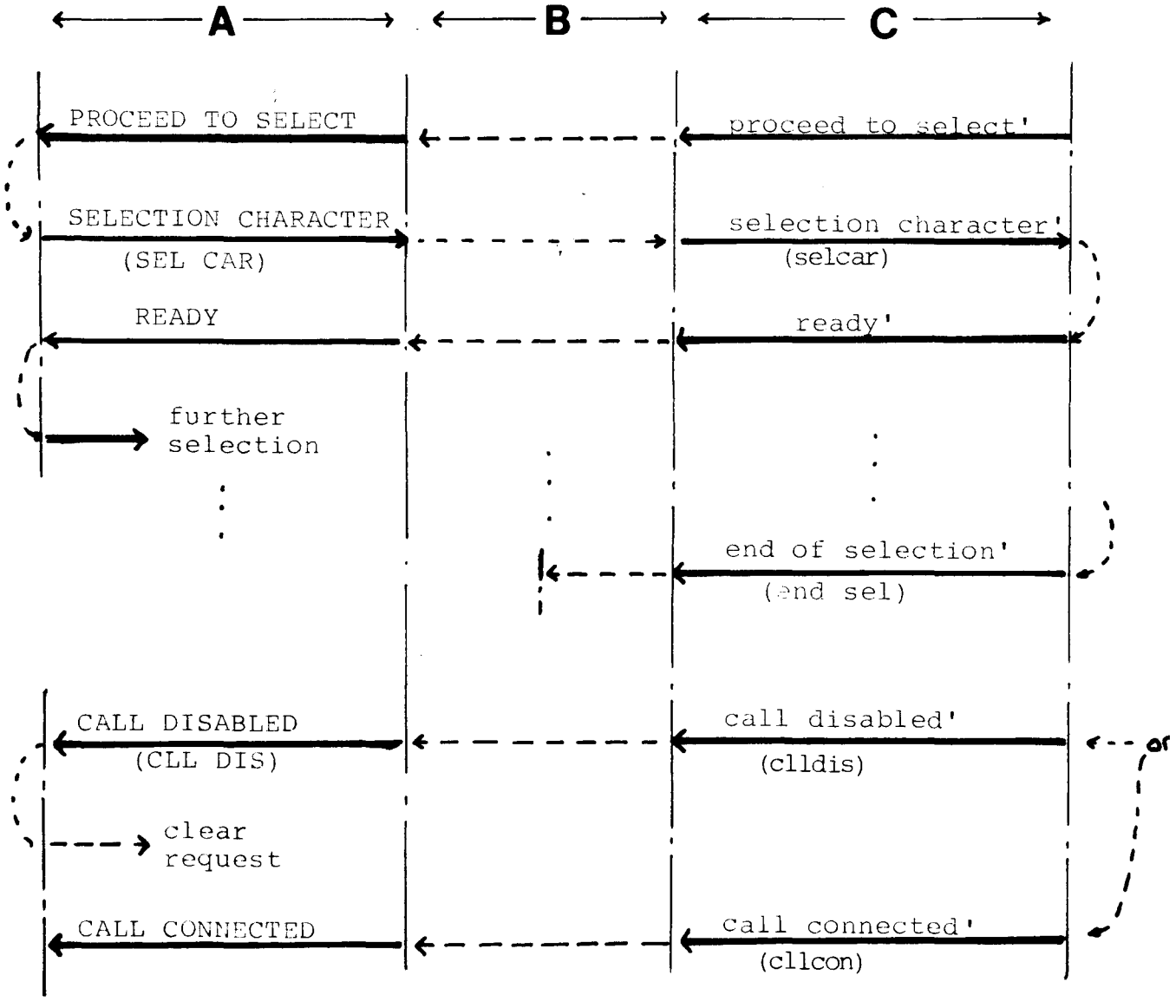
a) User initiated

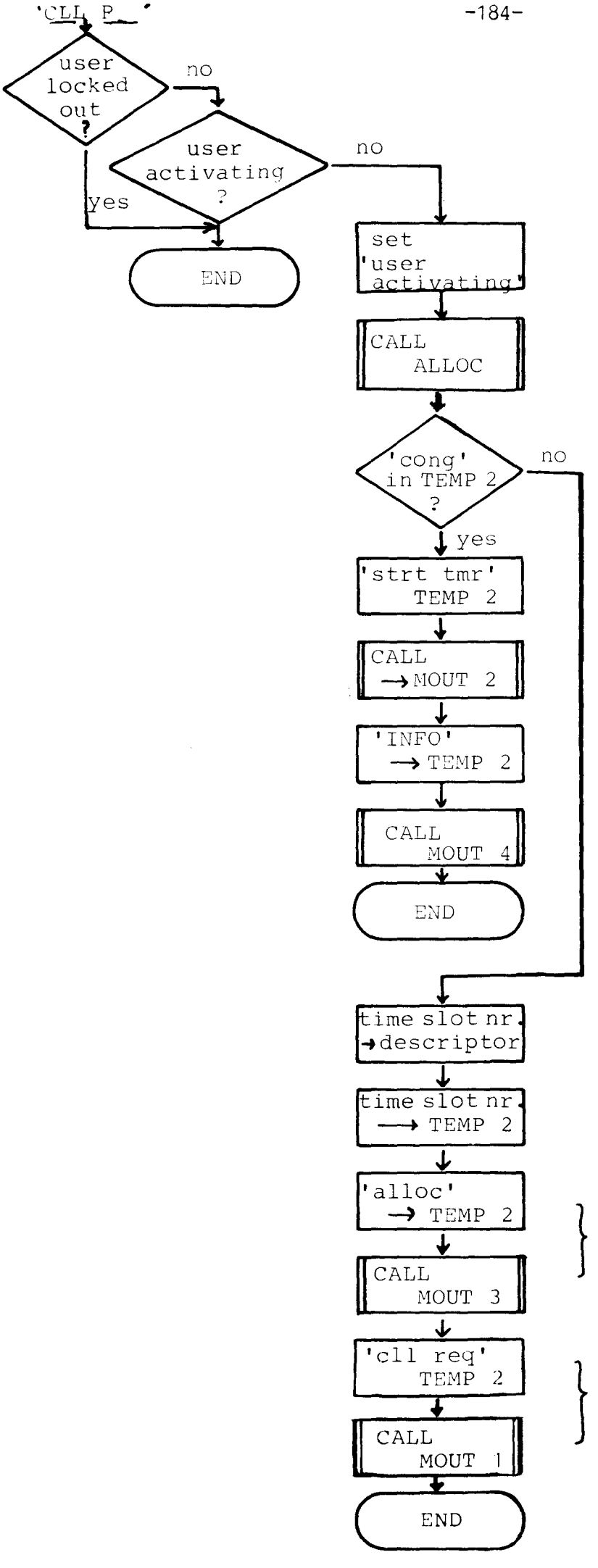


A connection build up is initiated, in this case, by a CALL REQUEST message from a user. This leads to the allocation and connection of an output circuit to this user by the FEU. Further a corresponding message, including the details of this time slot allocation, is transferred to the CPU. Should no free time slot be available an INFORMATION message (e.g. congestion) is returned to the user.
In the former case, the CPU replies with either a proceed to select command or an information signal. These messages are translated in the FEU and the result transferred to the specific users.

Audio tones may be inserted in the user circuit by the CPU, as the circuit has been connected as far as the main body of the exchange.
The next expected messages from the user are then selection information, respectively clear request, this latter message also being possible at any time during the connection.
Flow charts for these basic actions are given in figs. 6.31 and 6.32.

SELECTION





routine for handling a 'connection request' from a user

test if either the user is locked out or activating, and if so ignore the message

otherwise:

set user activating flag in the descriptor

call the routine for allocating an output circuit (i.e. time slot)

test if a time slot has been allocated

if not then:-

start the timer (class 4)

transfer 'INFORMATION' message to the user

end of routine

otherwise:

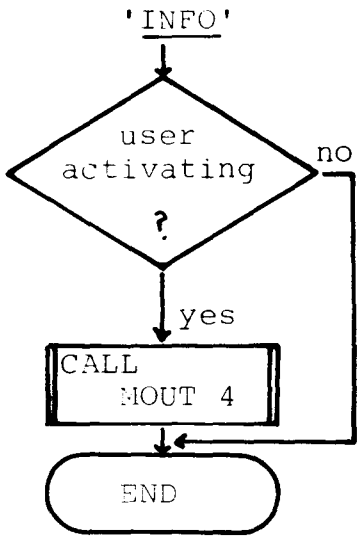
enter time slot number in the descriptor

enter time slot number in TEMP 2

add an 'allocation' command and transfer to the Interface Management Function

transfer a 'call request' message (including time slot allocation) to the CPU

Fig. 6.31



test if user is still in the activating state

if not ignore the message

otherwise transfer the message to the specified message handler

Fig. 6.32a

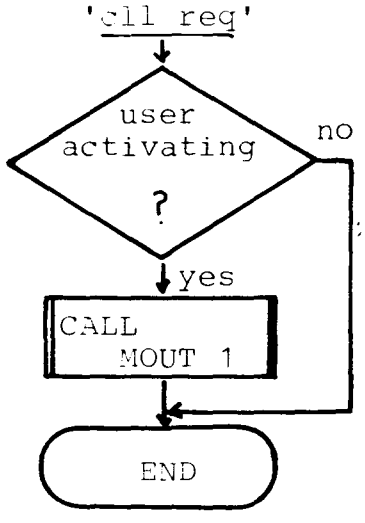


Fig. 6.32b

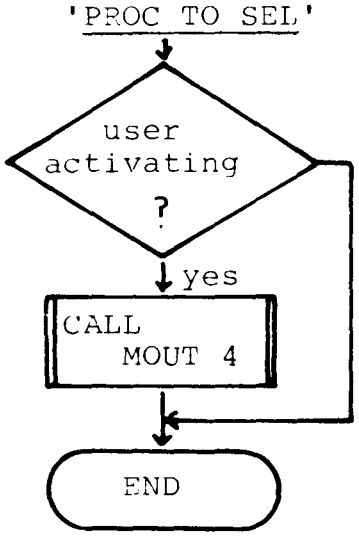
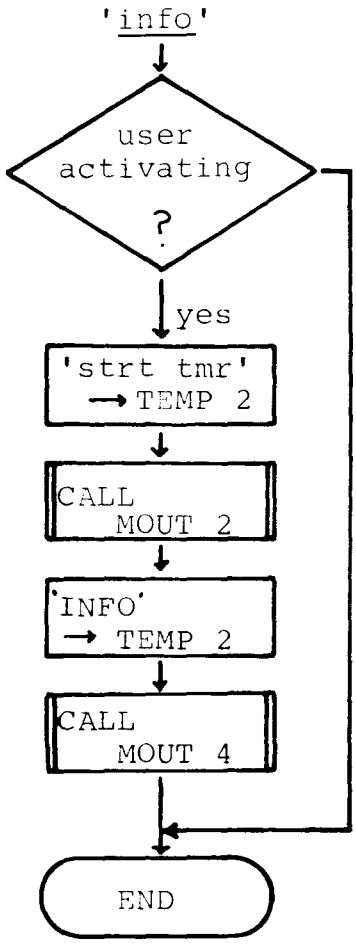


Fig. 6.32c



test if user is still in the'activatin state

start timer (class 4)

transfer 'INFO'message to message handler

Fig. 6.32d

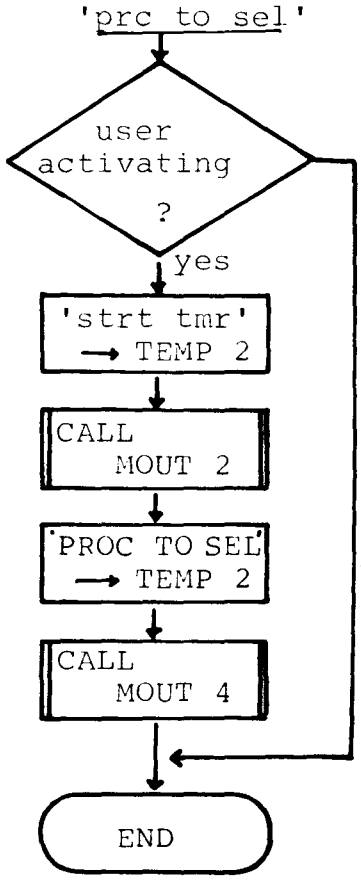
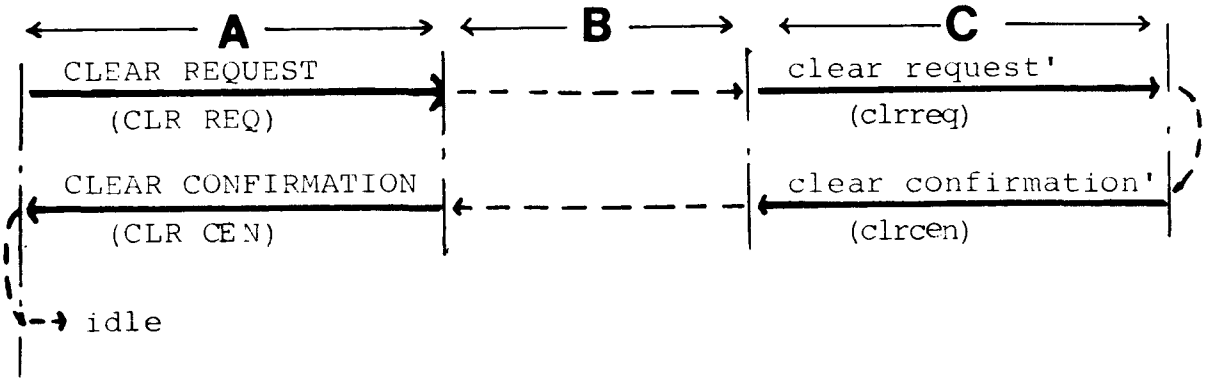


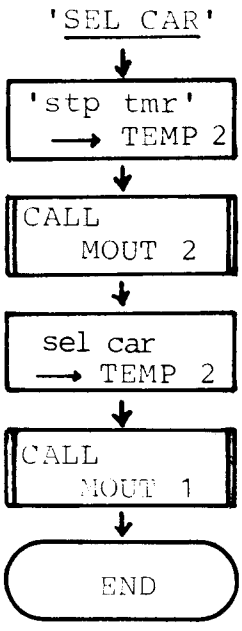
Fig. 6.32e

On receipt of a PROCEED TO SELECT message the user may continue with the identification of the called party (via selection characters). The FEU transfers these selection characters to the CPU, which acknowledges each in turn (as has been described previously other acknowledgement techniques could be used). The end of selection is detected in the CPU and a message indicating this state is transferred to the FEU (this carries out a management function which consists of the ending of this phase of the build up procedure).
A flow chart of this is given in fig. 6.33.
Eventually this selection leads to a reply from the called party - either a call connected or call disabled message. These are simply transferred via the FEU to the user.
The ready for data message is likely to occur within the user circuit. If this occurs exclusively so, then the call connected message would be used, in the signalling system of the user equipment, to indicate the system state. Otherwise a dedicated message could be used to do this.

Fig. 6.34 shows flow charts for the routines handling these messages.

CLEAR REQUEST





routine for handling a selection character (16 different routines)

stop the timer

transfer the corresponding character to the CPU

Fig. 6.33a

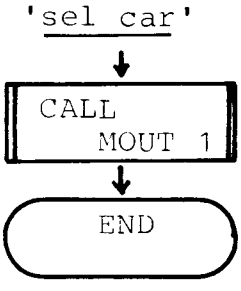


Fig. 6.33b

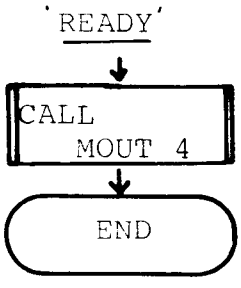
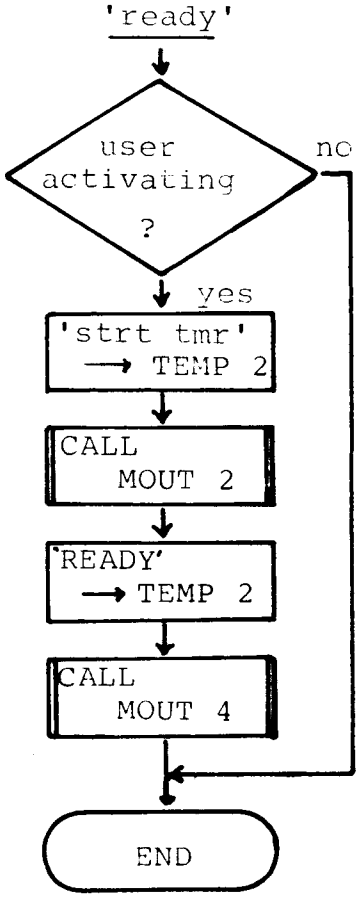


Fig. 6.33d



test if user is still in the activating state
-if not ignore the message

start the timer (class 3)

transfer a 'READY' message to the message handler

Fig. 6.33c

'cll dis'

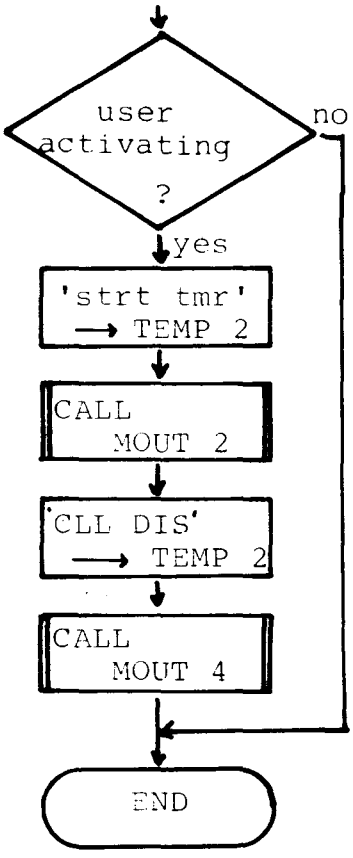


Fig. 6.34a

'CLL DIS'

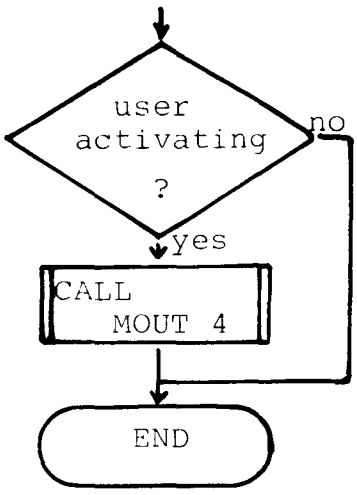


Fig. 6.34b

'cll con'

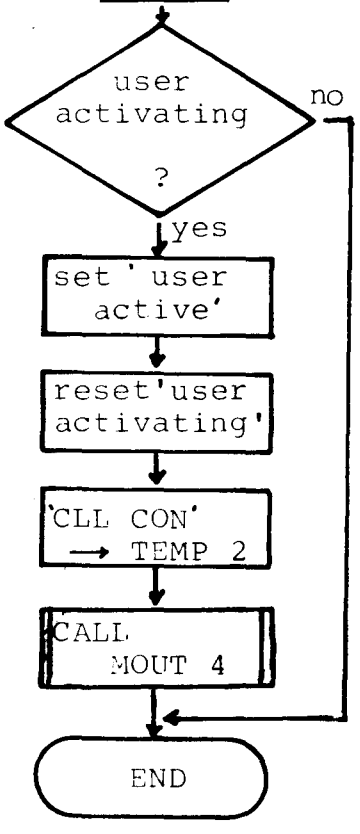


Fig. 6.34c

'CLL CON'

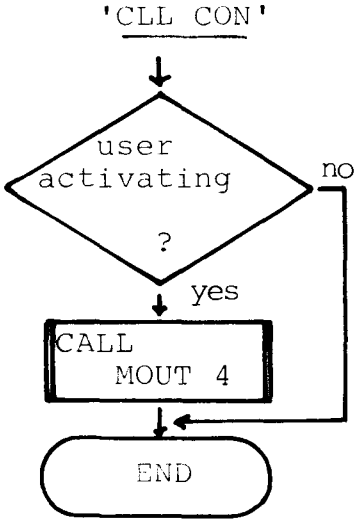
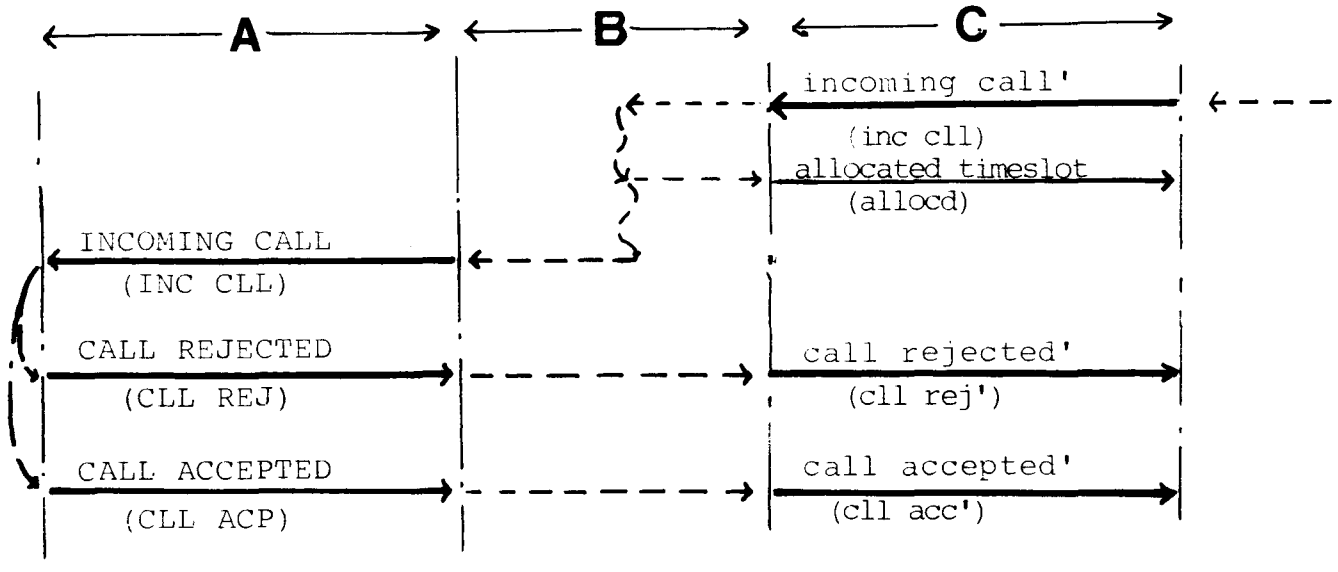


Fig. 6.34d

Here the user initiates the deactivation of the end to end connection via a CLEAR REQUEST message. The FEU then frees any time slot allocated to this user, carries out other management functionsand transfers a corresponding message to the CPU. The reply to this is a clear confirmation to the user, as shown above, which results in this equipment entering the 'idle' state.

The flow chart of this routine is given in fig. 6.35

b) Exchange Initiated



Here the CPU indicates that an incoming call is present for a given user. The FEU then causes the corresponding subscriber loop to be activated, if this is necessary. On completion of this a time slot pair is allocated on the I.H. and the details of this allocation are transferred to the CPU (this is necessary due to the possible injection of any audio tones).

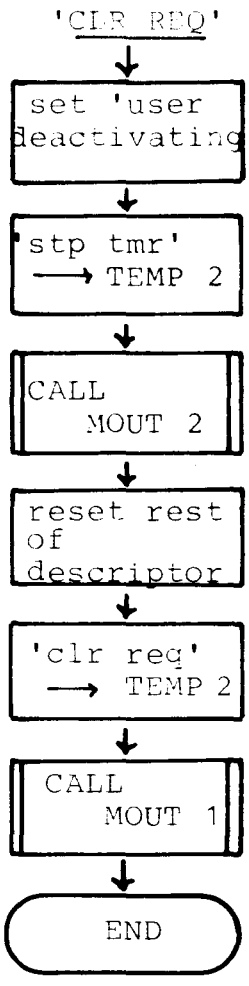


Fig. 6.35a

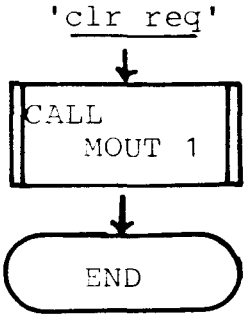


Fig. 6.35b

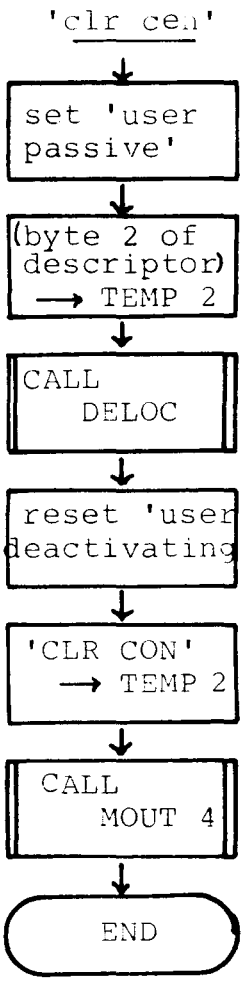


Fig. 6.35c

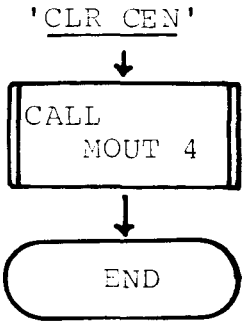
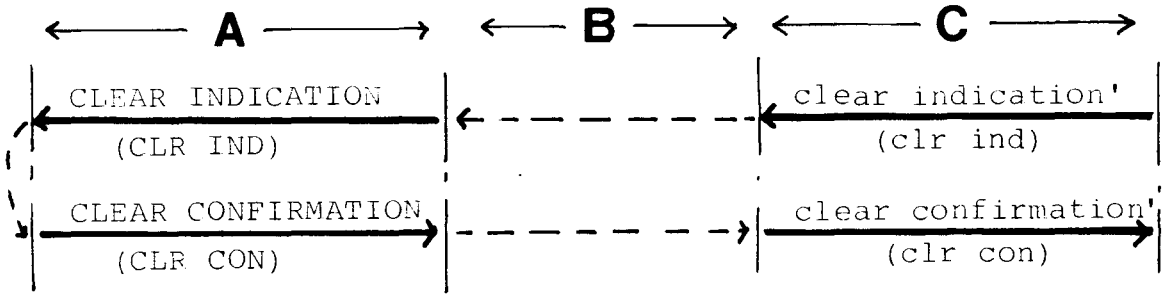


Fig. 6.35d

Then, an incoming call message is transferred to the specified user. This leads to either a call rejected message from the user, which in turn is transferred to the CPU and initiates the freeing of the time slot allocated, or to a call accepted message which is also transferred to the CPU. This latter case then leads to the 'ready for data' state as described in the previous sections.

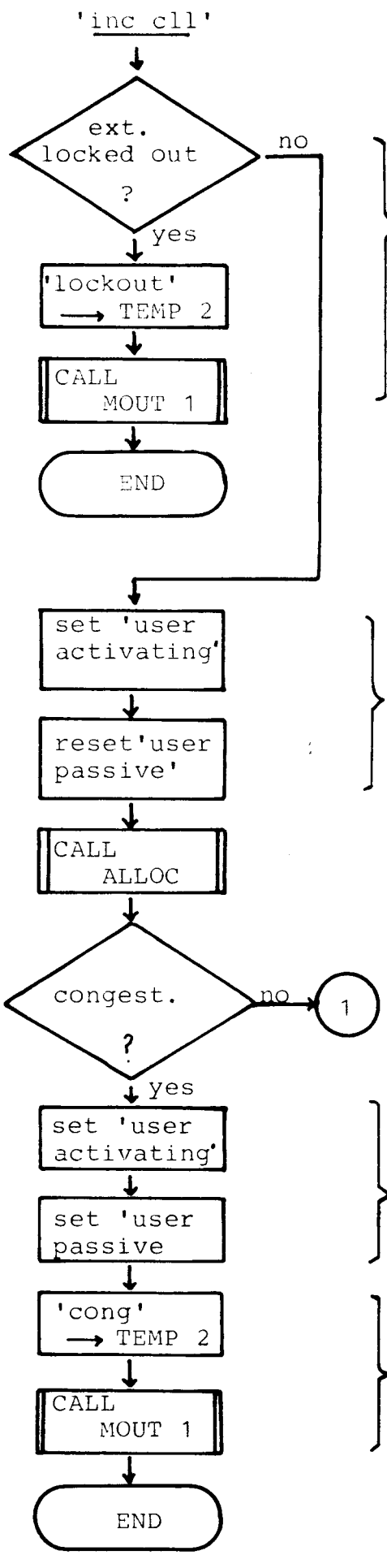
Flow charts for these routines are given in fig. 6.36 and 6.37.

CLEAR INDICATION



The CPU indicated, via a clear indication message, that a given connection is to be ended. The corresponding message is transferred to the user, whose reply is expected to be a clear confirmation message. On receipt of this, the FEU ends the connection and transfers the corresponding message to the CPU.

Fig. 6.38 shows a flow chart of this routine.



routine for handling an 'incoming call' message

check if extension is locked out - if so signal this fact to the CPU

otherwise:

update the user descriptor

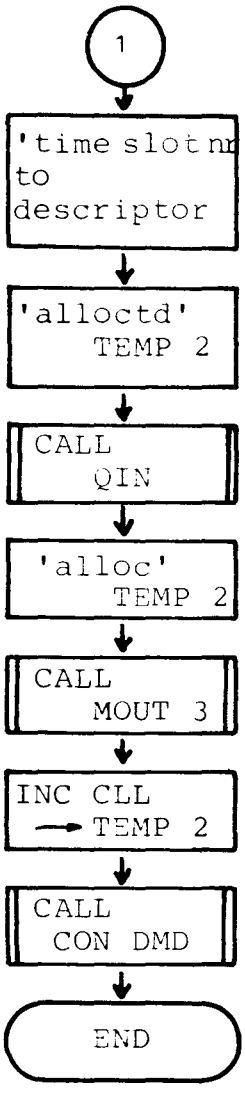
call the time slot allocation routine

test if a timeslot is available

if no free time slot is available update the user descriptor

signal this congestion condition to the CPU

Fig. 6.36a



otherwise:

enter the time slot allocated into the second byte of the user descriptor

enter the allocation details into the queue for transfer to the CPU

activate the allocation in the Interface Management Function

cause the extension to be activated, if necessary, and the 'INCOMING CALL' message to be transferred to the specified user equipment

end of the routine

Fig. 6.36a (continued)

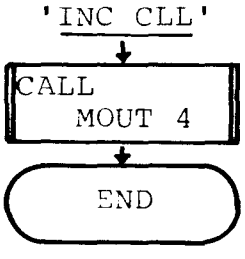


Fig. 6.36b

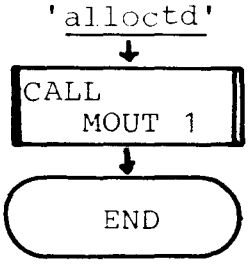


Fig. 6.36c

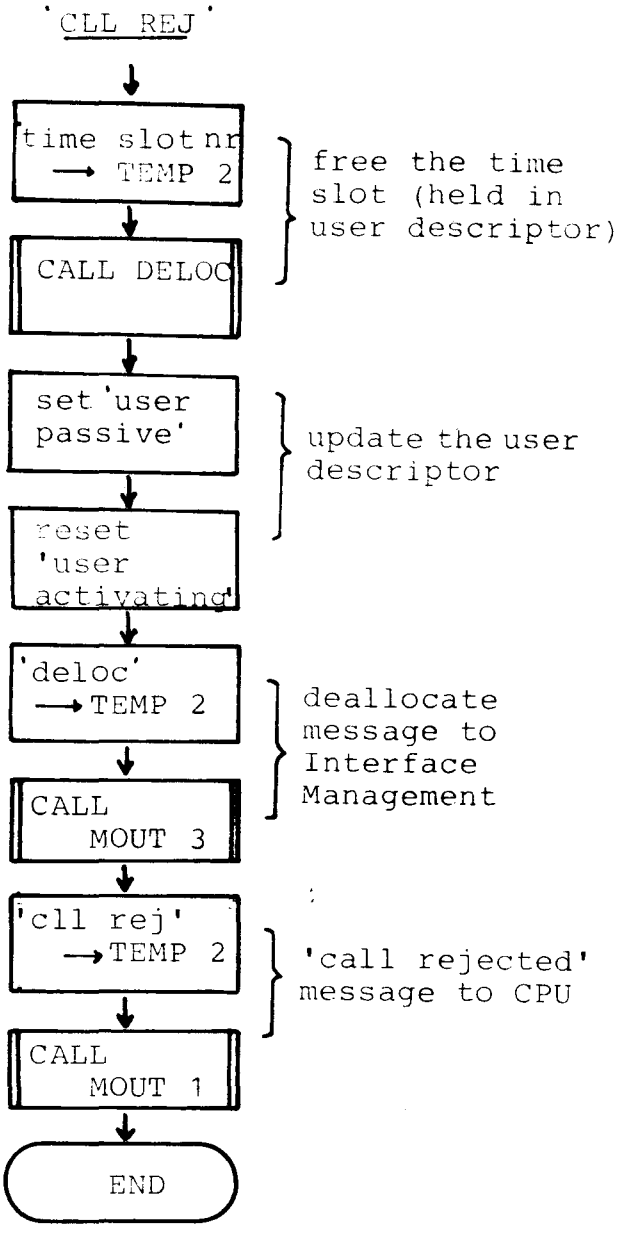


Fig. 6.37a

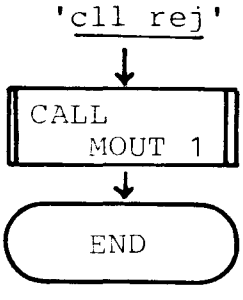


Fig. 6.37b

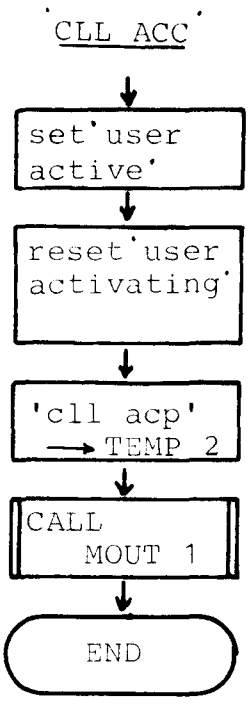


Fig. 6.37c

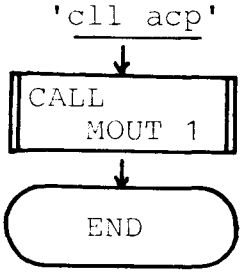
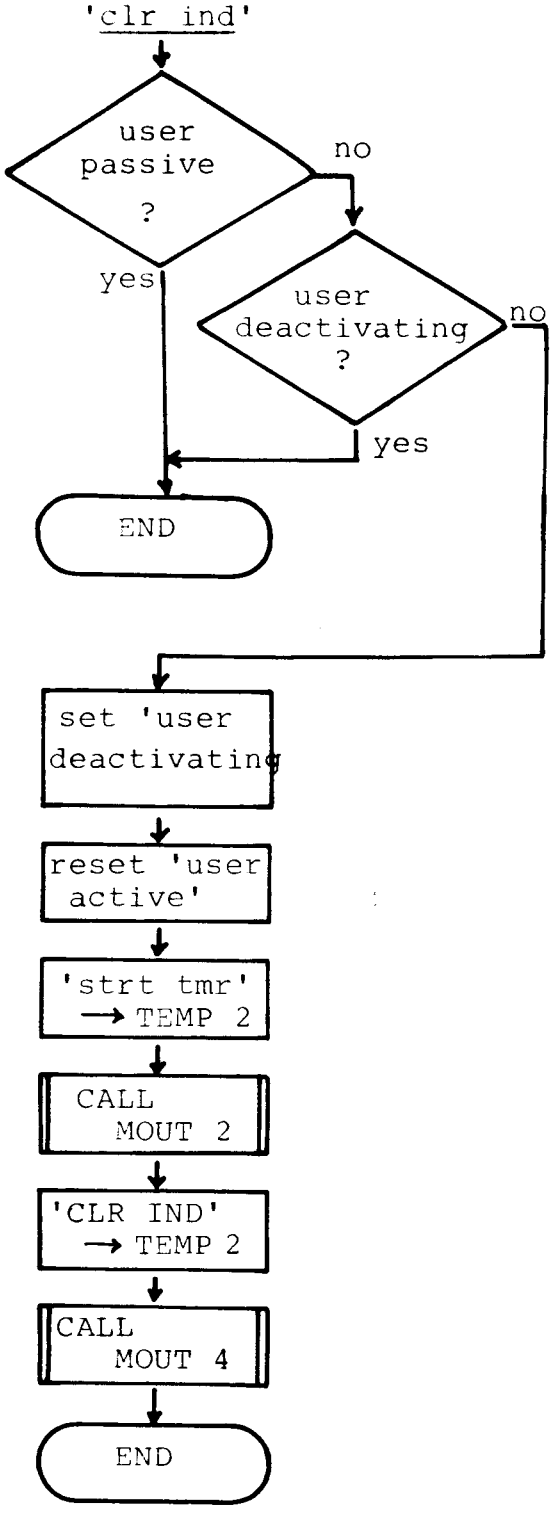


Fig. 6.37d

Routine for handling a 'clear indication' message



if the user is(or is becoming) passive,then ignore the message.

otherwise:-

update the user discriptor

start timer(class 4)

transfer a 'CLEAR INDICATION' message to the user.

Fig. 6.38.a

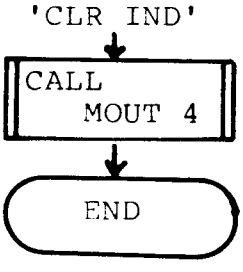


Fig.6.38.b

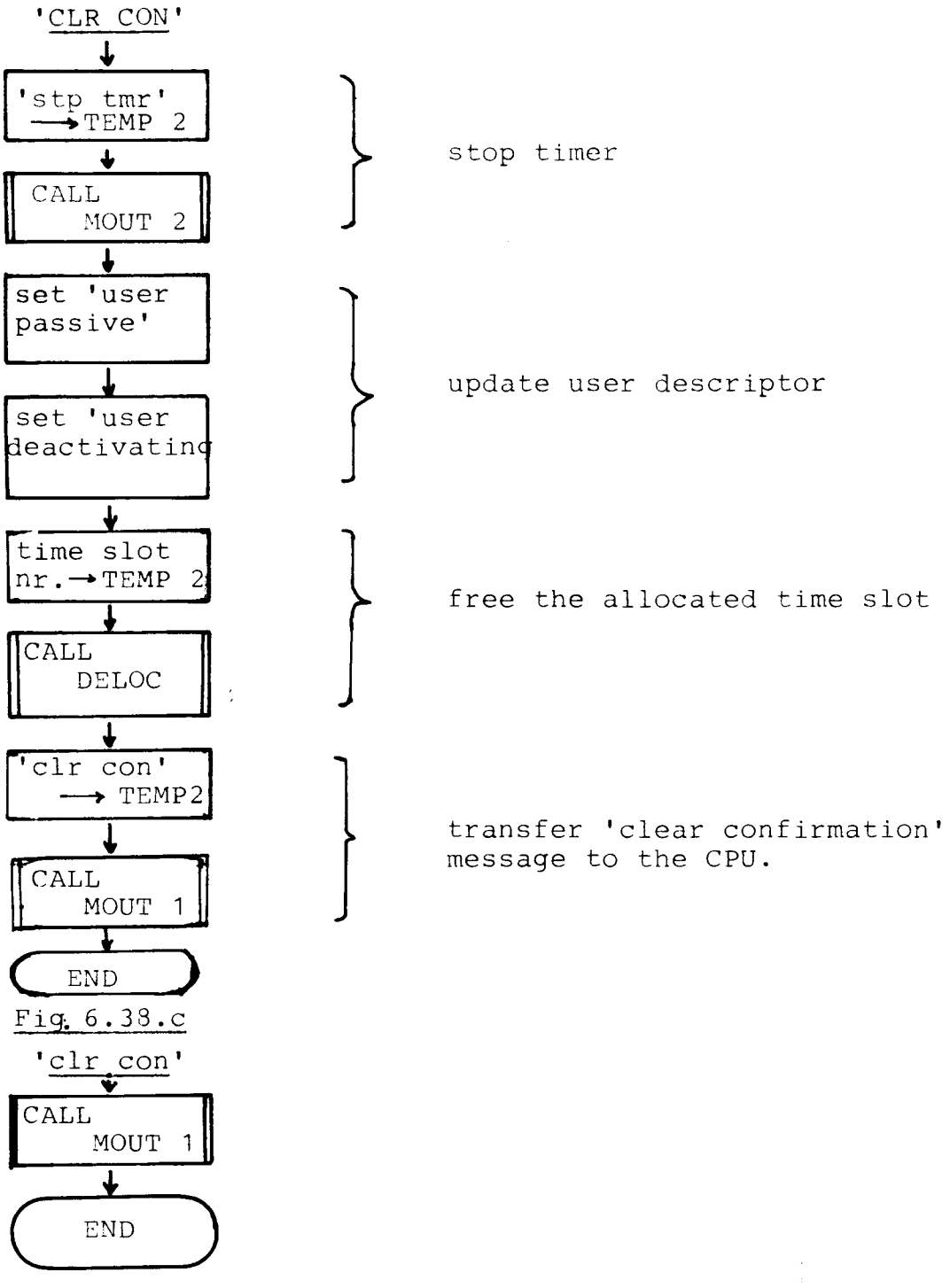


Fig. 6.38.c

Fig.6.38.d

```
61        LD      (HL),B          BYTE 1 TO PERIPHERAL
62        LD      (HL),C          BYTE 2 TO PERIPHERAL
63        JP      PROC            TRANSFER COMPLETE
64        SPC
65        EXTRN   PROC,DIN
66        ENTRY   MOUT1,MOUT2,MOUT3
67        SPC
68        END
```

```
1         TITL    TIMESLOT MANAGEMENT ROUTINES-AFS11;S
2  AFS11  CSECT
3         SPC
4  *              ALLOCATION ROUTINE
5  ALLOC  LD      A,(UCNT)              CHECK IF ALL
6         SUB     H'1F'                 TIMESLOTS ARE
7         JP      M,TESTI               IN USE
8         SPC
9         SET     7,C                   CONGESTION MESSAGE IN REG.C
10        RET
11        SPC
12 TESTI  LD      HL,(NPNT)             TEST IF TIMESLOT
13 TESTY  BIT     7,(HL)                BEING CHECKED IS
14        JP      Z,SEIZE               FREE,AND SEIZE
15        SPC
16        SPC
17        LD      DE,MAX
18        SBC     HL,DE                 GO TO NEXT
19        JP      Z,RSTPN
20        INC     HL
21        JP      TESTY                 TIMESLOT
22 RSTPN  LD      HL,BASEA              AND TRY
23        JP      TESTY                 AGAIN
24        SPC
25 SEIZE  LD      (HL),B                USER ID. TO
26        SET     7,(HL)                DESCRIPTOR AND
27 *                                    ACTIVATE
28        LD      A,(UCNT)
29        INC     A                     INCREMENT USER
30        LD      (UCNT),A              COUNT
31        INC     HL                    UPDATE
32        LD      (NPNT),HL             NPNT
33        DEC     HL                    POINTER
34        LD      DE,BASEA
35        SBC     HL,DE                 TIMESLOT
36        LD      C,L                   NUMBER IN C
37        SPC
38        RET                           END OF ALLOC.
39        SPC
40        SPC
41        SPC
42 *              DEALLOCATION ROUTINE
43 DELOC  LD      A,C
44        AND     H'1F'                 SET NPNT TO
45        LD      E,A
46        LD      D,H'00'               REQUIRED
47        LD      HL,BASEA              TIMESLOT
48        ADD     HL,DE
49        LD      (NPNT),HL             POSITION
50        SPC
51        RES     7,(HL)                FREE TIMESLOT
52        SPC
53        LD      A,(UCNT)              UPDATE
54        DEC     A                     USER
55        LD      (UCNT),A              COUNT
56        SPC
57        RET
58        SPC
59        SPC
60        EXTRN   UCNT,NPNT,MAX,BASEA
61        ENTRY   ALLOC,DELOC
62        SPC
63        END
```

```
 1        TITL    APPLICATION PROGRAMMES;AFS12;S
 2 AFS12  CSECT
 3        SPC
 4 FDESC  LD      HL,BASED        LOAD HL WITH FILE
 5        LD      A,B             BASE POSITION
 6        AND     H'1F'           LOAD ACCU WITH BYTE 1
 7        SLA     A               OF MESSAGE
 8        SLA     A
 9        SLA     A
10        SPC
11        LD      E,A             FIND DESCRIPTOR
12        LD      D,H'00'         POSITION
13        ADD     HL,DE
14        SPC
15        RET                     RETN TO CALLING PROCESS
16        SPC
17        SPC
18        SPC
19        SPC
20 CONRQ  CALL    FDESC
21        BIT     2,(HL)          TEST LOCKOUT
22        JP      NZ,NOGO
23        BIT     3,(HL)            "  ACTIVATING
24        JP      NZ,NOGO
25        BIT     4,(HL)            "  DEACTIVATING
26        JP      Z,ACTVT
27        SPC
28        CALL    DIN
29        JP      NOGO
30        SPC
31 ACTVT  SET     3,(HL)          SET
32        SET     5,(HL)          SET
33        LD      C,H'42'
34        CALL    MOUT2
35        SPC
36        LD      C,H'40'
37        JP      MOUT4
38        SPC
39 NOGO   JP      PROC
40        SPC
41        SPC
42        SPC
43        SPC
44 CONTD  CALL    FDESC           ISOLATE THE DESCRIPTOR
45        SPC
46        BIT     2,(HL)          TEST LOCKEDOUT
47        JP      NZ,NOGO
48        LD      C,H'46'         STOP THE
49        CALL    MOUT2           TIMER
50        SPC
51        SET     0,(HL)          UPDATE
52        RES     1,(HL)          THE
53        RES     3,(HL)          DESCRIPTOR
54        SPC
55        BIT     5,(HL)          TEST EXTENSION
56        JP      Z,NOGO          INITIATED?
57        SPC
58        LD      C,H'43'         START
59        CALL    MOUT2           THE TIMER
60        SPC
```

```
61          LD      C,H'01'         'CON RDY' TO
62          JP      MOUT4           EXTENSION
63          SPC
64          SPC
65          SPC
66 TMOUT    BIT     6,(HL)          TEST IF USER
67          JP      NZ,USTRM        ORIENTATED, IF
68          BIT     5,(HL)          SO JUMP TO USER
69          JP      NZ,USTRM        TIMEOUT ROUTINE
70          SPC
71          CALL    FDESC           IDENTIFY EXT. DESCRIPTOR
72          BIT     0,(HL)          TEST IF EXT. ACTIVE
73          JP      NZ,NOGO
74          SPC
75          SET     2,(HL)          SET LOCKOUT
76          LD      A,(HL)          RESET REST OF
77          AND     H'04'
78          LD      (HL),A          DESCRIPTOR
79          SPC
80          LD      C,H'1F'         LOCKOUT MESSAGE
81          JP      MOUT4           TO EXTENSION
82          SPC
83          SPC
84 USTRM    CALL    FDESU           IDENTIFY USER DESCRIPTOR
85          SET     2,(HL)          SET LOCKOUT
86          INC     HL              FREE THE
87          LD      A,(HL)          TIMESLOT
88          AND     H'1F'           ALLOCATED TO
89          LD      C,A             THIS
90          CALL    DELOC           USER CIRCUIT
91          PUSH    HL
92          SET     7,B             DELOCATE COMMAND
93          CALL    MOUT3           TO INTERFACE MANAGEMENT
94          POP     HL
95          SPC
96          DEC     HL
97          LD      A,(HL)          RESET REST
98          AND     H'01'
99          LD      (HL),A          OF DESCRIPTOR
100         SPC
101         LD      C,H'1F'         USRLKT MESSAGE TO
102         CALL    Q1N             USER (VIA Q1N)
103         SPC
104         LD      C,H'9F'         LKOUT MESSAGE TO
105         JP      MOUT1               CPU
106         SPC
107         SPC
108         SPC
109 STRTMR  CALL    MOUT2           START TIMER MESSAGE
110         JP      PROC
111         SPC
112 CONN    JP      MOUT4           CONNECT MESSAGE
113         SPC
114 STPTMR  CALL    MOUT2           STOP TIMER MESSAGE
115         JP      PROC
116         SPC
117 CONRDY  JP      MOUT4
118         SPC
119 USRLKT  JP      MOUT4
120         SPC
```

```
121 LKOUT  JP      MOUT1
122        SPC
123        SPC
124        SPC
125        EXTRN   BASED,MOUT1,MOUT2,MOUT3,MOUT4,QIN,DELOC,PROC,FDESU
126        ENTRY   FDESC,CONRQ,NOGO,CONTD,TMOUT,STRTMR,CONN,STPTMR,CONRDY,USRLKT,LKOUT
127        SPC
128        SPC
129        END
```

NB In the assembler listing,certain name changes of the messages have been made, e.g. MSSG is equivalent to 'MESSAGE' and an ,!,indicates one of the messages between FEU and CPU. The changes are obvious when considered in combination with the flow charts.

```
 1         TITL    APPLICATION PROGRAMMES(CONTINUED)-AFS13:5
 2 AFS13   CSECT
 3         SPC
 4 CONCMD  CALL    FDESC           IDENTIFY EXT. DESCRIPTOR
 5         SPC
 6         BIT     2,(HL)          TEST LOCKOUT
 7         JP      NEXTT
 8         SPC
 9         LD      C,H'9F'         LOCKEDOUT MESSAGE
10         JP      MOUT1           TO CPU
11         SPC
12 NEXTT  BIT     0,(HL)          TEST IF EXT. ACTIVE
13         JP      Z,STRTA
14         JP      MOUT4           MESSAGE IN STORE TO USER
15         SPC
16 STRTA  CALL    QIN
17         BIT     3,(HL)          TEST ACTIVATING
18         JP      NZ,NOGO
19         BIT     4,(HL)           "   DEACTIVATING
20         JP      NZ,NOGO
21         SPC
22         SET     3,(HL)          SET ACTIVATING
23         RES     1,(HL)          RESET PASSIVE
24         SET     6,(HL)          SET EXCHANGE INITIATED
25         SPC
26         LD      C,H'42'         START
27         CALL    MOUT2       ;   TIMER
28         LD      C,H'40'         CONNECT COMMAND
29         JP      MOUT4           TO HANDLER
30         SPC
31         SPC
32 CENREQ CALL    FDESC           IDENTIFY EXT. DESCRIPTOR
33         SPC
34         BIT     2,(HL)          TEST LOCKOUT
35         JP      Z,STRTB
36         SPC
37         SET     1,(HL)          SET PASSIVE
38         LD      A,(HL)          RESET
39         AND     H'02'           REST OF THE
40         LD      (HL),A          DESCRIPTOR
41         SPC
42         LD      C,H'03'         END CONFIRMATION MESSAGE
43         JP      MOUT4           TO EXTENSION
44         SPC
45 STRTB  SET     4,(HL)          SET DEACTIVATING
46         LD      A,(HL)          RESET REST
47         AND     H'1E'           OF THE
48         LD      (HL),A          DESCRIPTOR
49         SPC
50         LD      C,H'42'
51         CALL    MOUT2           START TIMER
52         SPC
53         LD      C,H'47'         DISCONNECT MESSAGE
54         JP      MOUT4           TO EXTENSION
55         SPC
56         SPC
57 DCOMD  CALL    FDESC           IDENTIFY EXT. DESCRIPTOR
58         SPC
59         BIT     2,(HL)          TEST LOCKOUT
60         JP      Z,STRTC
```

```
 61         SPC
 62         LD      C,H'1F'         IF LOCKOUT THEN
 63         JP      MOUT4           RESET LOOP
 64         SPC
 65 STRTC   LD      C,H'46'         START
 66         CALL    MOUT2           TIMER
 67         SPC
 68         SET     1,(HL)          SET PASSIVE
 69         LD      A,(HL)          RESET REST
 70         AND     H'EF'           OF THE
 71         LD      (HL),A          DESCRIPTOR
 72         SPC
 73         LD      C,H'03'         END CONFIRMATION
 74         JP      MOUT4           TO EXTENSION
 75         SPC
 76         SPC
 77 DCON    JP      MOUT4
 78         SPC
 79 ENDCON  JP      MOUT4
 80         SPC
 81         SPC
 82 MSSG1   LD      C,H'B8'         CORRESPONDING MESSAGE
 83         JP      MOUT1           TO CPU
 84 MSSG2   LD      C,H'B9'                 "
 85         JP      MOUT1
 86 MSSG3   LD      C,H'BA'                 "
 87         JP      MOUT1
 88 MSSG4   LD      C,H'BB'                 "
 89         JP      MOUT1
 90         SPC
 91 MESG!   JP      MOUT1           TANSFER MESSAGE TO CPU
 92 *                               (4 SUCH MESSAGES 1 TO 4)
 93         SPC
 94 MES!AC  LD      C,H'06'         TRANSLATED ACCEPT TO
 95         JP      MOUT4           USER
 96 MES!RJ  LD      C,H'05'
 97         JP      MOUT4                   "
 98         SPC
 99 MSSGAC  JP      MOUT4           TRANSFER TO USER
100 MSSGRJ  JP      MOUT4                   "
101         SPC
102 MESG1   LD      C,H'0C'         TRANSLATED MESSAGE
103         JP      CONDMD          TO USER
104 MESG2   LD      C,H'0D'                 "
105         JP      CONDMD
106 MESG3   LD      C,H'0E'                 "
107         JP      CONDMD
108 MESG4   LD      C,H'0F'                 "
109         JP      CONDMD
110         SPC
111 MSSG!   JP      CONDMD          TRANSFER TO USER
112 *                               (4 SUCH MESSAGES,1 TO 4)
113 MSG!AC  LD      C,H'B4'         TRANSLATED MESSAGE TO
114         JP      MOUT1           CPU
115 MSG!RJ  LD      C,H'B5'                 "
116         JP      MOUT1
117 MESAC   JP      MOUT1           TRANSFER TO CPU
118 MESRJ   JP      MOUT1                   "
119         SPC
120         SPC
121         EXTRN   FDESC,MOUT1,MOUT2,MOUT4,NOGO,QIN
122         ENTRY   CONDMD,CENREQ,DCOND,DCON,ENDCON,MSSG1,MSSG2
123         ENTRY   MSSG4,MESG!,MES!AC,MES!RJ,MSSGRJ,MESG1,MESG2,MESG3
124         ENTRY   MSSG!,MSG!AC,MSG!RJ,MESAC,MESRJ,MSSGAC,MSSG3,MESG4
125         SPC
126         SPC
127         END
```

```
1         TITL      APPLICATION PROGRAMMES(CONTINUED)-AFS8;S
2 AFS8    CSECT
3         SPC
4 *               DESCRIPTOR IDENTITY SUBROUTINE
5 FDESU   LD        HL,BASED        LOAD DESCRIPTOR FILE BASE ADDRESS
6         LD        A,B             LOAD IDENTITY IN A
7         SPC
8         SLA       A
9         RLC       A               MOVE USER IDENTITY
10        RLC       A               INTO REQUIRED
11        SLA       A               POSITION
12        SPC
13        LD        E,A             PUT FIRST BYTE
14        LD        D,H'00'         OF DESCRIPTOR ADDRESS
15        ADD       HL,DE           IN HL
16        SPC
17        RET                       RETURN
18        SPC
19        SPC
20 *              CALL REQUEST PROGRAMME
21        SPC
22 CLLREQ CALL      FDESU           IDENTIFY USER DESCRIPTOR
23        SPC
24        BIT       2,(HL)          TEST LOCKED OUT
25        JP        NZ,NOGO
26        BIT       3,(HL)          TEST ACTIVATING
27        JP        NZ,NOGO
28        SPC
29        SET       3,(HL)          SET ACTIVATING
30        RES       2,(HL)          RESET PASSIVE
31        PUSH      HL
32        CALL      ALLOC           ALLOCATE A TIMESLOT
33        POP       HL
34        SPC
35        BIT       7,C             TEST IF ALLOTED
36        JP        Z,BLDUP
37        SPC
38        LD        C,H'45'         START
39        CALL      MOUT2           TIMER
40        SPC
41        LD        C,H'1E'         CONGESTION MESSAGE
42        JP        MOUT4           TO USER
43        SPC
44 BLDUP  INC       HL              POINTER TO BYTE 2 OF DESC.
45        LD        A,C             TIMESLOT NUMBER TO
46        AND       H'1F'
47        LD        (HL),A          DESCRIPTOR
48        SPC
49        RES       7,B             DELOCATE COMMAND
50        CALL      MOUT3           TO INTERFACE MANAGEMENT
51        SPC
52        LD        A,(HL)
53        BIT       4,A             PUT THE
54        JP        NZ,HORDR        CORRESPONDING
55 *                                CALL REQUEST
56        RES       7,A             MESSAGE (INCLUDING
57        SET       6,A             TIMESLOT INDICATION)
58        SET       5,A
59        SET       4,A             INTO THE ACCU
60        JP        OUTMS
```

```
61          SPC
62 HORDR    SET     7,A
63          RES     6,A
64          RES     5,A
65          RES     4,A
66          SPC
67 OUTMS    LD      C,A             MESSAGE INTO REG  C
68          JP      MOUT1           TRANSFER TO CPU
69          SPC
70          SPC
71 *                PROGRAMME FOR 'INFORMATION SIGNALS
72 *                INF01 TO INF04 ALL USE THIS
73          SPC
74 INFO     CALL    FDESU           IDENTIFY USER DESCRIPTOR
75          SPC
76          BIT     3,(HL)          TEST IF ACTIVATING
77          JP      Z,NOGO
78          SPC
79          JP      MOUT4           TRANSFER MESSAGE TO USER
80          SPC
81          SPC
82 CLLRQ!   CALL    FDESU
83          SPC
84          BIT     3,(HL)
85          JP      Z,NOGO
86          SPC
87          JP      MOUT1       :   TRANSFER MESSAGE TO CPU
88          SPC
89          SPC
90 PROSEL.  CALL    FDESU
91          SPC
92          BIT     3,(HL)
93          JP      Z,NOGO
94          SPC
95          JP      MOUT4
96          SPC
97          SPC
98 INF01!   CALL     FDESU
99          SPC
100         BIT     3,(HL)
101         JP      Z,NOGO
102         SPC
103         LD      C,H'45'         START TIMER
104         CALL    MOUT2
105         SPC
106         LD      C,H'11'         INF01 MESSAGE TO USER
107         JP      MOUT4
108         SPC
109 INF02!  CALL    FDESU
110         SPC
111         BIT     3,(HL)
112         JP      Z,NOGO
113         SPC
114         LD      C,H'45'
115         CALL    MOUT2
116         SPC
117         LD      C,H'12'
118         JP      MOUT4
119         SPC
120 INF03!  CALL    FDESU
```

```
121         SPC
122         BIT     3,(HL)
123         JP      Z,NOGO
124         SPC
125         LD      C,H'45'
126         CALL    MOUT2
127         SPC
128         LD      C,H'13'
129         JP      MOUT4
130         SPC
131 INF04!  CALL    FDESU
132         SPC
133         BIT     3,(HL)
134         JP      Z,NOGO
135         SPC
136         LD      C,H'45'
137         CALL    MOUT2
138         SPC
139         LD      C,H'14'
140         JP      MOUT4
141         SPC
142         SPC
143 PRSEL!  CALL    FDESU
144         SPC
145         BIT     3,(HL)
146         JP      Z,NOGO
147         SPC
148         LD      C,H'45'         START
149         CALL    MOUT2           TIMER
150         SPC
151         LD      C,H'15'         PROSEL MESSAGE
152         JP      MOUT4           TO USER
153         SPC
154         SPC
155 *               PROGRAMME FOR SELECTION CHARACTERS
156 *               ALL 16 USE THIS ROUTINE
157         SPC
158 SELCHR  LD      A,C             MESSAGE TO ACCU
159         LD      C,H'46'         STOP
160         CALL    MOUT2           TIMER
161         SPC
162         SET     7,A             TRANSLATE TO
163         LD      C,A             OUTPUT CHAR.(SELCR')
164         SPC
165         JP      MOUT1           TRANSFER TO CPU
166         SPC
167         SPC
168 SELCR!  JP      MOUT1           TRANSFER TO CPU
169         SPC
170         SPC
171 READY!  CALL    FDESU           IDENTIFY DESCRIPTOR
172         SPC
173         BIT     3,(HL)          TEST ACTIVATING
174         JP      Z,NOGO
175         SPC
176         LD      C,H'45'         START
177         CALL    MOUT2           TIMER
178         SPC
179         LD      C,H'17'         CALL DISABLED MESSAGE
180         JP      MOUT4           TO USER
```

```
181         SPC
182         SPC
183 CLCON!  CALL    FDESU
184         SPC
185         BIT     3,(HL)
186         JP      Z,NOGO
187         SPC
188         SET     0,(HL)          SET USER ACTIVE
189         RES     3,(HL)          RESET USER ACTIVATING
190         SPC
191         LD      C,H'18'         CALL CONNECTED MESSAGE
192         JP      MOUT4           TO USER
193         SPC
194         SPC
195 CLICON   CALL    FDESU
196         SPC
197         BIT     3,(HL)
198         JP      Z,NOGO
199         SPC
200         JP      MOUT4           TRANSFER TO USER
201         SPC
202         SPC
203         EXTRN   BASED,NOGO,ALLOC,MOUT1,MOUT2,MOUT3,MOUT4
204         ENTRY   FDESU,CLIREQ,INFO,CLLRQ!,PROSEL,INFO1!,INFO2!
205         ENTRY   INFO3!,INFO4!,PRSEL!,SELCAR,SELCR!,READY!
206         ENTRY   CLCON!,CLICON
207         SPC
208         END
```

```
1         TITL    APPLICATION PROGRAMMES(CONTINUED):AFS9:S
2  AFS9   CSECT
3         SPC
4  CLRREQ CALL    FDESU           IDENTIFY USER DESCRIPTOR
5         SPC
6         SET     4,(HL)          SET USER DEACTIVATING
7         SPC
8         LD      A,(HL)          RESET THE REST
9         AND     H'10'           OF THE
10        LD      (HL),A          DESCRIPTOR
11        SPC
12        LD      C,H'45'         STOP
13        CALL    MOUT2           TIMER
14        SPC
15        LD      C,H'B0'         CLEAR REQUEST
16        JP      MOUT1           TO      CPU
17        SPC
18        SPC
19 CLRR0! JP      MOUT1           TRANSFER TO CPU
20        SPC
21        SPC
22 CLRCE! CALL    FDESU
23        SPC
24        SET     1,(HL)          SET USER PASSIVE
25        RES     4,(HL)            "    "   DEACTIVATING
26        SPC
27        INC     HL        ;     DEALLOCATE
28        LD      C,(HL)          TIMESLOT
29        PUSH    HL
30        SET     7,B             AND DELOCATE COMMAND
31        CALL    MOUT3           TO INTERFACE MANAGEMENT
32        RES     7,B
33        CALL    DELOC
34        POP     HL
35        SPC
36        LD      C,H'30'         CLEAR CONFIRMATION MESSAGE
37        JP      MOUT4                TO USER
38        SPC
39        SPC
40 CLRCEN JP      MOUT4           TRANSFER MESSAGE TO USER
41        SPC
42        SPC
43 INCLL! CALL    FDESC           IDENTIFY EXTENSION DESCRIPTOR
44        SPC
45        BIT     2,(HL)          TEST IF LOCKED OUT
46        JP      NZ,INITA
47        SPC
48        LD      C,H'9F'         IF SO GIVE LOCKOUT MESSAGE
49        JP      MOUT1           TO CPU
50        SPC
51 INITA  CALL    FDESU           IDENTIFY USER DESCRIPTOR
52        SPC
53        SET     3,(HL)          SET ACTIVATING
54        RES     1,(HL)          RESET PASSIVE
55        PUSH    HL
56        CALL    ALLOC           ALLOCATE A TIMESLOT
57        POP     HL
58        SPC
59        BIT     7,C             TEST IF ALLOCATED
60        JP      Z,INITB
```

```
61          SPC
62          RES     3,(HL)          RESET ACTIVATING
63          SET     1,(HL)          SET PASSIVE
64          LD      C,H'9E'         CONGESTED MESSAGE
65          JP      MOUT1              TO CPU
66          SPC
67 INITB    INC     HL
68          LD      (HL),C          TIMESLOT NR. TO DESCRIPTOR
69          SPC
70          LD      A,C
71          BIT     4,A             DERIVE REQUIRED
72          JP      NZ,UPR          ALLOCATION MESSAGE
73          OR      H'70'           AND TRANSFER THIS TO
74          RES     7,A             THE CPU (VIA QIN)
75          JP      TRANS
76 UPR      AND     H'8F'
77          SET     7,A
78 TRANS    LD      C,A
79          PUSH    HL
80          CALL    QIN
81          POP     HL
82          SPC
83          LD      C,(HL)
84          RES     7,B             ALLOCATION COMMAND
85          CALL    MOUT3           TO INTERFACE MANAGEMENT
86          SPC
87          LD      C,H'18'         INCOMING CALL MESSAGE
88          JP      CONDMD          TO THE USER
89          SPC
90          SPC
91 INCCLL   JP      MOUT4           TRANSFER MESSAGE TO USER
92          SPC
93          SPC
94 ALLOC!   JP      MOUT1           TRANSFER TO CPU
95 *                                31 SUCH MESSAGES
96          SPC
97          SPC
98 CLLREJ   CALL    FDESU
99          SPC
100         INC     HL              ALLOCATED TIMESLOT
101         LD      C,(HL)          NUMBER  INTO
102         DEC     HL              TEMP 2
103         SPC
104         PUSH    HL
105         CALL    DELOC           DEALLOCATE TIMESLOT
106         POP     HL
107         SPC
108         SET     1,(HL)          SET     USER PASSIVE
109         RES     3,(HL)          RESET USER ACTIVATING
110         SPC
111         INC     HL
112         LD      C,(HL)          DELOCATE COMMAND TO
113         SET     7,B             INTERFACE MANAGEMENT
114         CALL    MOUT3
115         SPC
116         LD      C,H'9A'         CALL REJECTED MESSAGE
117         JP      MOUT1               TO CPU
118         SPC
119         SPC
120 CLREJ!  JP      MOUT1           TRANSFER TO CPU
```

```
121         SPC
122         SPC
123 CLLACC  CALL    FDESU
124         SPC
125         SET     0,(HL)          SET USER ACTIVE
126         RES     3,(HL)          RESET USER ACTIVATING
127         SPC
128         LD      C,H'9B'         CALL ACCEPTED TO
129         JP      MOUT1                 CPU
130         SPC
131         SPC
132 CLLAC!  JP      MOUT1           TRANSFER TO CPU
133         SPC
134         SPC
135 CLRND!  CALL    FDESU
136         SPC
137         BIT     1,(HL)          TEST IF USER PASSIVE
138         JP      NZ,NOGO
139         BIT     4,(HL)           "   "   "   DEACTIVATING
140         JP      NZ,NOGO
141         SPC
142         SET     4,(HL)          SET USER DEACTIVATING
143         RES     0,(HL)          RESET USER ACTIVE
144         SPC
145         LD      C,H'45'         START
146         CALL    MOUT2           TIMER
147         SPC
148         LD      C,H'1C'         CLEAR INDICATION MESSAGE
149         JP      MOUT4                TO USER
150         SPC
151         SPC
152 CLRIND  JP      MOUT4           TRANSFER TO USER
153         SPC
154         SPC
155 CLRCON  CALL    FDESU
156         SPC
157         PUSH    HL
158         LD      C,H'45'         STOP
159         CALL    MOUT2           TIMER
160         POP     HL
161         SPC
162         SET     1,(HL)          SET USER PASSIVE
163         RES     4,(HL)          RESET USER DEACTIVATING
164         SPC
165         INC     HL
166         LD      C,(HL)          DEALLOCATE
167         CALL    DELOC           TIMESLOT
168         SET     7,B
169         CALL    MOUT3           DEALLOCATE COMMAND
170 *                               TO INTERFACE MANAGEMENT
171         SPC
172         LD      C,H'9D'         CLEAR CONFIRMATION
173         JP      MOUT1           MESSAGE TO CPU
174         SPC
175         SPC
176 CLRCN!  JP      MOUT1           TRANSFER TO CPU
177         SPC
178         SPC
179         EXTRN   FDESU,FDESC,MOUT1,MOUT2,MOUT3,MOUT4,DELOC
180         EXTRN   ALLOC,QIN,CONDMD,NOGO
181         ENTRY   CLRREQ,CLRRQ!,CLRCE!,CLRCEN,INCLL!,INCCLL
182         ENTRY   ALLOC!,CLLREJ,CLREJ!,CLLACC,CLLAC!,CLRND!
183         ENTRY   CLRIND,CLRCON,CLRCN!
184         SPC
185         END
```

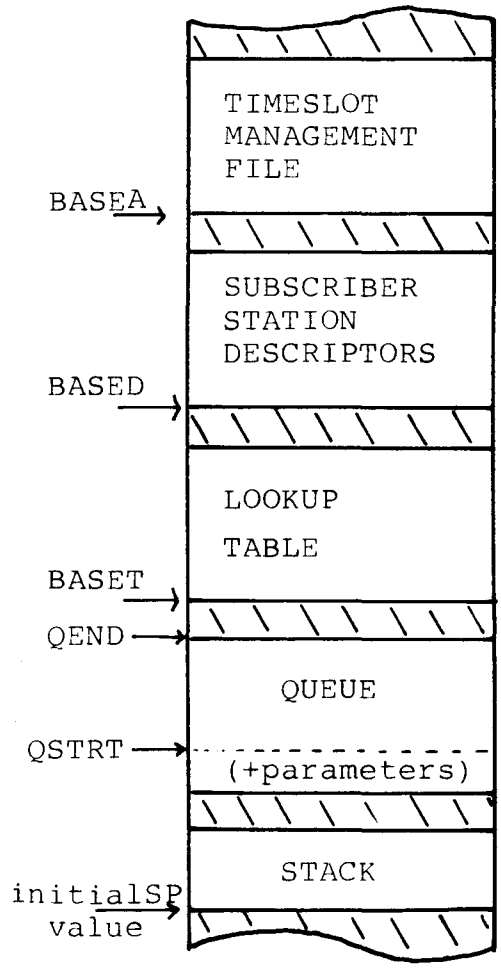
Phase III: Data Transfer

In this phase the actual user to user information transfer takes place. The phase is entered on completion of one of the build up procedures of phase II and ends when one of the users initiates a 'clear request' message.
Messages (or facility change commands) are simply translated into a corresponding message, in the FEU, and transferred to the CPU. This operation is identical with that of Mode 1 of phase II.

This then, completes the consideration of the type of application programmes met in the software. Those covered are merely a sample of all such possibilities - expansion of the total number being possible by simply filling the corresponding entry in the lookup table and adding the required software at the specified point.
The assembler listing contains a realisation of the programmes discussed in the proceeding section.

6.4.5. Assembler Listings

In the following section, an indication is given as to the actual programmes involved in the implementation of the flow charts of the previous sections.
These were passed through an assembler routine in order to give some initial indication as to their validity, pending a full verification using simulation techniques and etc.
The positioning (in RAM) of the various variable data structures, was left open, as is seen from the listings, for specification at a later time. One possibility for this is indicated below.



RAM

The lookup table is presented in RAM to increase flexibility wrt a ROM implementation. The entries of this table are, as discussed previously, the start addresses of the various application programmes (e.g. first entry is that of CONREQ), present in ROM

At startup, or reset, these structures would have to be set to their original values using some initiation procedure.

```
1        TITL        INTERRUPT ROUTINES;AFSB1;S
2 AFSB1  CSECT
3        SPC
4 *              INTERRUPT ROUTINE FOR INTERPROCESSOR MESSAGE HANDLER
5        SPC
6 INT1   DI                          DISABLE INTERRUPTS
7        EX      AF,AF'              SWITCH TO ALTERNATE
8        EXX                         REGISTER BANK
9        SPC
10       LD      HL,H'DFFF'          LOAD HL WITH INT. PROC.
11 *                                 RECEIVER ADDRESS
12       LD      B,(HL)              BYTE 1 TO REG. B
13       LD      C,(HL)              BYTE 2 TO REG. C
14       SPC
15       CALL    QIN                 CALL QUEUE INPUT ROUTINE
16       SPC
17       EXX                         SWITCH TO NORMAL
18       EX      AF,AF'              REGISTER BANK
19       EI                          ENABLE INTERRUPTS
20       SPC
21       RETI                        RETURN FROM INTERRUPT
22       SPC
23       SPC
24       SPC
25 *             INTERRUPT ROUTINE FOR MESSAGE HANDLERS
26 INT2   DI                          DISABLE INTERRUPTS
27       EX      AF,AF'              SWITCH TO ALTERNATE
28       EXX                         REGISTER BANK
29       SPC
30       LD      A,(H'8FFF')         LOAD INT. REG. IN ACCU
31       AND     H'1F'               ISOLATE EXTENSION NR.
32       LD      B,A                 LOAD RESULT IN REG. B
33       SPC
34       LD      A,(H'FFFF')         BYTE 1 TO ACCU
35       AND     H'60'               ISOLATE USER IDENTITY
36       SPC
37       ADD     A,B                 FORM BYTE 1 OF MESSAGE
38       LD      B,A                 RESULT IN REG. B
39       SPC
40       LD      HL,H'FFFF'          SOURCE ADDRESS TO HL
41       LD      C,(HL)              BYTE 2 TO REG. C
42       SPC
43       CALL    QIN                 CALL QUEUE INPUT ROUTINE
44       SPC
45       EXX                         SWITCH TO NORMAL
46       EX      AF,AF'              REGISTER BANK
47       EI                          ENABLE INTERRUPTS
48       SPC
49       RETI
50       SPC
51       SPC
52       SPC
53 *             INTERRUPT ROUTINE FOR REAL TIME FUNCTION
54       SPC
55 INT3   DI
56       EX      AF,AF'
57       EXX
58       SPC
59       LD      HL,H'BFFF'          LOAD HL WITH SOURCE ADDRESS
60       LD      B,(HL)              BYTE 1 TO REG B
```

```
61        LD      C,(HL)          BYTE 2 TO REG. C
62        SPC
63        CALL    QIN
64        SPC
65        EXX
66        EX      AF,AF'
67        EI
68        SPC
69        RETI
70        SPC
71        SPC
72        SPC
73        EXTRN   QIN
74        ENTRY   INT1,INT2,INT3
75        SPC
76        SPC
77        END
```

```
 1        TITL   QUEUE ROUTINES   AFS10:S
 2 AFS10  CSECT
 3        SPC
 4 *                    QUEUE INPUT ROUTINE
 5        SPC
 6 QIN    LD     D,QMAX          LOAD REG D WITH MAX. QUEUE LENGTH
 7        LD     A,(QCNT)        LOAD ACCU WITH PRESENT COUNT
 8        SUB    D               CHECK IF QUEUE IS FULL
 9        JP     Z,QFULL         IF FULL GO TO FULL QUEUE ROUTINE
10        SPC
11        LD     HL,(NXTIN)      LOAD HL WITH POINTER TO NEXT INPUT
12        LD     (HL),B          BYTE 1 TO QUEUE
13        INC    HL              INCREMENT QUEUE POINTER
14        LD     (HL),C          BYTE 2 TO QUEUE
15        SPC
16        SPC
17        LD     DE,QEND         LOAD REG DE WITH POINTER TO TAIL
18        LD     A,D             HIGH ORDER BYTE TO ACCU
19        CP     H               COMPARE WITH HIGH ORDER TAIL
20        JP     NZ,QINC         JUMP TO INCREMENTING ROUTINE
21 *                                     IF DIFFERENT
22        SPC
23        LD     A,E             LOW ORDER BYTE TO ACCU
24        CP     L               COMPARE WITH LOW ORDER TAIL
25        JP     NZ,QINC         JUMP IF DIFFERENT
26        SPC
27        SPC
28        LD     HL,QSTRT        IF END OF QUEUE THEN SET
29        JP     PARAM           POINTER TO START
30        SPC
31        SPC
32 QINC   INC    HL              INCREMENT QUEUE POINTER
33        SPC
34 PARAM  LD     (NXTIN),HL      UPDATE QUEUE POINTER
35        SPC
36        LD     A,(QCNT)        INCREMENT QUEUE LENGTH
37        INC    A               VARIABLE
38        LD     (QCNT),A
39        SPC
40        RET
41        SPC
42        SPC
43        SPC
44        SPC
45 *                    QUEUE OUTPUT ROUTINE
46        SPC
47        SPC
48        SPC
49 QOUT   LD     A,(QCNT)        CHECK IF
50        JP     Z,QMPTY         QUEUE IS EMPTY
51        SPC
52        LD     HL,(NXTOUT)     ENTER NEXT
53        LD     B,(HL)          OUTPUT MESSAGE
54        INC    HL              INTO REGISTER
55        LD     C,(HL)          PAIR BC
56        SPC
57        LD     DE,QEND         CHECK IF
58        LD     A,D             END OF QUEUE
59        CP     H               AREA HAS BEEN REACHED
60        JP     NZ,QINCO         IF SO JUMP TO RESET
```

```
61 *                                        ROUTINE
62         LD      A,E             SAME FOR LOW
63         CP      L               ORDER POINTER
64         JP      NZ,QINCO
65         LD      HL,QSTRT        IF SO THEN SET HL TO
66         JP      PARIM           START OF QUEUE
67         SPC
68         SPC
69 QINCO   INC     HL              INCREMENT POINTER
70 PARIM   LD      (NXTOUT),HL     UPDATE POINTER TO
71 *                               NEXT OUTPUT MESSAGE
72         SPC
73         LD      A,(QCNT)        DECREMENT THE
74         DEC     A               QUEUE COUNT
75         LD      (QCNT),A
76         RET                     END OF ROUTINE
77         SPC
78 QMPTY   LD      A,H'0'          SET STATUS OF
79         LD      (QSTATE),A      QUEUE TO EMPTY
80         RET                     AND RETURN
81         SPC
82 QFULL   LD      HL,OVERF        LOAD HL WITH OVERFLOW ADDRESS
83         INC     (HL)            INCREMENT OVERFLOW COUNT
84         RET                      AND RETURN
85         SPC
86         EXTRN   QMAX,QCNT,NXTIN,NXTOUT,QEND,QSTRT,QSTATE,OVERF
87         ENTRY   QIN,QOUT    ;
88         SPC
89         END
```

```
1        TITL    MESSAGE PROCESSING ROUTINE-AFSB3;S
2 AFS3B  CSECT
3        SPC
4 *              MESSAGE PROCESSING ROUTINES
5        SPC
6 PROC   LD      A,H'01'          SET LOCATION QSTATE
7        LD      (QSTATE),A       TO 1
8        SPC
9        DI                       DISABLE INTERRUPTS
10       CALL    QOUT             GET NEXT MESSAGE FROM QUEUE
11       EI                       ENABLE INTERRUPTS
12       LD      A,(QSTATE)       CHECK THE RESULT
13       JP      Z,PROC           IF EMPTY TRY AGAIN
14       SPC
15 CONV  LD      HL,BASET         LOAD HL WITH BASE
16 *                              ADDRESSOF LOOKUP TABLE
17       LD      D,H'00'          SET REG. D TO ZERO
18       LD      A,H'00'          SET ACCU TO ZERO
19       LD      E,B              TRANSFER BYTE 1 TO REG E
20       SLA     E                SHIFT LEFT
21       ADC     A,D
22       LD      D,A              CALCULATE DISPLACEMENT
23       ADD     HL,DE            INDEX TABLE
24       SPC
25       LD      D,(HL)           GET START
26       INC     HL               ADDRESS OF
27       LD      E,(HL)           APPLICATION
28       EX      DE,HL            PROGRAMME IN HL
29       SPC
30 CALAP JP      (HL)             JUMP TO APPLICATION
31 *                              PROGRAMME
32       SPC
33       SPC
34       EXTRN   QSTATE,QOUT,BASET
35       ENTRY   PROC
36       SPC
37       END
38       SPC
```

```
1         TITL    MESSAGE OUTPUT ROUTINES AFS0;S
2  AFS0   CSECT
3         SPC
4  *              MESSAGE OUTPUT ROUTINE 1;INTERPROCESSOR HANDLER
5         SPC
6  MOUT1  LD      A,(H'CFDF')     LOAD STATUS INTER.PROCESSOR HANDLER
7         SLA     A               CHECK IF BUSY
8         JP      C,QRET          IF SO RETURN TO QUEUE
9         SPC
10 TRANS1 LD      HL,H'CFFF'      LOAD ADDRESS PERIPHERAL
11        LD      (HL),B          BYTE 1 TO PERIPHERAL
12        LD      (HL),C          BYTE 2 TO PERIPHERAL
13        JP      PROC            TRANSFER COMPLETE
14        SPC
15 QRET   DI                      DISABLE INTERRUPTS
16        CALL    QIN             CALL QIN ROUTINE
17        EI                      ENABLE INTERRUPTS
18        JP      PROC            TRANSFER BACK INTO
19 *                              QUEUE COMPLETED
20        SPC
21 *              MESSAGE OUTPUT ROUTINE 2;REAL TIME FUNCTION
22        SPC
23 MOUT2  LD      A,(H'ADFF')     LOAD STATUS RL.TIME REGISTER
24        SLA     A               CHECK IF BUSY
25        JP      C,QROT          IF SO PUT MESSAGE IN QUEUE
26        SPC
27 TRANS2 LD      HL,H'AFFF'      LOAD PER. ADDRESS
28        LD      (HL),B          BYTE 1 TO PERIPHERAL
29        LD      (HL),C          BYTE 2 TO PERIPHERAL
30        RET
31 QROT   DI                      DISABLE INTERRUPTS
32        CALL    QIN
33        EI                      ENABLE INTERRUPTS
34        RET                     RETURN TO CALLING PROCESS
35        SPC
36        SPC
37 *              MESSAGE OUTPUT ROUTINE 3;INTERFACE MANAGEMENT
38        SPC
39 MOUT3  LD      A,(H'9FDF')     LOAD STATUS INT. MAN. REGISTER
40        SLA     A               CHECK IF BUSY
41        JP      C,MOUT3         IF SO TRY AGAIN
42        SPC
43 TRANS3 LD      HL,H'9FFF'      LOAD INT MAN. RECEIVER ADDRESS
44        LD      (HL),B          BYTE 1 TO PER.
45        LD      (HL),C          BYTE 2 TO PER.
46        RET                     TRANSFER COMPLETE,RETURN
47        SPC
48        SPC
49 *              MESSAGE OUTPUT ROUTINE 4;SUBS. LOOP HANDLER
50        SPC
51 MOUT4  LD      L,B             BYTE 1 TO REG. L
52        LD      A,B
53        AND     H'1F'
54        LD      L,A
55 *                              STATUS ADDRESS
56        LD      H,H'EF'         LOAD H WITH GLOBAL
57        BIT     7,(HL)          TEST IF BUSY
58        JP      NZ,QRET         IF SO RETURN TO QUEUE
59        SPC
60        SET     5,L
```

7. SYSTEM CONFIGURATION

The Front End Unit consists of a number of printed circuit cards, on which the hardware described in the previous chapters is mounted, plus a collection of interconnection lines.
These connection lines occur in three groups:

1) Subscriber lines, which connect the subscriber stations with the LTU's in the front end unit. Thirty two subscriber stations are connected to the FEU.
2) Information Highway, connecting the front end unit with the main body of the exchange.
3) Internal connections between the cards forming the front end unit itself.

The optimisation of the number of these internal connections, with respect to decoding hardware and etc., has been considered in combination with the implementations of the various system functions.

The resulting system configuration is given in fig. 7.1. This relies to a great extent on the use of customised integrated circuit design to reduce the volume of the hardware concerned. Should this prove impractical an increase in the number of printed circuit cards is easily feasible.

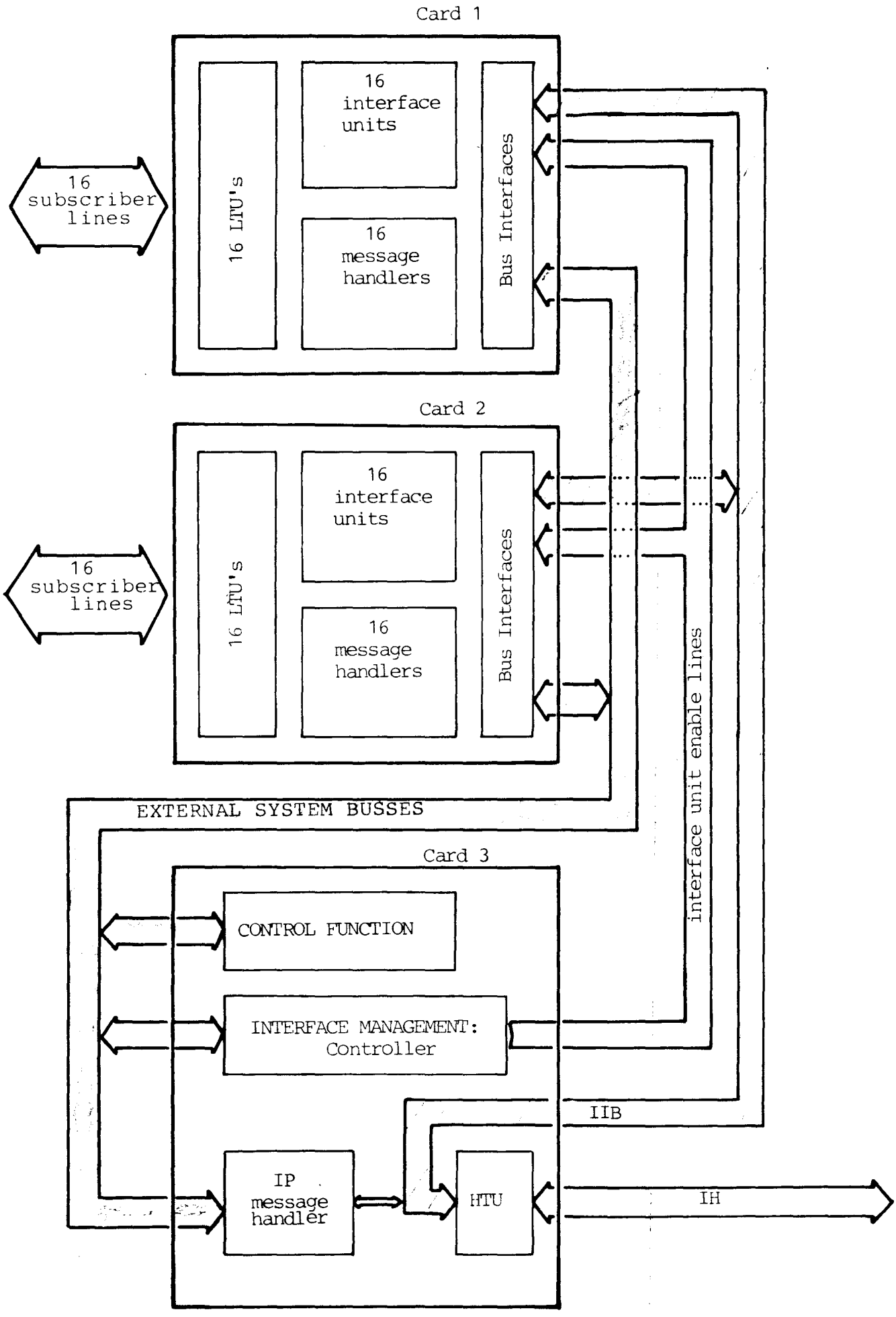


Fig.7.1 GEOGRAPHICAL SYSTEM CONFIGURATION.

References

1a. Proc. of the International Zurich Seminar on Digital Communications. Zurich, March 1978.

1b. Proc. of the International Symposium on Subscriber Loops and Services. Atlanta, March 1978.

2. Studies of Digital Local Networks. CCITT Study Group VII; contribution COM.VII - No. 240-E.

3. Z80 - CPU Technical Manual. Zilog (UK) Ltd., 1976.

4. Wachttijd en Stagnatie Problemen. Collegedictaat nr. 5.564, TH Eindhoven.

5. Sorting and Searching: D. KNUTH. Vol. 3 of series 'The Art of Computer Programming', 1968, Addison Wesley.

6. Loss probability charts calculated with the formula of Engset. M.M. JUNG; Philips Telecom. Review, Vol. 23, No. 4, October 1962.

7. MCS-48™ Microcomputer User's Manual. Intel Corp.,1977.

Abbreviations

AUC - Additional User Circuit

CRC - Cyclic Redundancy Check

CPU - Central Processing Unit (in the main body of the exchange)

FEU - Front End Interface Unit

GC - Global Clock

HTU - Highway Termination Unit

IDN - Integrated Digital Network

IH - Information Highway

IIB - Internal Information Bus

IP - Inter Processor

ISDN - Integrated Services Digital Network

LTU - Line Termination Unit

MUC - Main User Circuit

PCM - Pulse Code Modulation

SR - System Reset

APPENDIX A

TRAFFIC CONSIDERATIONS

In order to obtain an indication as to the quality of service supplied by the system, an approximation must be made as to the probability of congestion occuring ie the chance that a connection request will be rejected due to lack of output circuits.

The system can be depicted as in fig. A1. Here the system (FEU) supplies connection facilities between c sources and n output circuits ($n \leq c$). In the case being considered these output circuits are created on a serial highway structure (IH). Each output circuit consists of one time slot per frame on this highway. The sources are the user circuits of the subscriber station.
It will be assumed that the traffic generated by a user is 0,25 Erlangs in the busy period.

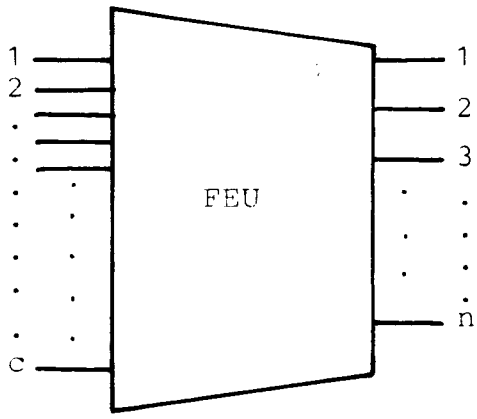


Fig.A.1

The probability of congestion in such a system can be described using an Engset distribution (see reference) i.e.

$$WE = \frac{\binom{c-1}{n}\left(\frac{\alpha}{1-\alpha(1-W)}\right)^n}{\sum_{i=0}^{n}\binom{c-1}{i}\left(\frac{\alpha}{1-\alpha(1-W)}\right)^i} = EW_\infty$$

Where n = number of full availability servers ie output circuits
c = number of sources
a = total traffic from all sources
α = a/c = traffic offered/source
WE = prob. of loss to which calls from a source are subjected ie congestion
W_∞ = probability of loss for $c \longrightarrow \infty$, $\alpha \rightarrow 0$ (with a remaining finite) ie Erlangs loss formula

In total 64 users (two per subscriber station) are to be served, giving a total traffic of 64 x 0,25 = 16 Erlangs.
Of the 32 output circuits on the IH one is dedicated to inter processor signalling, leaving 31 for random allocation to a server, ie n=31.
The loading of these output circuits is then $\pm$ 0,5.

Interpolation of the graphs in reference , as shown in fig. A.2., to an E value of $\pm$ 0,1 and $W_\infty \leq 0{,}00025$, which gives WE=$\pm$ 0,000025.

This low probability of congestion can be neglected ie effectively a non-blocking network is obtained.

n	$E = \frac{WE}{W_\infty}$
1	± 0,98
5	0,93
8	0,83
10	0,78
12	0,72
15	0,60
18	0,47
20	0,37
22	± 0,29
24	± 0,22
26	± 0,17
28	± 0,13

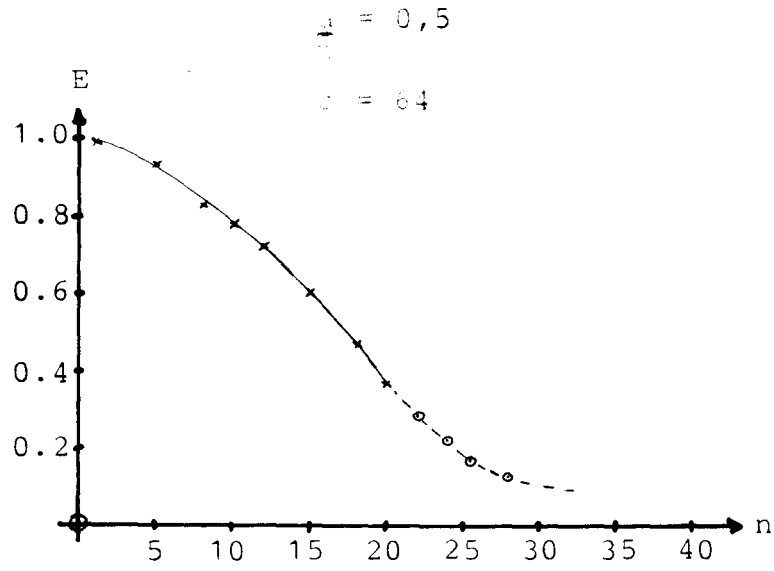


Fig.A.2

(n ≧ 22 from interpolated values)

APPENDIX B

DATA FACILITIES

The object of this appendix is to consider, briefly, the facilities necessary, in the exchange, to support data transfer between two user terminals.
In the main report, two user circuits were considered :

- Main User Circuit (MUC), of capacity 64 kb/sec, used for digitised speech transfer.
- Additional User Circuit (AUC), of capacity 8 kb/sec, used for data transfer.

Although this is likely to be the most common configuration for the multi-function set, there is a little reason to exclude the MUC from use as a data carrying circuit.
The case will now be considered in which two terminals are to be connected having, as access path to the exchange, any one of the above two circuits.
In order to remain brief, a number of assumptions will be made :

- the two terminals being considered are of exactly the same form and operate at a data rate of x kb/sec.
- x will have a value within the capacity of both circuits, namely $x \leq 8$ kb/sec.
- a transparent connection is to be created, ie no real processing of the contents of the data streams is to take place.

In order to transfer the data from a terminal to the exchange, some bit rate conversion must first take place (and vice versa wrt received signal), as shown in fig. B.1. Secondly, as the exchange switches exclusively 64 kb/sec bit streams, a second conversion must take place on the AUC within the exchange. This is also shown in fig. B.1.

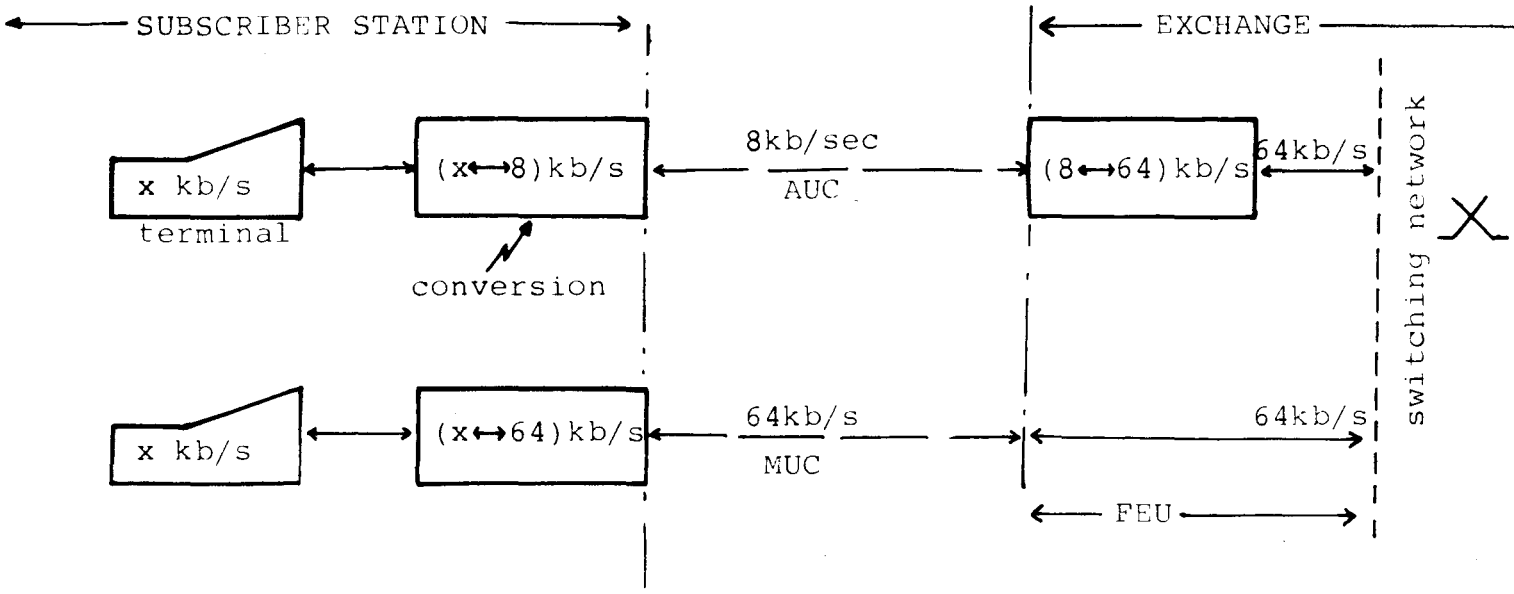
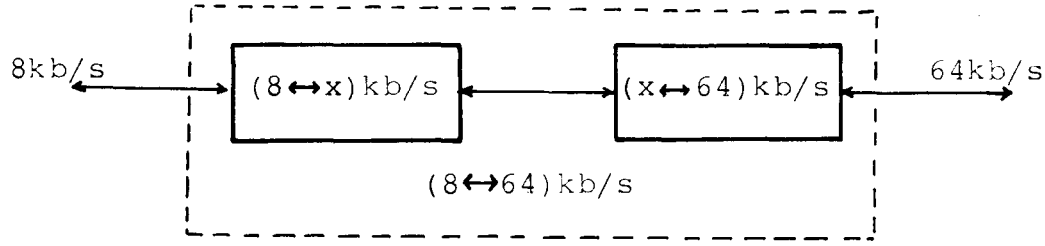


Fig. B.1

In order to create an effective connection between the two terminals, the overall conversion law - terminal to switching network, must be identical in both cases.
This can be accomplished by considering the (8 ↔ 64)kb/sec conversion as the summation of two more basic conversions, as shown below.



Compatibility is thus achieved by the insertion, within the FEU, of this complex conversion unit in each AUC.
This solution has two main disadvantages :

1) It is uneconomical, as each AUC must have this complex, and therefore relatively expensive, unit inserted in the FEU.
2) Flexibility. As no standard conversion can be defined at present, a system configuration should be chosen which can be easily adapted to such a definition at a future time. Using the above solution this this would entail the alteration, or replacement, of every conversion unit.

These problems can be solved, to a great extent, by simplifying the (8↔64)kb/sec conversion in the FEU.

Here, a simple bit multiplication technique is used to create the required 64 kb/sec signal to the switching network, as shown in fig. B.2. The two bit streams, A and B, are now incompatible, as they were not formed via identical conversion laws.

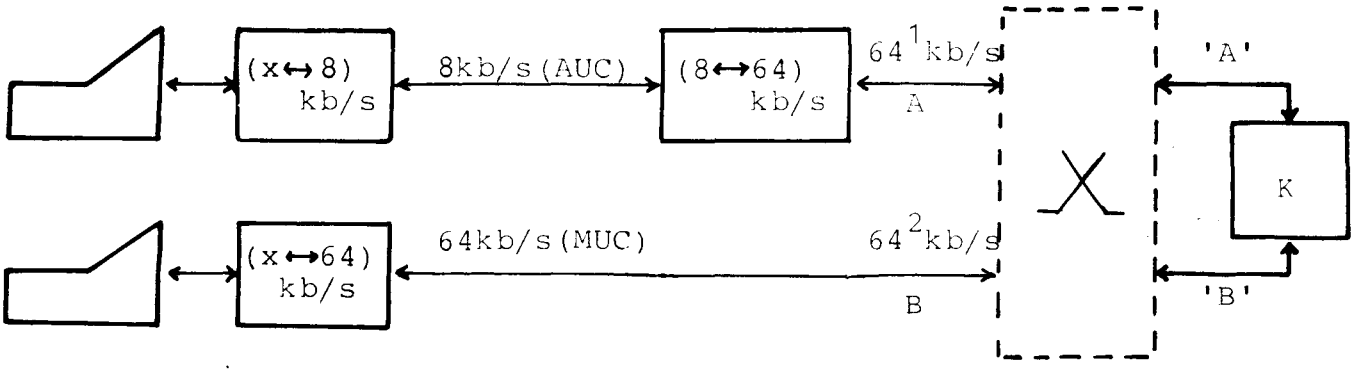
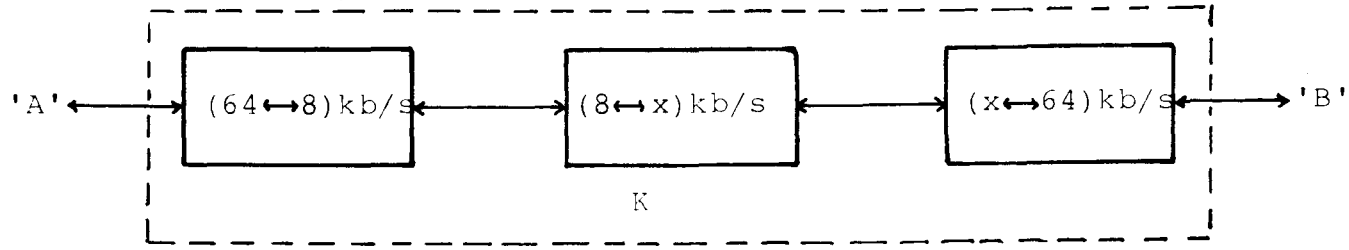


Fig. B.2

Compatability between the two can be retrieved by connecting the AUC, via the switching network, with a special conversion unit, K, as in Fig. B.2. This converts the "A" stream into a form compatible with 'B', as shown below.



This solution has the advantage that a relatively small number of these 'K' units can be used to serve all the AUC's connected to the exchange, ie on a concentrated basis. A significant concentration factor can be used as :

- the expectation is that most data connections will occur purely via AUC's. For a connection between two of these, no 'K' unit need be included (ie two 'A' streams can be connected directly)
- similary a data connection via two MUC's also requires no 'K' unit
- not all AUC's would be active simultaneously (an AUC will be active for less than 1/4 of the time, on average).

A further advantage is that as these 'K' units can now be geographically concentrated in the exchange, the ease with which they can be altered, or replaced, is improved (wrt future standardisation). The FEU conversion is unlikely to require any alteration in this configuration.

Disadvantages of this solution are that :
- an extra conversion rule has been added (albeit an extremely simple one)
- the exchange must now have knowledge of the equipments connected to it. An AUC/MUC connection request, for data transfer, must be identified and the 'K' unit inserted.

As the advantages outweigh the disadvantages (at least in the foreseeable future), this latter solution is considered more suitable and implemented, as described in chapter 5.

Connections in which one of the terminals is external to the exchange area considered in this report, can be split into two groups:
1) where an ISDN is fully operational
2) where one or more transmission sections based on the conventional 4kHz analogue bandwith, can occur.

In the former case no problem arises. Connections are made with the ISDN via the 'B' stream- this having been created by the standard (x ⟷ 64) kb/sec conversion law.
In the latter case, which is likely to occur during the transition period between the conventional network and the ISDN, a more complex situation occurs. This leads to the requirement that further conversions would have to be used in order to create a totally compatible system. Due to the extent of such a subject, it is considered outwith the scope of this project. This could be considered as an area for further research, along with the detailed definition of the conversion laws themselves and consideration of certain 'value added' features, such as the connection of two terminals with differing data rates.

APPENDIX C

REAL TIME FUNCTION

As was explained in chapter 6, this function consists of the collection of timing processes (timers) necessary to support system operation.
These timers are used to create a limit for the reaction times to events generated from the application programmes. A timing process is initiated by the message, 'start timer', from the control function, and ends either due to a 'stop timer' message, or a time out condition when the time limit has been exceeded. This latter leads to an alarm message being returned to the control function.

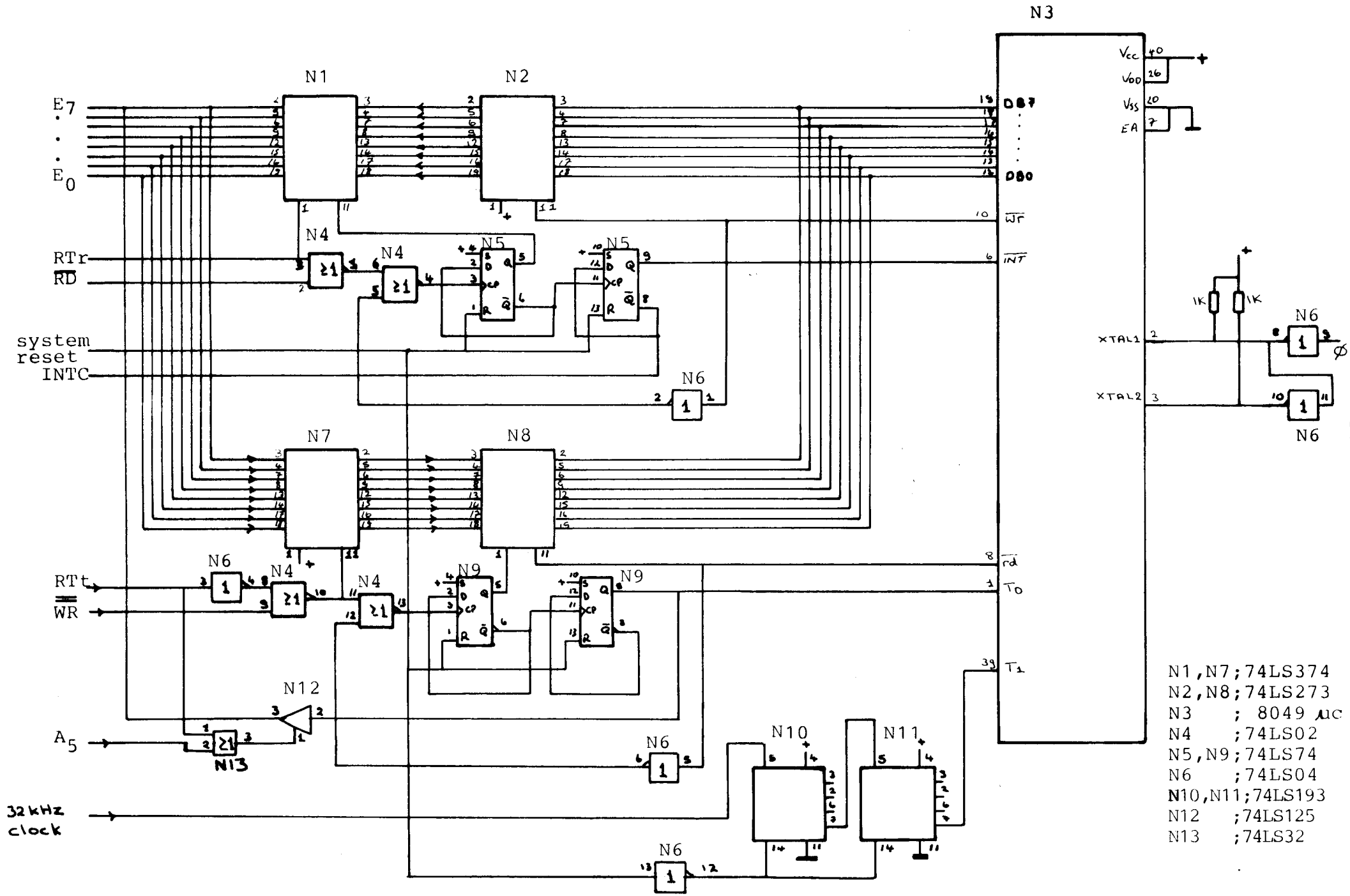
It would be possible to include a physical timing unit for each of these processes i.e. a hardware counter.
Due to the relatively large number of such processes, this is an uneconomical solution. A significant improvement is made by creating the timing processes in software, and thus limiting the hardware used to a basic minimum.

C.1. Hardware

The basic hardware configuration is given in fig. 6.8.
This consists of:

- an I/O buffer to implement message transfers to and from the control function.
- a 8049 microcomputer to support the actual timing processes.
- a hardware counter, clocked at 32 kHz, used to drive the internal counter of the 8049 and thus produce a clock, of period 1 sec, within the 8049.

A detailed circuit diagram is given in Fig.C.1.



N1,N7;74LS374
N2,N8;74LS273
N3 ; 8049 μc
N4 ;74LS02
N5,N9;74LS74
N6 ;74LS04
N10,N11;74LS193
N12 ;74LS125
N13 ;74LS32

Fig. C.1.

C.2. System Operation

The internal/external counter combination is used to initiate an interrupt every second.
This interrupt causes:

a) a software counter, operating modulo 240, to be incremented.
b) a test to be made of all the timing processes.

A timing process is identified by a two byte descriptor. This indicates:

- the identity of the user with which the timer is concerned.
- the 'time' (the value of the software counter) at which an alarm message is to be generated.

Testing of the timing processes, (b), thus consists of the comparison of the present counter value with the alarm value given in the descriptor. This 'alarm time' is obtained from the message which initiated the process. This message includes an 'alarm class', i.e. an indication as to the period which must elapse before an alarm is given. On initiation of a timing process this period is added to the present counter value (modulo 240), the result being the required 'alarm time'.
For simplicity, four such classes will be used, as shown in table 1. (This can easily be expanded to cover a greater range or, using a shorter counter period, to increase precision.)

alarm class	code	alarm period (sec)
0	00	60
1	01	120
2	10	180
3	11	240

Table 1

Should the test show that the 'alarm time' and present counter value are identical, an alarm message is generated (and the timing process deleted).

These timing process descriptors will be implemented in a fixed memory area,i.e. a number of descriptor areas (31) will be created, whether in use or not.
Active descriptors will be indicated by a status bit being set to 1. A descriptor now takes the form shown in fig. C.2.a., while the collection of descriptors and connected variables is indicated as shown in fig. C.2.b.

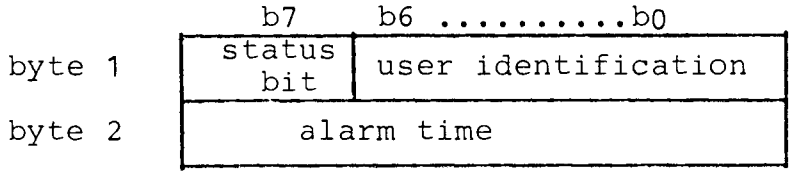


Fig. C.2.a. Timing Process Descriptor

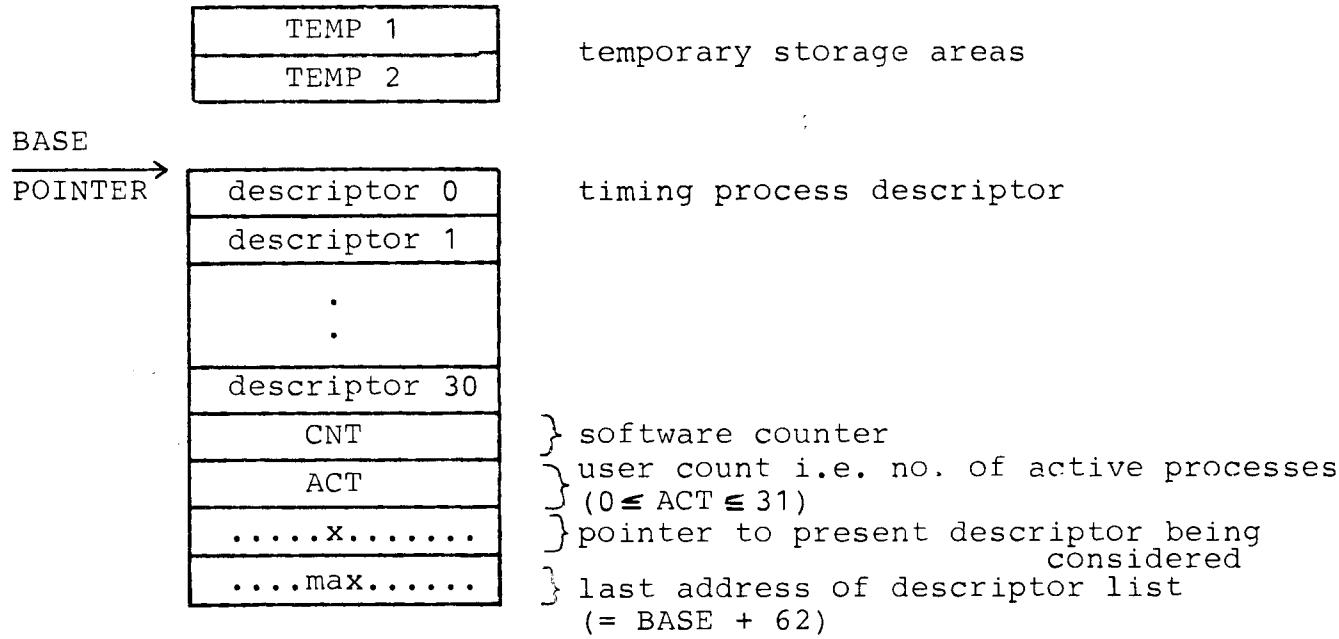


Fig. C.2.b.

C.3. Software

This consists of three programmes which must support the system operation.
These are:

a) Main Programme

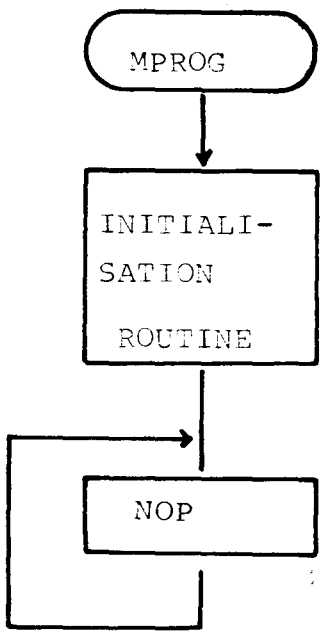
b) An interrupt service routine (int) for the handling of internal interrupts. These are generated every second by the internal/external counter combination.

c) An interrupt service routine (INT) for the handling of external interrupts. These are generated by the I/O buffer and are connected with message transfer from the control function.

In the following sections, a description of these routines will be made using flow charts to show their operation.

a) Main Programme :

The operation of this is shown in fig. C.3.



Main Programme

Set varables to their initial value values:
all status bits = 0
CNT = 0
ACT = 0
X = BASE
Start timer counter-load initial value. Enable interrupts, and etc.

No operation

Fig. C.3

As is seen, after setting the variables to the required initial values, the system enters a waiting state, consisting of the continuous execution of the NOP instruction.
An interrupt occuring breaks this cycle and, as explained in ref...., causes the following two actions :

- the disabling of further interrupts
- a 'call' being made to memory location 3 or 7 for an external respectively internal interrupt. These locations would have as contents, an unconditional jump instruction to the corresponding service routine INT resp. int.

At the end of the particular service routine a return is made to the main programme and the interrupt inputs enabled.

b) Internal Interrupts

This is described using the flow charts of fig. C.4.a and b. As this is seen from these figures, the routine consists of the testing of the 'alarm time' in each active descriptor, and the transfer, to the control function, of an alarm message (Time Out) for any of these in which the alarm time has been reached.

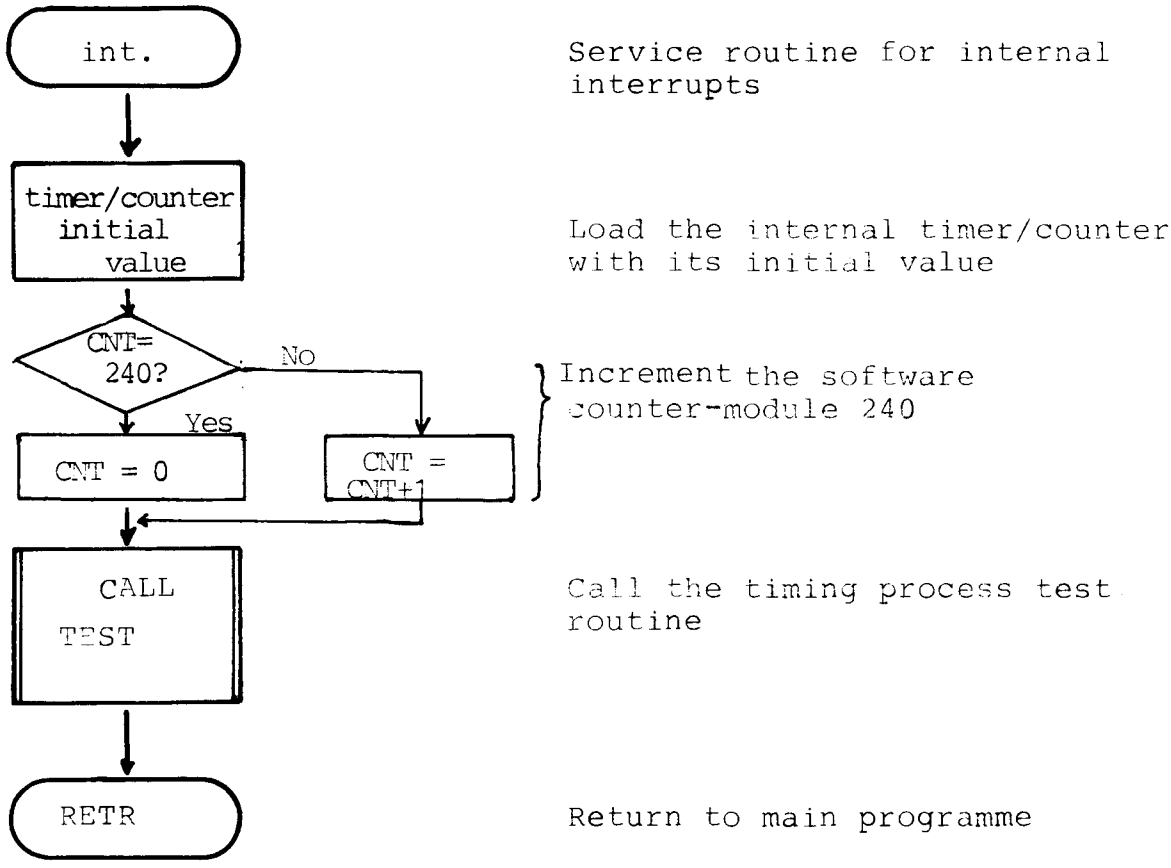
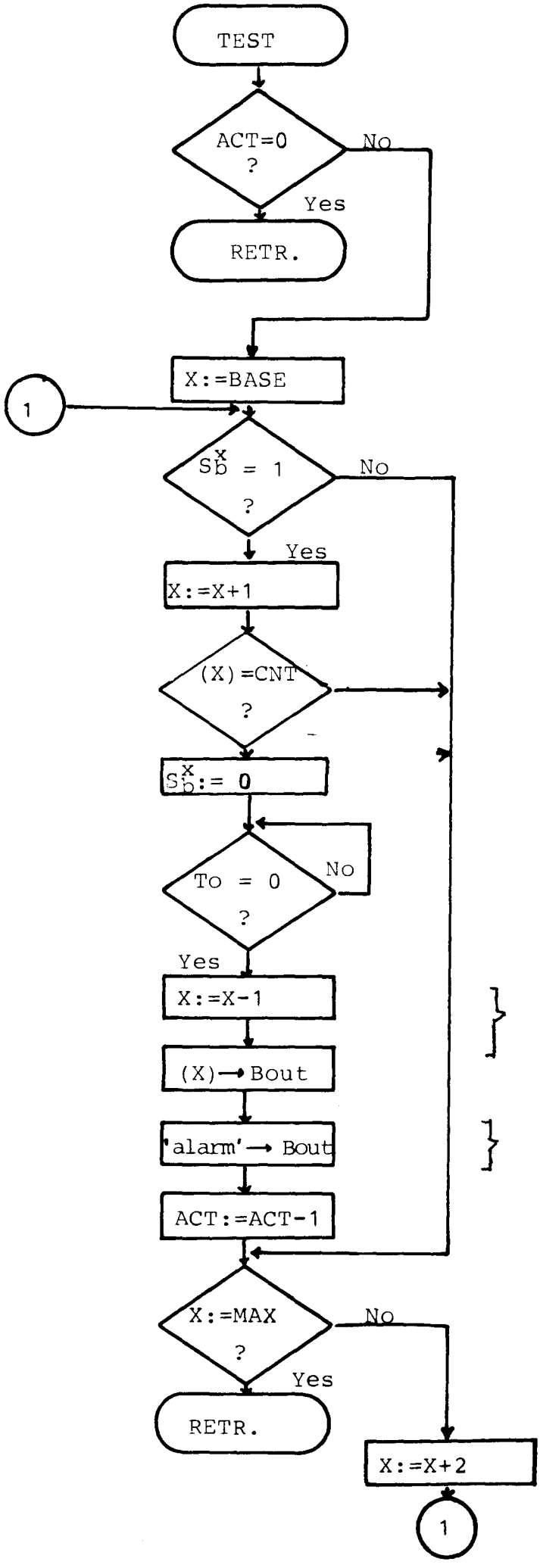


Fig. C.4.a



Timing process test routine

Test whether any timing processes are active

if not then return to the calling process

otherwise begin testing the process descriptors

Set x pointer to first descriptor

Test status bit

If active set x pointer to second byte of descriptor

Test if 'alarm time' of descriptor is the same as the present 'time' (ie software counter value)

If so deactivate process

Wait until I/O buffer is free

Transfer first byte of output message to I/O buffer (ie user identity)

Transfer second byte of message to I/O buffer

Decrement user count

Test if all active descriptors have been checked

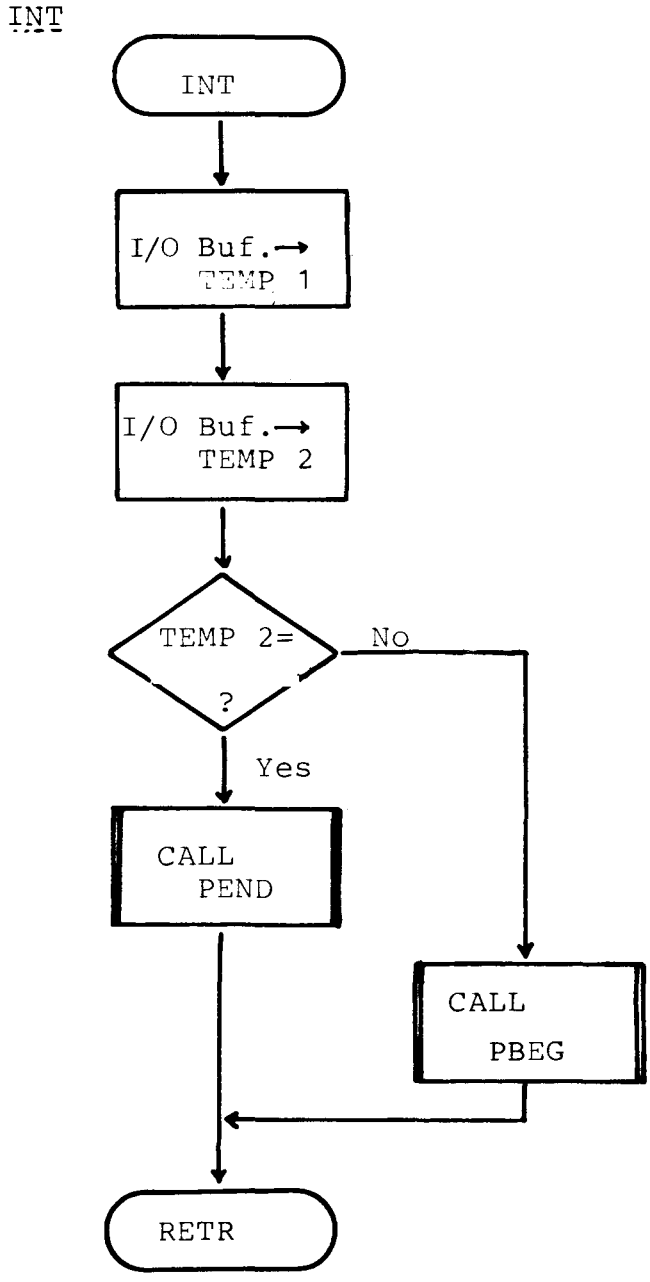
If so return to calling process

Otherwise continue with next descriptor

Fig. C.4.b

c) External Interrupts

This routine must first cause the incoming message to be transferred from the I/O buffer into a temporary storage area (TEMP 1 and 2). This message must now be checked to ascertain whether a new timing process is to be created or a present process ended. The requirement is indicated in the second byte of the message -a 'start timing process' message having the timing class parameter included in this second byte. hort descriptions of the operation of this service routine are given, in combination with the flow charts of the various elements, in figs. C.5.a.b and c .



External Interrupt Routine

1st byte of input message (in I/O buffer) to temporary storage area TEMP 1

2nd byte of message to TEMP 2

Test if message is a 'stop timer' command

if so call sub-routine for ending of timing process, PEND

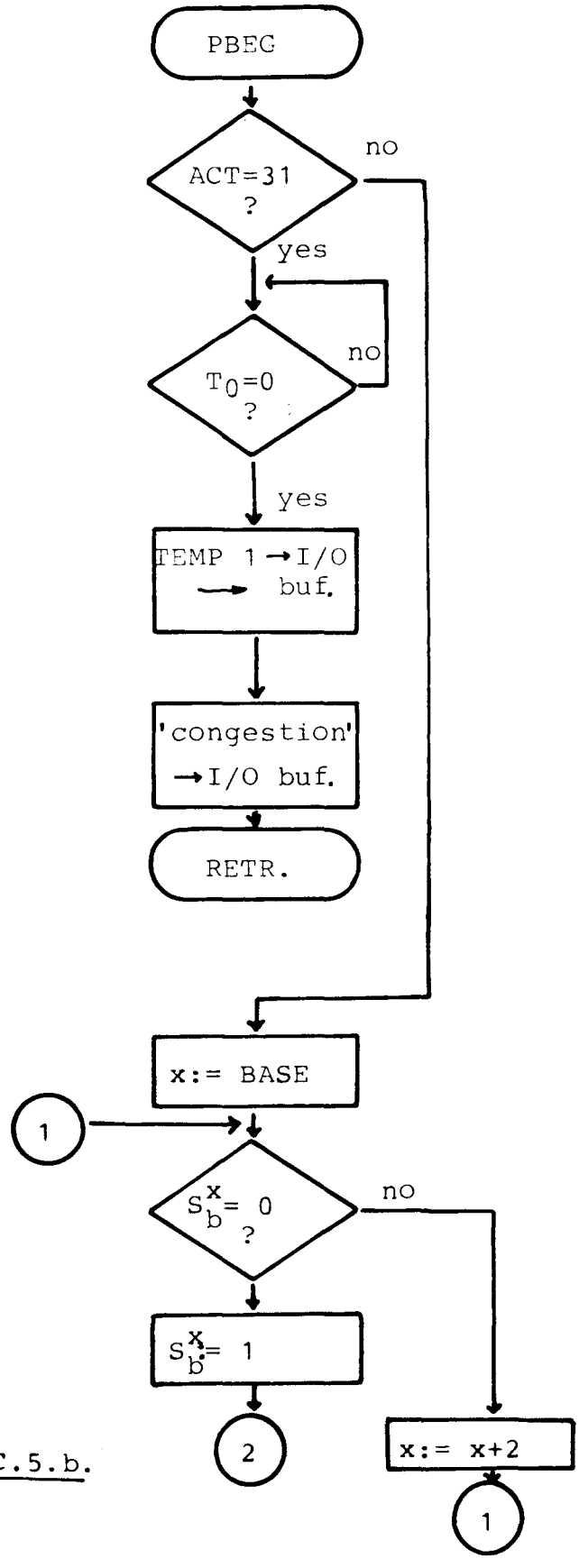
Otherwise call sub-routine for initiating timing process, PBEG

End of service routine, return to Main Programme

Fig. C.5.a

PBEG

This subroutine, described in the flow chart of fig. C.5.b. causes a new timing process to be initiated. A search is made for a free descriptor and, when found, the user identification and 'alarm time' are entered. The timing process is then activated by setting the descriptor status bit to '1'.



subroutine for the generation of a new timing process

test if all process descriptors are in use i.e. user count = 31

if so wait until I/O buffer if free (T_0 input is zero)

transfer 1st byte of output message to I/O buffer

transfer 'congestion' message to I/O buffer

return to calling process

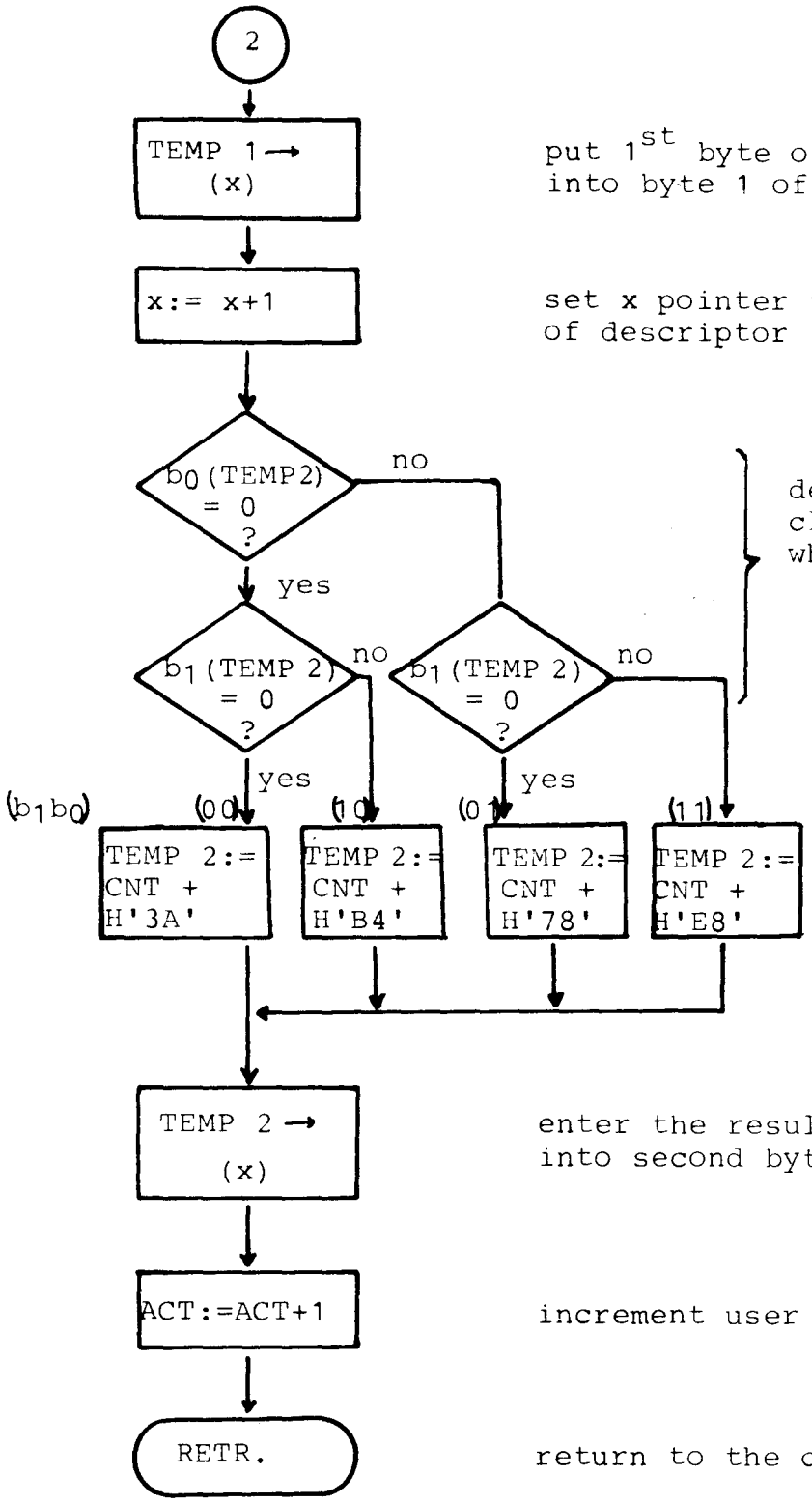
otherwise set x pointer to first descriptor

test status bit of this descriptor

if free then activate the descriptor

otherwise continue with next descriptor

Fig. C.5.b.



put 1st byte of received message into byte 1 of descriptor

set x pointer to second byte of descriptor

determine the alarm class of the process which is to be created

add the corresponding tim to the present value of the software counter and enter the result in TEMP 2. This addition occurs modulo 240

enter the resulting 'alarm time' into second byte of descriptor

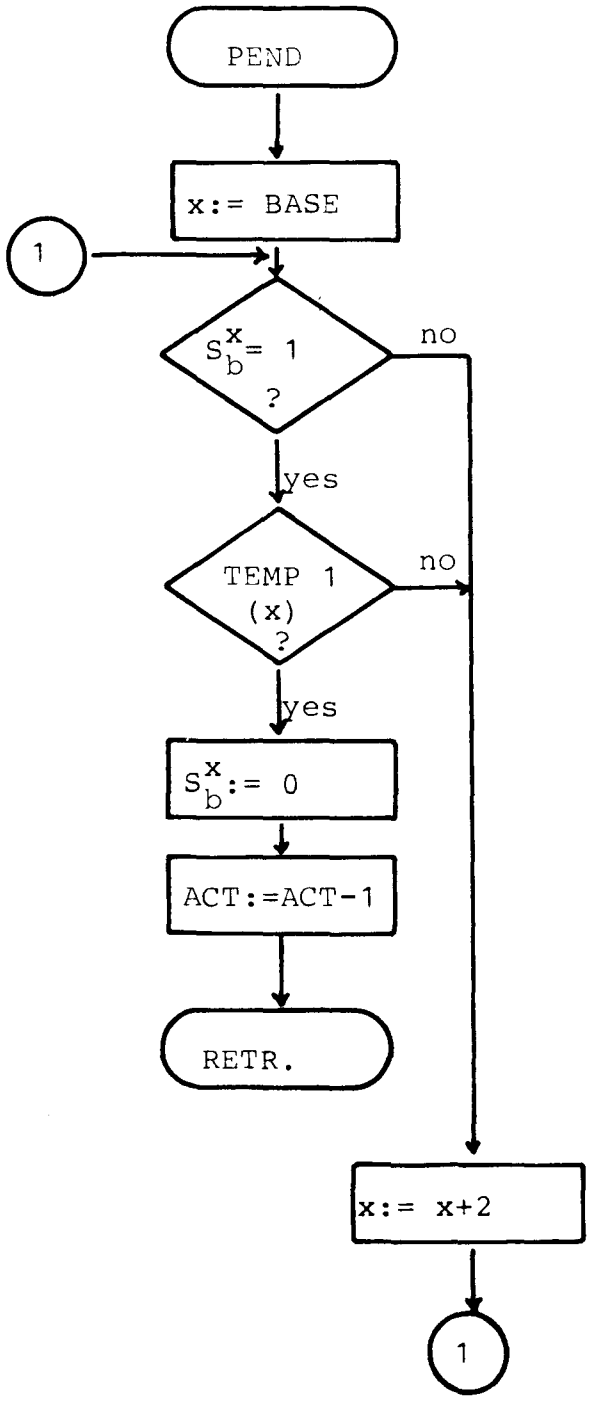
increment user count

return to the calling process

Fig. C.5.b.

PEND

This is the 'stop timer' routine. Beginning at the first descriptor, the user identity of each active process is checked against that in the message.
When a match is found the status bit of that descriptor is set to 'free', i.e. the timing process is ended.
Fig. C.5.c. gives a flow chart of this routine.



sub routine for ending a timing process

set x pointer to base position i.e. to first descriptor

test status bit (b7 of first byte of descriptor)

if 'active' compare the identity in this descriptor with that of the received message

if the same, set the status bit of the descriptor to zero i.e. end the timing process

decrement the user count

return to the calling process

otherwise continue with the next descriptor

Fig. C.5.c.

C.4. Remarks

In the preceding sections the operation of the Real Time function has been described. This could of course be improved - e.g. the use of an output queue, for message transfers to the control function, would reduce system waiting times - but it is considered that the basic solution advocated is sufficient to meet any requirements which will be made on the system.
Due to lack of time,neither assembler listing of the routines have been made nor has the system been constructed and tested.

STAGE/AFSTUDEEROPDRACHT VAKGROEP DIGITALE SYSTEMEN

Onderwerp: (zie toelichting) Het ontwikkelen van een front end configuratie voor een telefooncentrale

Naam: R. Bell

Adres: Giek 11

Woonplaats: Huizen N.H. tel. 02152 50182

Inschrijfnr.: 140405

Mentor: Ir. M. Stevens/ ir. H. Dortmans

Datum aanvang: oktober 1979

Datum einde: november 1980

Verslag ingeleverd d.d.: 28.11.80

Verslag beoordeeld d.d.:

Verslag paraaf:

Cijfer mentor:

Cijfer hglr.:

Bonnummer:

Bon datum:

Aantal bladen:

Vergeet U niet een korte inhoud + trefwoordenlijst bij te voegen ?

VERSLAG INLEVEREN BIJ VAKGROEPSSECRETARESSE !

R. Bell ECB 017
Afstudeerverslag. The design of a front end
Configuration for a digital telephone exchange
serving digital multi function subscribers.

THE afd. E vakgroep EB- trefwoordenlijst juni 1980:

aankomstverdeling, accumulator, A/D, adapter, adres, adresberekening, aftrekken, algorithme, ALU, analoog, APL, architectuur, assembler, associatief, automatisch,

BCD, bemonsteren, bestemming, besturen, betrouwbaarheid, beveiligen, binair, bit, blok, Boole, bootstrap, buffer, busbar,

cardiograaf, carry, cassette, cellulair, centrale, channel, cluster, code, coherent, communicatie, compiler, complement, conditie, controller, converter, conversie, core, correctie, correlatie, CPU, cross, cyclisch,

D/A, data, decode, delen, delta, demonstratie, demultiplex, design, detectie, diagnostisch, diagram, digitaal, diode, disk, display, division, DMA, dynamisch,

EBCDIC, ECL, ECMA34, editing, element, emulatie, encode, encryption, error, Euclides, excess, expressie, Eyemarker,

fase, FIFO, file, Fire, filter, fixed, flag, flip-flop, floating, floppy, flow, Fourier, fouten, frame, frequentie,

geheugen, generatie, GPSS, grafisch, Gray,

half, halfgeleider, Hamming, handleiding, hardware, herkomst,

I2L, IBM 360, I/O, ic, index, interaktief, interface, interpolatie, interrupt, input, instruktie,

job,

kanaal, karakter, kern, klok, komputer, kwhmeter,

LED, LIFO, line, LISP, loader, logica, lookahead,

macro, main, mask, masterslave, matrix, MCS-8, MCS8080, micro, minimalisatie, model, modem, modificatie, modulatie, multi, multiplex,

nagalm, netwerk, niveau,

one, ontwerp, operating, opstel, optellen, optimaal, optisch, output, overflow,

PAL, parallel, partitie, PCM, performance, Petri, PLA, PL/I, plot, plotter, point, pointer, polynoom, printer, prioriteit, processor, programma, PROM, pseudo, puls,

queue,

z.o.z.

RAM, random, real, recorder, redundantie, register, registratie, rekenkundig, relocatable, ROM, route, routine,

schakel, scheduler, Schottky, schuifregister, SDM, selector, sequentie, sequentieel, serie, signaal, simulatie, software, spel, stagnatie, stapel, star, storing, strook, subroutine, switch, synchronisatie, synchroon, syntax, system,

tabel, TDM, techniek, telefoon, tellen, teller, terminal, testen, time, transistor, transmissie, TTL, two, tijd, typewriter,

underflow, unit, universeel,

variabel, verkeer, vermenigvuldigen, vertraging, video, voeding,

wachten, wegvak, woord,

zender, zero,

Combinatie van trefwoorden levert b.v.: digitale cassette recorder, verkeer-simulatie-plot, schakeltechniek, enz.