1284

ONLINE INTERFACE FOR # 7 SIGNALLING SYSTEM

Dessertation submitted to Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the Degree of

MASTER OF TECHNOLOGY

1989

BHOOPESH RAGHAV

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067

This project work entitled ONLINE INTERFACE FOR # 7 SIGNALLING SYSTEM has been carried out by Mr. Bhoopesh Raghav, for partial fulfilment of the requirements of the degree of Master of Technology in Computer Science and Engineering of Jawaharlal Nehru University, New Delhi.

This work is original and has not been submitted so far in part or full for any degree in any University or Institute.

[illegible]

BHOOPESH RAGHAV

[illegible]

Lt.Col. C.P.C.Nath

Supervisor

SCSS, JNU, N.Delhi

Shankaranarayan

Mr. S. Shankarnarayan

Manager

CCITT group, C-DOT,N.Delhi

[illegible]

Prof Dr. N . P. Mukherjee

Dean

SCSS, JNU, N. Delhi

A C K N O W L E D G E M E N T

First of all I would like to express my deep gratitudes for Mr. S. Shankarnarayan, Manager CCITT # 7 group at C-DOT, for involving me in group activities and assigning me a project as important as this. His guidance and lectures given on technical matters time to time will remain unforgettable and very helpful in future.

I also take this oppurtunity to thank Lt. Col. C.P.C.Nath, for encouraging me to work on SS 7 and ISDN related activities and costantly guiding me throughout this project.

I also thank Jalaj Swami in no unequal terms, for his untiring help and his pivotal role in my interaction with C-DOT related activities.

I would also like to mention the names of Mr. Jasbir Singh, Mr. B. S. Chauhan and all group members for maintaining an encouraging and enthusiastic working environment which not only made this period memorable but important and quite enjoyable.

Bhoopesh Raghav

PREFACE

The SPC Exchange plays an important role in the new generation of Telecom networks. With the implementation of CCITT recommendations on # 7 Signalling System the signalling methodology has become very efficient and has provided a user with data communication facilities as a stepping stone towards ISDN and beyond .

Having fully implemented the Online Interface Commands, the Data Management in DSS Architecture will become very efficient and give a thrust to CDOT DSS through powerful Man Machine Commands and hence, will improve upon the functioning of SS7 in DSS.

CONTENTS

CONTENTS

PART II

CHAPTER 1

INTRODUCTION

The online inierface for #7 Signalling System provides an intrface to the Man Machine Commands. These commands when issued by an operator are routed through an Input Output Ptocessor(IOP) at Administration Processor(AP). The purpose of these commands is to modify or update the data required for exchange maintenance, call processing and outward signalling etc., maintained in the form of database at various processors. In the case of No 7 Signalling System, the AP communicates to the Signalling Unit through message communication. At SU the messages are received by the Numbr 7 Data Update Process(NDUP), which has been written for Online Interface in this project.

The NDUP is an external process running in a single state. Like other processes it's function is to add, delete or modify the necessary data related to Number 7 Signalling System at SU as implemented in C-DOT DSS.

The DSS is a modular architecture and the concepts of distributed processing are adopted for interaction between different processes effectively. To make the designing and implementation features of NDUP clear and to depict it's role in C-DOT DSS System, this report has been divided suitably as follows.

Starting with the chapter 2 in Part 1 on DSS Architecture, the report gives an overview of Hardware and Software sub-systems, Administration Module(AM), Base Module(BM) and Signalling Unit(SU) in particular, which support most of the processes NDUP is interacting with.

Administrative Processor supports the Man Machine Interface through Input Output Processor (IOP), this is described in chapter 3. The need of NDUP is more appreciable in case of Online commands which operate on distributed databases at AP, BP and SU in real time frame.

An overview has been given on Signalling System # 7 in Chapter 4. This deals with the acceptability of SS 7 as a Common Channel Signaling System, it's various mode of operation, Architecture of SS 7 and finally its implementation in C-DOT DSS.

The second part of this report has been devoted to the Online Interface Commands and its related features in C-DOT DSS.

Chapter 5 identifies the Network and Circuit related commands required for Online implementation in C-DOT DSS.

Chapter 6 describes the No 7 Data Update Process (NDUP) written for interfacing with other processes. This resides in Signalling Unit of DSS and communicates with other simlar processes through message communication. The design aspects of the NDUP are covered in Chapter 7. The Chapter 8 outlines the Command Flow and different function calls used in NDUP. Chapter 9 deals with the Network and Data management for No 7 SS. and gives an idea of distributed data maintained at AP, BP and SU. The last Chapter of this part is on the Software recovery strategy studied and implemented in NDUP for System recovery.

The SS 7 provides many useful services through its efficient signalling methodolgy. With the digital facilities extended upto subscriber premises, The ISDN is possible to implement through SS 7. Therefore the networking features are ananlysed in chapter 11 of part 3. Though this chapter is not essential to describe Online features explained here but considering the importance of this topic and having goen in the details of SS7, it looked interesting to study this aspect of SS 7 in more details and append the matter here.

To support this report an appendix is also included in the end which describes the naming convention standards followed at C-DOT, higher level supplementary files for global variables and data structures along with the pseudo code of NDUP process.

CHAPTER TWO

C-DOT DSS ARCHITECTURE

C-DOT digital switching system employs state of art technology. It has a modular, distributed architecture consisting of Admnistration Module, Central Module and number of Base Modules depending on the confoguration.

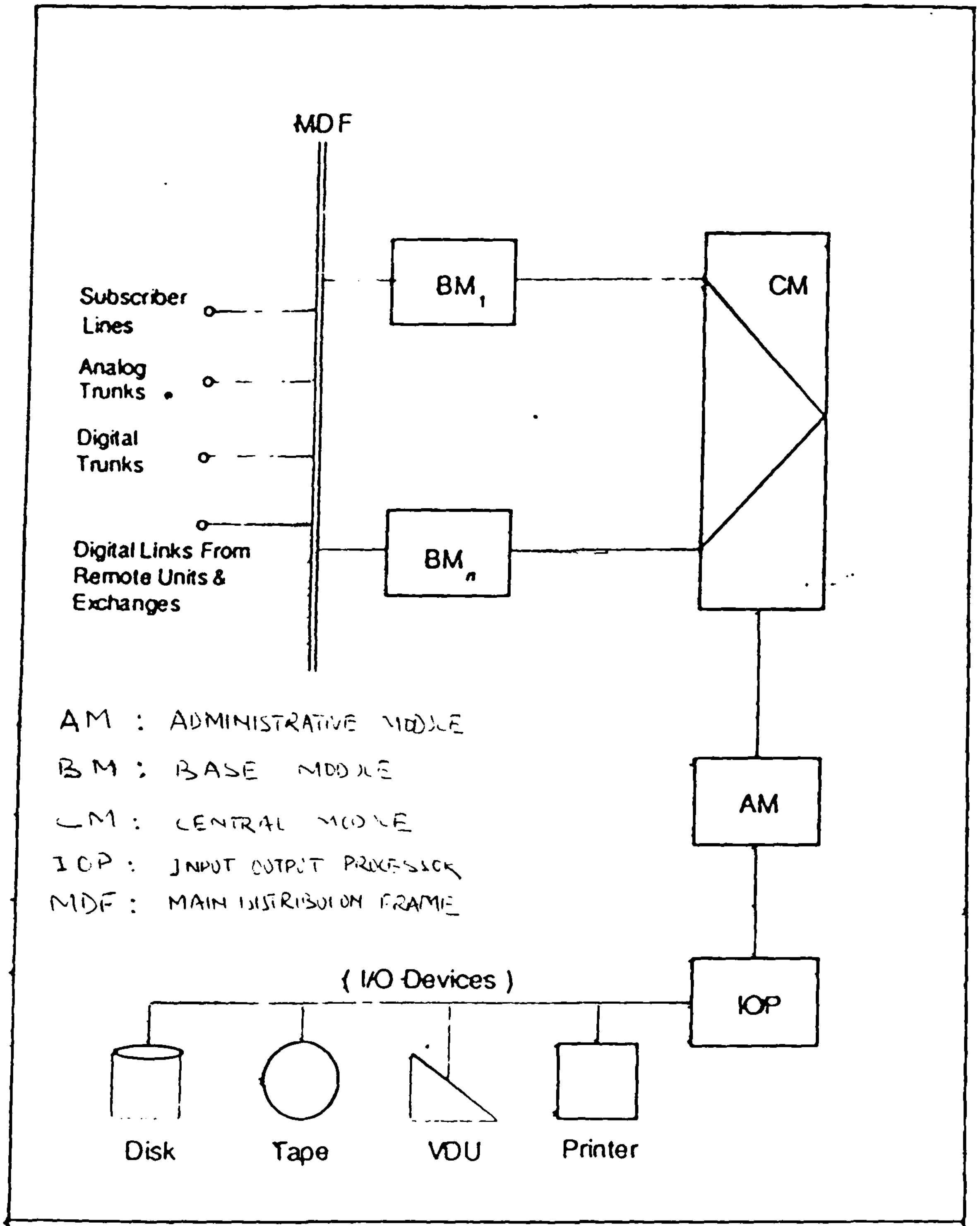
2.1 SYSTEM ARCHITECTURE

2.1.1 Administratve Module (AM) :

The AM provides system level administrative functions, which are as follows :

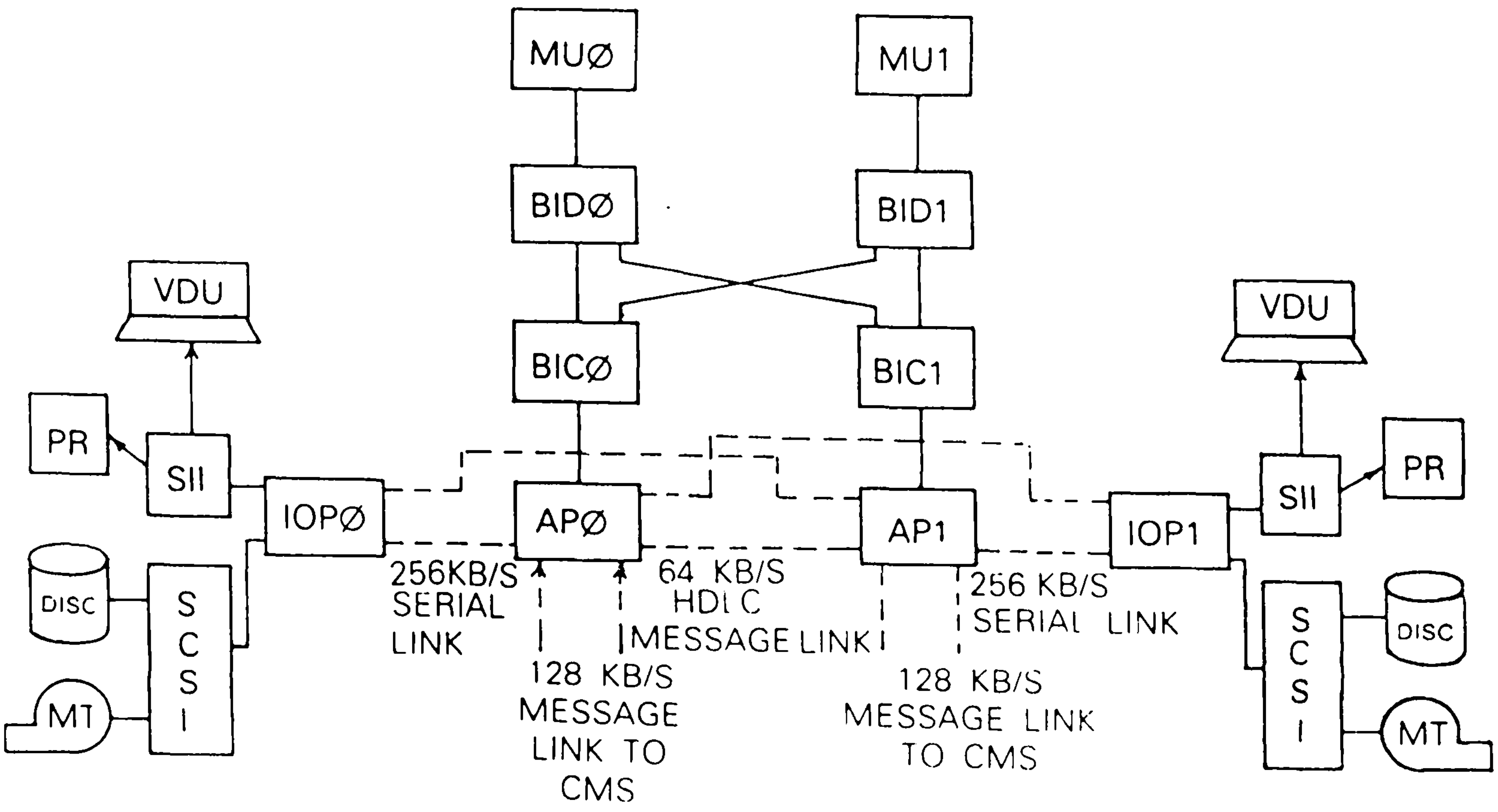
- Directory to equipment number translation for call processing support.
- System wide maintenance.
- Software recovery.
- Overall initialization,etc.

It also provides interface to mass memory and to man machine communication terminals through the Input Output



AM	:	ADMINISTRATIVE MODULE
BM	:	BASE MODULE
CM	:	CENTRAL MODULE
IOP	:	INPUT OUTPUT PROCESSOR
MDF	:	MAIN DISTRIBUTION FRAME

CDOT DSS BASIC ARCHITECTURE



ADMINISTRATIVE MODULE (AM)

-processor (IOP).

The important components of the AM are

- Administrative Processor
- Input Output Processor (IOP)

2.1.2 Central Module (CM):

The CM provides the connectivity between BMs, between AM and BMs, and between AM and CM itself.

The important components of the CM are

- Space Switch (SS)
- Space Switch Controller (SSC)
- Central Message Switch (CMS)

The BMs are interfaced to SS through two 512 channels, 10 bit parallel, 4 MB/sec links. Four of the 512 time slots carry control messages and rest of them carry digitized voice. For a call setup an idle time slot is selected by SSC and the desired terminals are connected to the time slot by the BM. Four time slots are terminated on four central message switches for message communication.

2.1.3 BASE MODULE (BM) :

The Base Module performs time switching portion of the time-space-time network. The lines and trunks terminate on terminal units in the BM. Each BM is capable of interfacing with CM through a duplicated link. It provides the necessary capability for configuring a multimodule switching system. The important components of the BM are

- Terminal unit(TU)

  consisting of

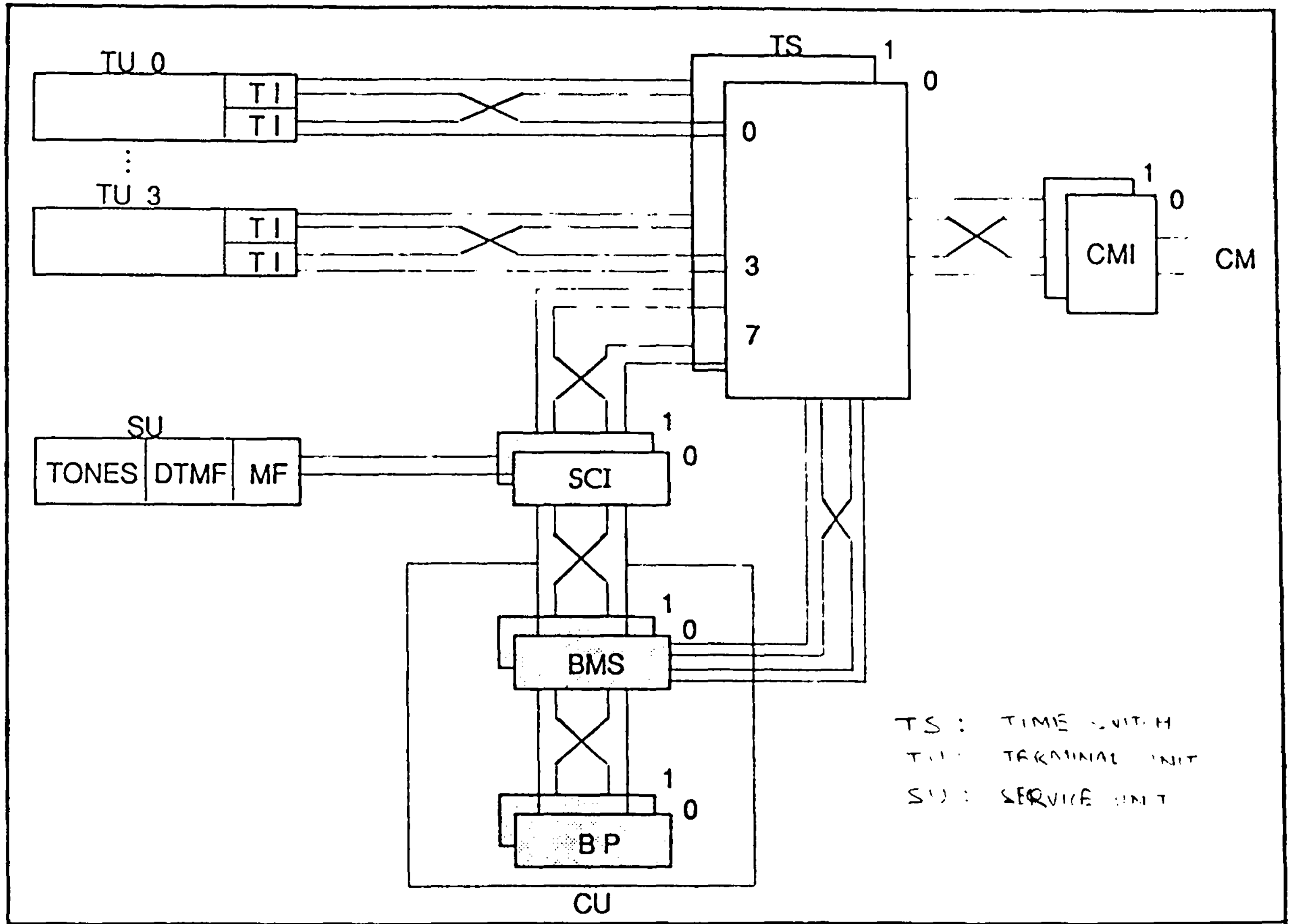
  - Terminal Interface Controller(TIC)
  - Signalling Procesor(SP)
  - Digital Trunk Intterface
  - Line Card
  - Analog Trunk Card, etc.

- Time Switch(TS)
- Base Message Switch(BMS)
- Service and Control Unit(SCU)

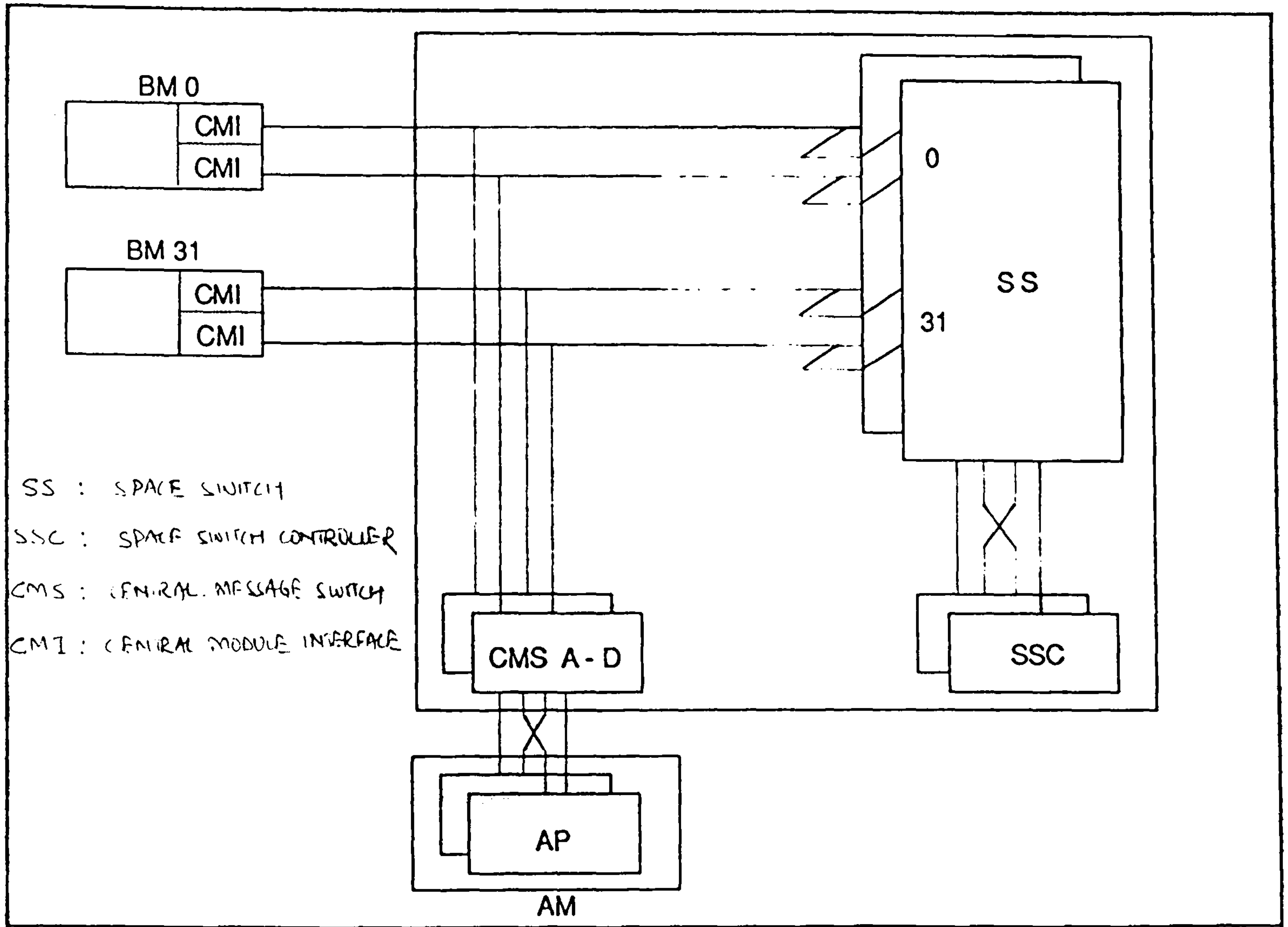
The TIC interfaces with signal processor. Four terminal groups of thirty two channels each are terminated on the TIC for providing 128 subscibers or trunks on 8 MB/sec link connecting TIC to the time switch.

The different telephony events like call organization detection,digit reception, digit out pulsing, reversal detection/ transmission etc are detected and performed by SP which inturn communicates to TIC.

The Digital Trunk Interface offers the same interface towards the SP as offered by line card for analog signal. The important functions performed by it are



: BASE MODULE ARCHITECTURE


BM 0
CMI
CMI
BM 31
CMI
CMI
0
31
S S
SS : SPACE SWITCH
SSC : SPACE SWITCH CONTROLLER
CMS : CENTRAL MESSAGE SWITCH
CMI : CENTRAL MODULE INTERFACE
CMS A - D
SSC
AP
AM

CENTRAL MODULE ARCHITECTURE

- electrical matching to the external interface.
- HDB-3 to NRZ code conversion.
- trace alignment / reconstitution / insertion / alarm information, etc.

The Line Card converts the signalling information from lines to a digital signalling format. The speech information is converted 8 bit PCM format and is multiplexed on duplicated 2 Mb/sec link.

The Analog Trunk Card is similar to line card with the difference that digits are pulses out on an outgoing trunk and ringing is not required to be fed on trunks.

The Time Switch (TS) performs time slot interchange functions for the base modules. The switching is performed for eight 128 channels , 4 Mb/sec serial links and 512 channel, 4 Mb/sec parallel links towards the central module. It also performs intra BM switching between 1024 time slot.

The Base Message Switch provides message communication between different controllers and base procassor. It also does the job of error detection and retransmission. The serial links used are of 128 Kb/sec and 64 Kb/sec for different controllers. The BM supports a maximum of two message device cards, each supporting sixteen HDLC links.

The important components of Service and Control Unit (SCU) are

- Service And Control Interface (SCI)
- Control Unit

SCI is similar to terminal interface but without an SP. It provides an interface between TS and service circuits and TS and Control unit.

The CU comprises of base processor memory interfaces and the memory unit. Bus interface connects the duplicated BP with duplicated memory unit.

To provide redundancy in the architecture all the control units and the links between the controllers are duplicated i.e., duplicated BPs, BMs, IOPs etc.

2.2 DISTRIBUTED PROCESSING IN C-DOT DSS

The C-DOT DSS architecture is highly modular. The Hardware and Software has been designed suitably to cater the distributed processing in it. Processors are built around 68K uP mostly and Processes use messages for inter-process-communication. The detail is as follows:

2.2.1 HARDWARE DESIGN

Processors

The processors are employed for both front end processing as well as main processing. Following processors are used in the system for this purpose-

- 6502 micro processor is used for front end.
- 16 bit 68000 microprocessor is used for main processing.

Main Memory

- 2 Mb dynamic memory board with on board RAM controller.

Mass storage

- Disk memory for data backup.
- magnetic tape for data transfer.

Module Control Unit

- 68000 micro processor based subsystem which is configurable as :
  - Base processor complex in BM.
  - Space switch controller in CM.
  - Administration processor in AM.

Interface Controller

8 bit, 6502 micro processor along with a time switching network.(128 channel multiplexed time slots). This same unit is configurable as :

- Terminal interface controller in TU.
- Service control interface controller in SCU

Message switch

This has two parts :

1. 68000 microprocessor based message switch controller having six HDLC links for message communications with other controllers.
2. Message switching device. One PCB of this module provides sixteen 64 Kb/sec HDLC channels time multiplexed to yield 2 Mb/sec link. Two such cards provide maximum of 32 time multiplexed HDLC channels. It is configurable as
   - Base message switch in BM.
   - Control message switch in CM.

     yield 2 Mb/sec link

2.2.2 SOFTWARE ARCHITECTURE

The switch hardware is surrounded by a number of software layers which provides higher level of abstraction. The important features of software are simplicity of layered architecture and loosely coupled modules, maintainability due to fault tolerant software and modular design and efficiency due to time critical process.

1 Operating System

The OS depending on it's functioning in DSS is divided as follows :

1. C-DOT real time operating system (CDOS).
2. OS for IOP.
3. Peripheral processor OS.

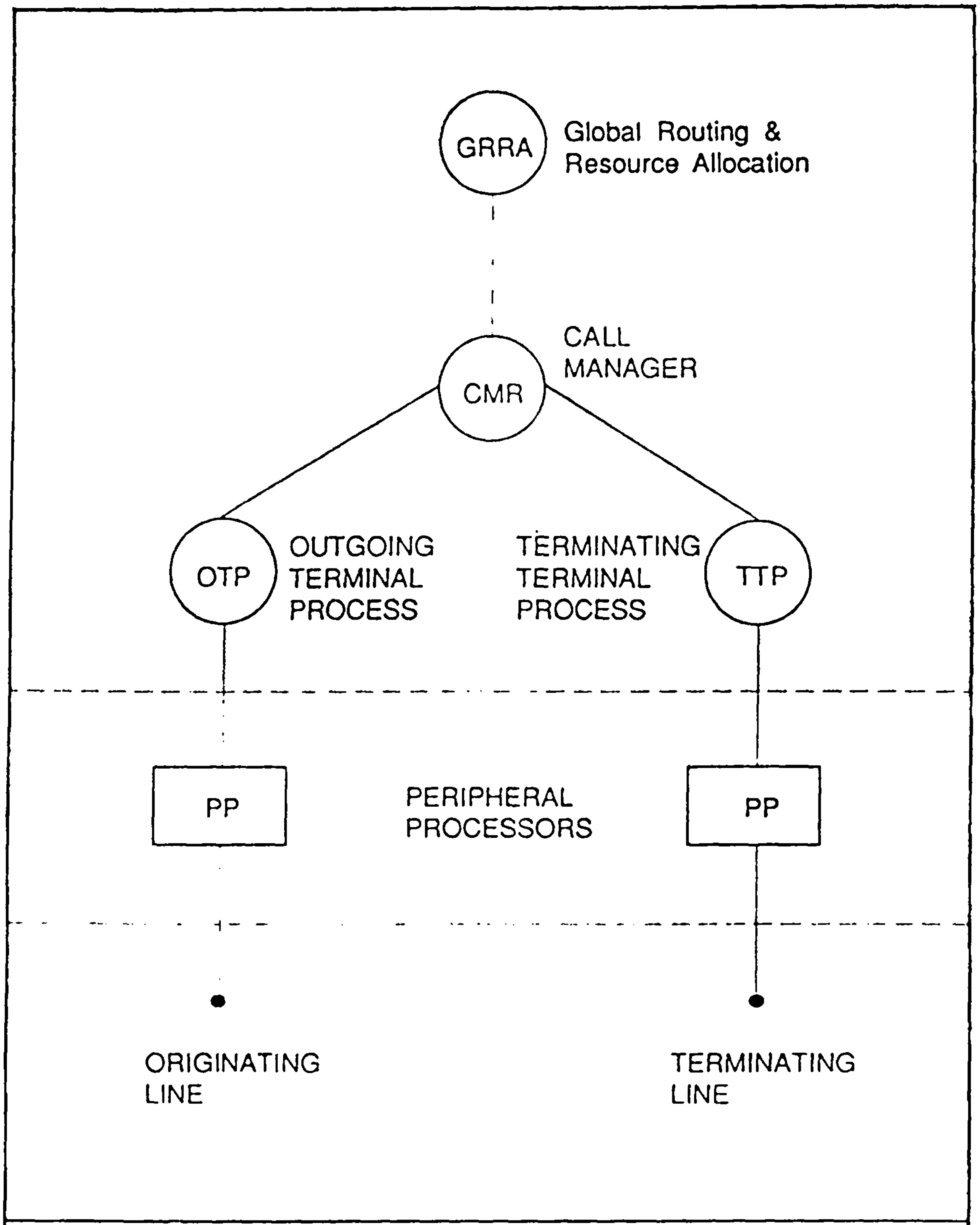
CDOS, designed suitably for minimizing overheads in terms of real time, provides effective interprocess communication between the processes residing in same or different processors. For communication it uses C.85 protocol which utilises the HDLC based message network.

Peripheral processor sub-system hides the telephony hardware. The 6502 microprocessor programmed in assembly level languages does the sensing and detection of events and communicate to BP for call processing. It also carries out maintenance related test functions on hardware.

The above mentioned software subsystems are distributed amongst BMs, CMs and AM. The OS does the partitoning for these subsystems in the form of processes and take care of inter process synchronization, interrupt handling, timing services and etc. This helps in efficient resource sharing.

2 Call Processing (CP) :

It constitutes the main part of the application


GRRA
Global Routing & Resource Allocation
CMR
CALL MANAGER
OTP
OUTGOING TERMINAL PROCESS
TERMINATING TERMINAL PROCESS
TTP
PP
PERIPHERAL PROCESSORS
PP
ORIGINATING LINE
TERMINATING LINE

PROCESSES IN CALL PROCESSING SOFTWARE

software. It uses the terminal interface controller primitives for controlling telephony features. Its important processes are

- Incoming Terminal Process No 7 (ICCTP7)
- Outgoing Terminal Process No 7 (OGTP7)
- Status Control Process (SCP)
- Global Path Control Process (GPC)
- Call Manager (CMR)

3 Administration Software:

It provides man-machine-interface with the system. Its salient features are :

- Provision for billing.
- Detailed billing record for STD calls.
- Exchange traffic and performance measurement.
- Managing recent change functions for program and data updates.

3 Maintenance Software :

It provides extensive mechanism for reliable operation of the system. Hardware units generate necessary triggers for the action to be taken by the subsystem. The important functions are:

- Fault detection for localization.

- Fault isolation and prevention from propogation.
- Fault recovery through the multiple levels of initializations.
- Preventive maintenance functions like audits , overload control, performance monitoring and etc.

5 DataBase (DB) :

The database has been organized for the following category :

- Fixed office data.
- Extended office data.
- Transient data for call processing.

Fixed office data comprises of terminal type, class of service, etc. Extended office data is maintained for feature related calls. Fixed and transient data are arranged on per terminal basis.

For easy recovery the important data items like billing, configuration atc are stored periodically on the disk backup.

6 CCITT No 7 SIGNALLING :

A part of the signalling system No 7 software is residing in No 7 SU which handles protocol message routing and network management. The other part, distributed among various BMs, integrates the No 7 SU with call processing, administration and maintenance sub-systems.

CHAPTER 3

MAN - MACHINE INTERFACE

The Man-Machine Language (MML) is used to facilitate the various administrative functions. It provides a transparent interface to the operator to communicate with the exchange and to modify the internal state.

The information related to the call processing, exchange maintenance, signalliing etc. is stored in the form of database. This database is normaly created or modified by the operator using man-machine commands.

The data is stored in different processors depending upon the requirements. Normaly data is stored in AP, BP, #7SU and at IOP. If at any time the exchange goes down, all the data for the exchange status can be downloaded from the IOP. The data local to each BM is stored there itself where as the global data needed by all the BM's is stored at AP.

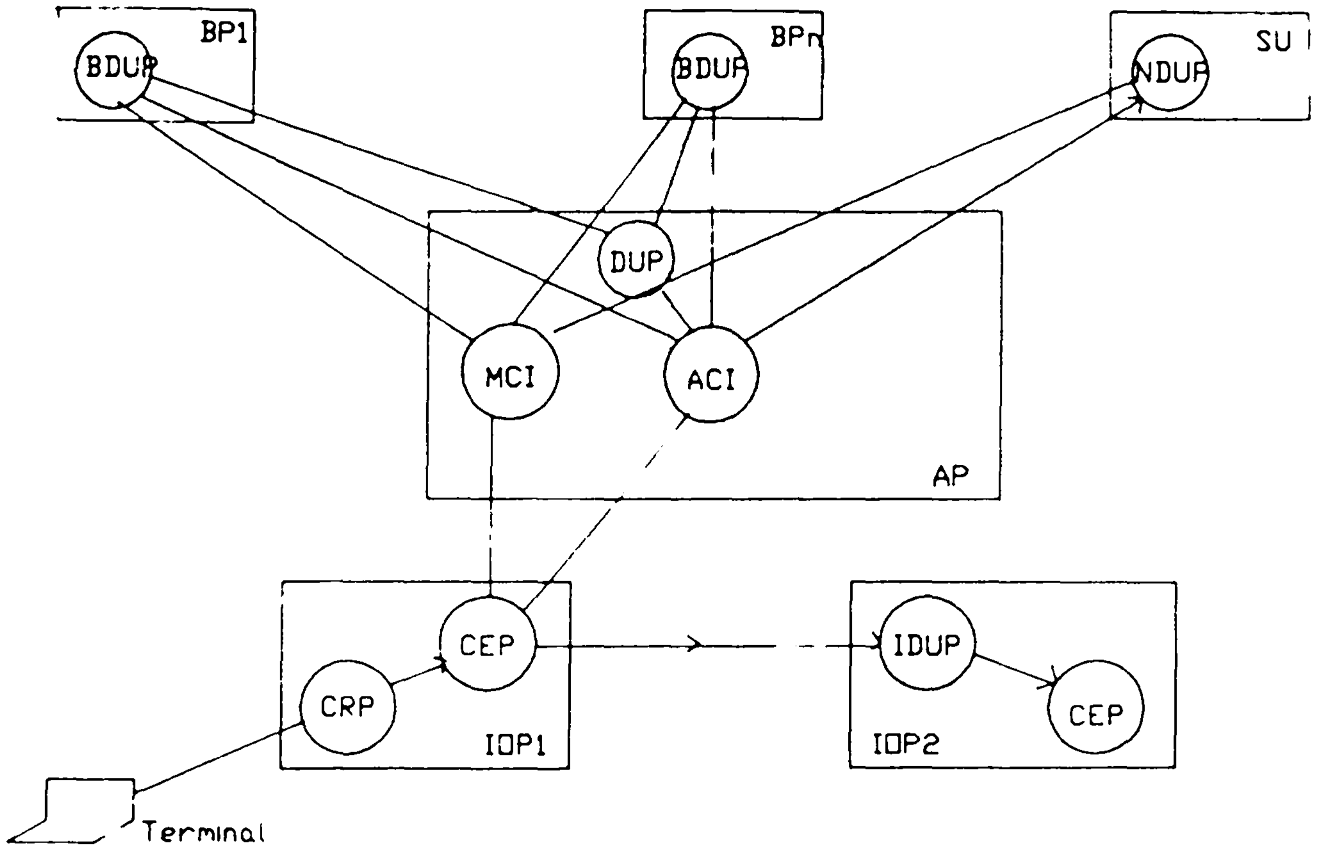
3.1 Man-Machine Interface Software Architecture

Keeping the flow into consideration, the MMI comprises of following different processes

- CRP : Command Recognition Process
- CEP : Command Execution Process
- IDUP : Input Output Data Update Process
- ACI : Administrative Command Interpreter
- DUP : Data Update Process (AP)
- BDUP : Base processor Data Update Process
- NDUP : No 7 Data Update Process

Command Recognition Process (CRP) : TH-6838

The CRP is a dynamic process residing at IOP. Whenever the operator logsin the terminal,this process is created. It is a command interpreter and takes the parameters from the imput keyed. It performs validation checks like range, set checks etc. on the input parameters. After all the validations it creates a dynamic process CEP, for that particular command and passes the relevant information.



CEP	:	Command Execution Process
CRP	:	Command Recognition Process.
ACI	:	Administratve Command Interpreter
DUP	:	Data Update Process
BDUP	:	Base Processor Data Update Process
NDUP	:	No 7 Data Update Process
IDUP	:	Input/Output Data Update Process

MAN MACHINE INTERFACE

Command execution process (CEP) :

There is one CEP per man-machine command. Its functions include :

- performing consistency checks.
- Exhaustive parameter validations.
- Updations of database at IOP and make the same in the exchange.

When the CEP is created, it gets arguments from CRP and makes an exhaustive parameter validation. After all the checks it sends a message to ACI from where the information goes to AP in AM, BP in BM and NDUP in #7SU. The CEP gets either a Success/Failure acknowledgement message from ACI, based on which it sends output message to CRP.

The destination of the command may be IOP, AP, BP or NDUP. If it is IOP , the CEP will be responsible for the data updation, otherwise CEP sends a message to ACI and waits for the response.

Input-Output Data Update Process (IDUP) :

IDUP is an eternal process residing in IOP. For reliability issues IOP work in Duplex mode. The CEP sends a message to IDUP which creates an identical CEP at the other IOP for the data updtion. The CEP of IOP1 waits for the message from duplicate CEP indicating the successful completion of updations before proceeding further.

Administrative Command Interpreter (ACI) :

The ACI is an eternal process residing in AP. It provides an interface between the IOP and AP/BP/SU. It gets the message from CEP and checks its class and command code to know about the place where the data updations are needed. Depending upon it the ACI sends messages to Data Updation Process at AP or Base Processor Data Updation Process at BP or No 7 Dataa Update process at SU. Depending upon the result it sends an acknowledgement message to CEP.

Data Update Processes :

It consisists of :

DUP at AP,

BDUP at BP and

NDUP at SU.

These processes are single state processes. These waitfor all possible messages in a single wait sate. The processing is done for every message received, at the end of which, these processes come back to the original state. For database updations these processes pass the control to the relevant command segment based on the Command class and Command code. Each process has the provision of sending and receiving acknowledgement and take necessary action.

3.2 Modes of operation :

The man-machine commands are executed in two modes.

i. Growth Mode and

ii. On-line Mode

Growth mode is also known as Off-line operation Mode. The command in this case updates the data at IOP only and at the Disk for backup. The modified data is periodically downloaded from IOP to the exchange.

In On-line mode data residing at both IOP and the exchange is modified. These modifications immediately effects the performance of the exchange. It is also known as Normal mode.

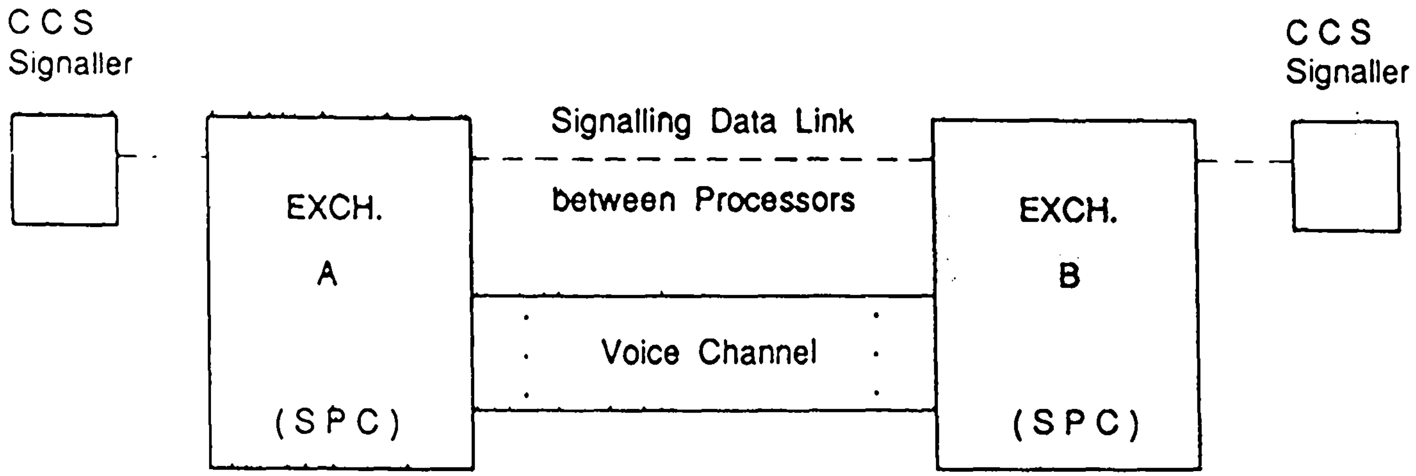
CHAPTER 4

No 7 SIGNALLING SYSTEM

4.1 COMMON CHANNEL SIGNALLING (CCS)

The CCITT No7 signalling system is a standard for interexchange common channel signalling system (CCS). A CCS is different from IN-BAND analog signalling channel as separate shared ((common) channel i.e. signalling link) is used to convey the signalling information.

In CCS networks terminal points or signalling points, (SP), are distinguished from switching nodes. SP like a control unit in SPC exchange, works as a source and destination for signalling information. CCS network is connected through trunks called signalling links (SL).


C C S
Signaller
EXCH.
A
(S P C)
Signalling Data Link
between Processors
Voice Channel
C C S
Signaller
EXCH.
B
(S P C)

C C S METHOD

4.2 Modes Of Operation

The modes of operation in CCS network are as follows :

1. Associated mode
2. Non-associated mode
3. Quasi-associated mode

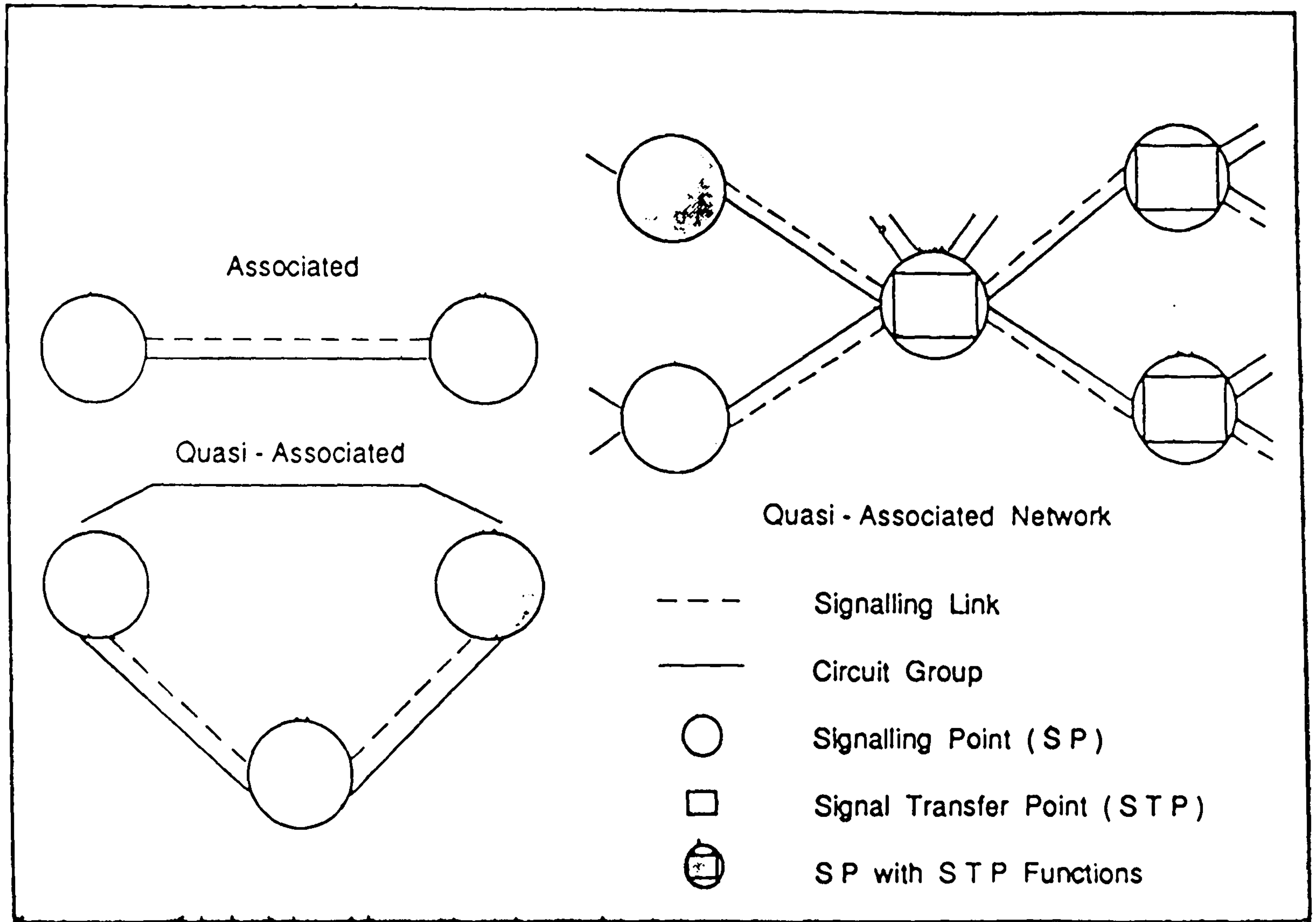
In associated mode, an SL serves the circuits belonging to a given route only. This has a disadvantage of low utilization if traffic is limited on the route.

In non-associated mode the utilization improves because of induction of STPs connecting SPs uniquely. In this case a fewer SLs provide the required traffic flow.

In quasi-associate mode, some SPs provide STP functions to make a CCS network combination of both associated and non-associated modes.

4.3 Layered Architecture of CCS

A CCS network incorporates the features essential to any intelligent communications network. It also resembles the OSI model of computer networking. The adjoining figure depicts the similiraty between the two.


Associated
Quasi - Associated
Quasi - Associated Network
Signalling Link
Circuit Group
Signalling Point (S P)
Signal Transfer Point (S T P)
S P with S T P Functions

C C S MODES

As against seven layers of OSI model, only four layers of SS7 are structured for communication. The lower levels 1, 2 and 3 together called message transfer part (MTP), correspond to the layers 1, 2 and 3 of OSI model respectively. OSI layer 4 is, though absent in SS No7 but a few networking related end to end communicatio features are incorporated in the form of network manager at level 3. The higher level layers 5, 6 and 7 of OSI are mapped at level 4 and distinctily divided in the form of user parts depending on the application.

level 1 :

signalling - data - circuit functions like provision of medium, error performance, protection, switching etc

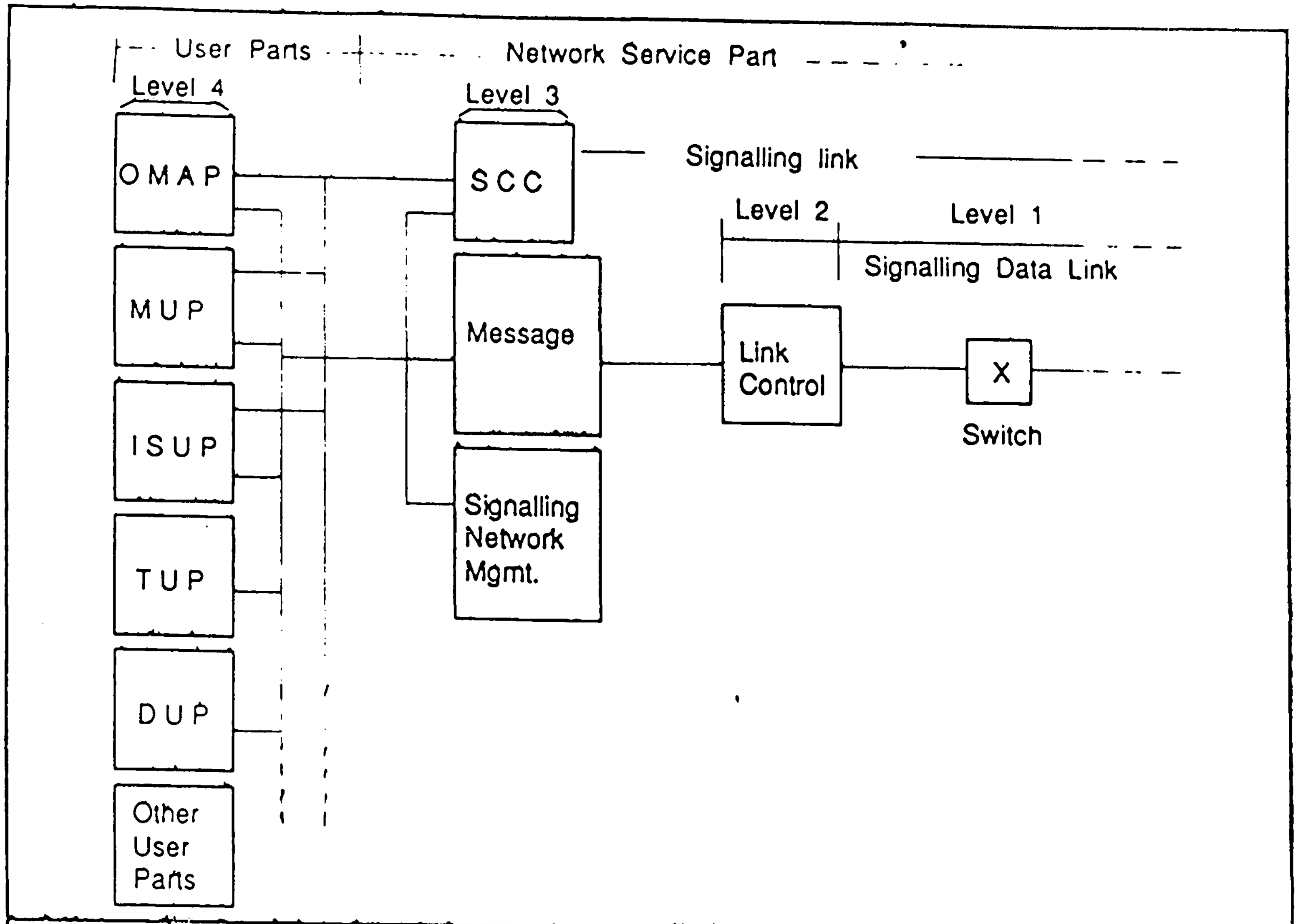
level 2 :

Signalling - data link functions

- message frame formatting
- Sequencing
- Error recovery through retransmission
- Error rate monitoring
- Link initialization

level 3 :

- Message handling
- Network managament
- Route and link management



LAYERED STRUCTURE OF CCITT NO. 7 SS

level 4 :

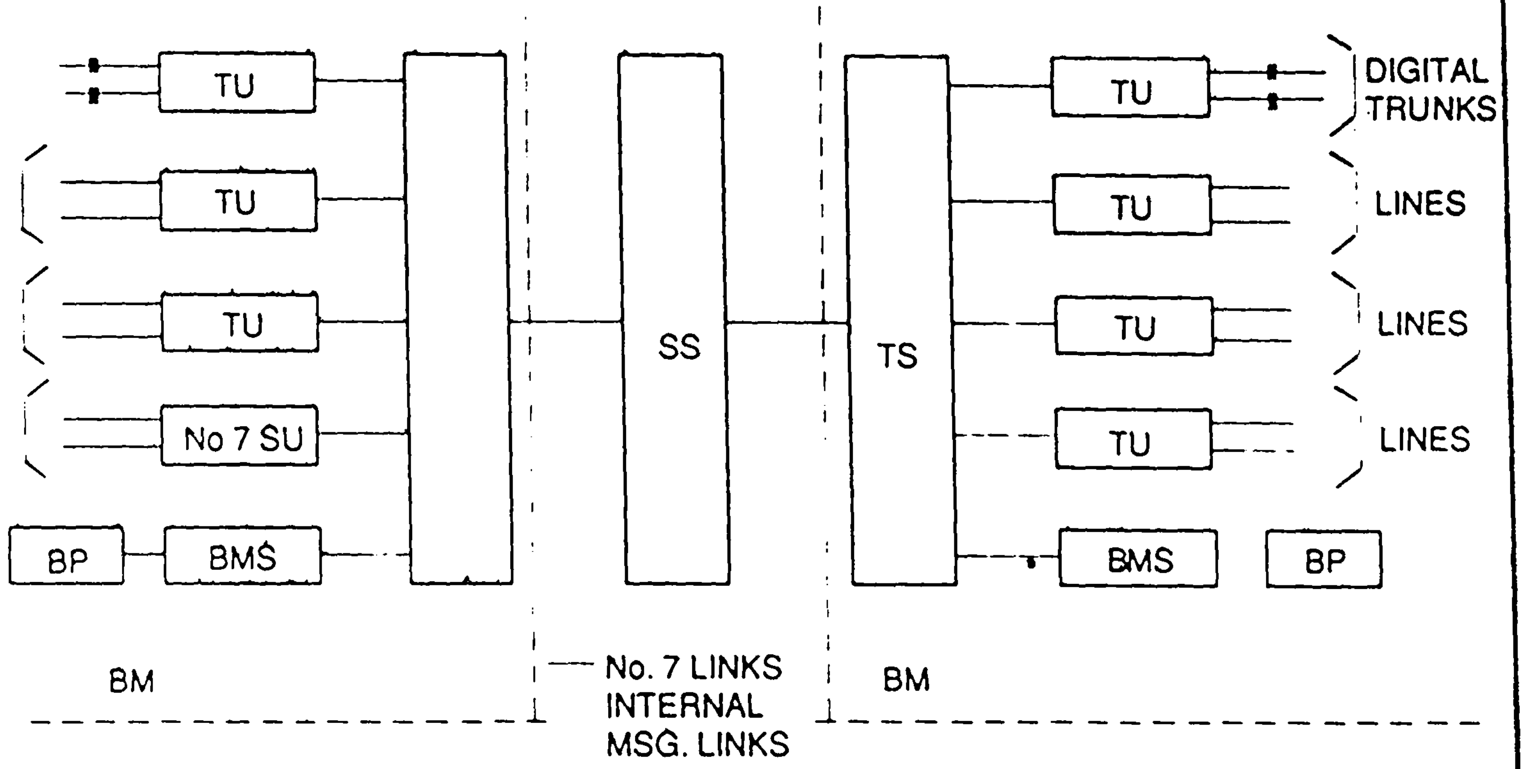
Application parts for TUP, DUP, OAMUP etc

4.4 IMPLEMENTATION OF No 7 SS IN DSS

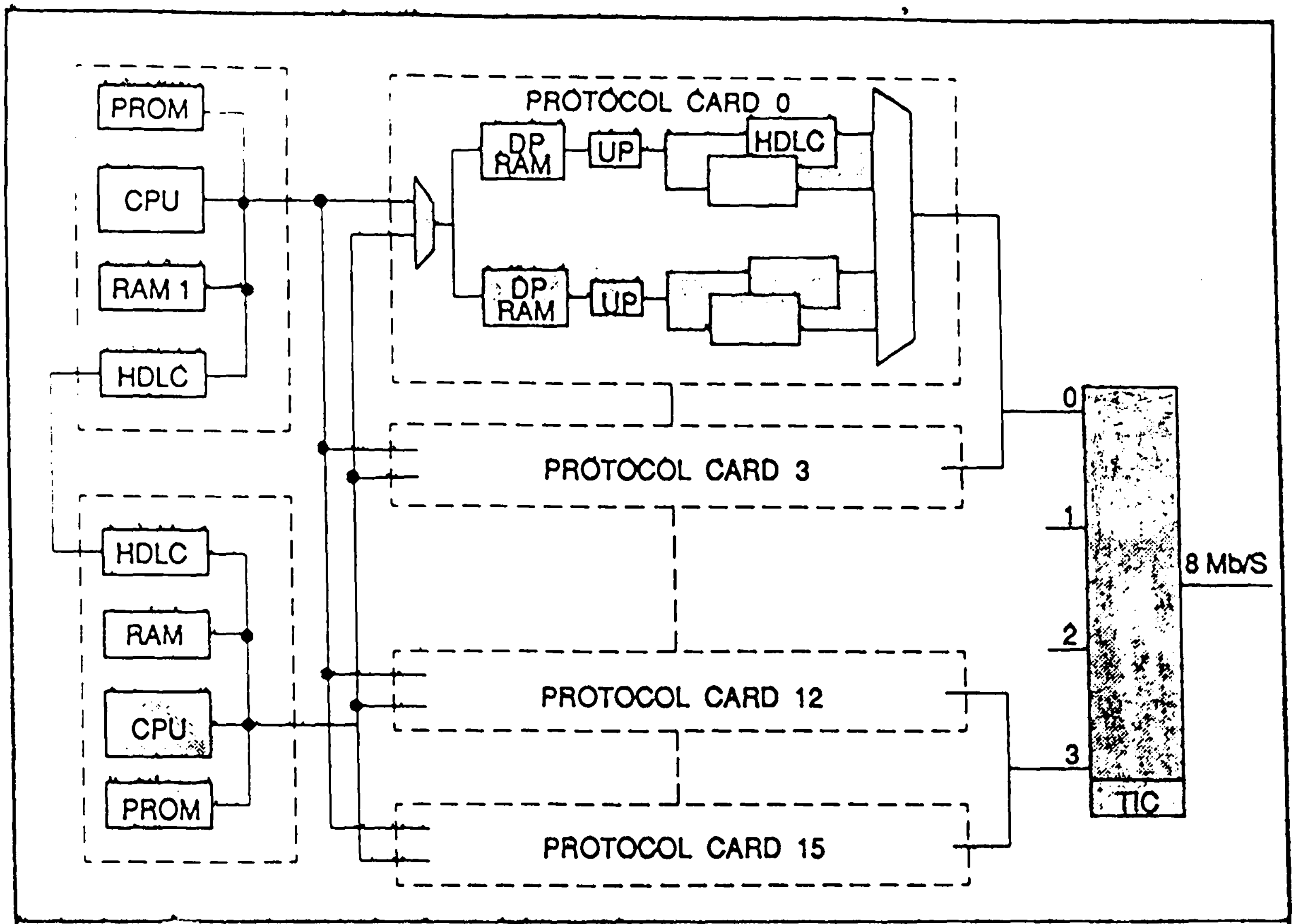
The No 7 SU is housed in a standard terminal frame connecting to the time switch of a BM on one of the eight serial interfaces of 128 time slots. An incoming No 7 message is distributed to the approriate process within the software sub-system and also to the approriate BM (in case of multiple BMs) based on incoming route and circuit identity. This helps in distribution of loadamongst BMs.

4.4.1 HARDWARE DESIGN

No 7 SU is implemented in DSS using two cards, protocol cards and SUP cards as shown in the figure. One protocol card handles eight protocol channels and such channels are controlled by one microcomputer. The mocrocomputer along with HDLC chip does the level 2 functions for each channel. Buffers are used in memory for incoming messages and cleared later on after outgoing messages. The 64 Kb/s serial streams of 32 channels are converted to 2048 Kb/s. Four such links (8 Mb serial) connect to a standard TIC card for Time Switching Interface.



NO. 7 SU IN C-DOT DSS



NO. 7 S S HARDWARE SCHEME

The second card, signal unit processor (SUP), in duplicated mode has on board ROM for code and RAM for routing tables. SUP polls the protocol channels for incoming messages, analyse and route on to outgoing channels. A serial link between duplkicated SUPs is used for updating of routing tables to maintain consistency and easy recovery.

4.4.2 SOFTWARE ARCHITECTURE

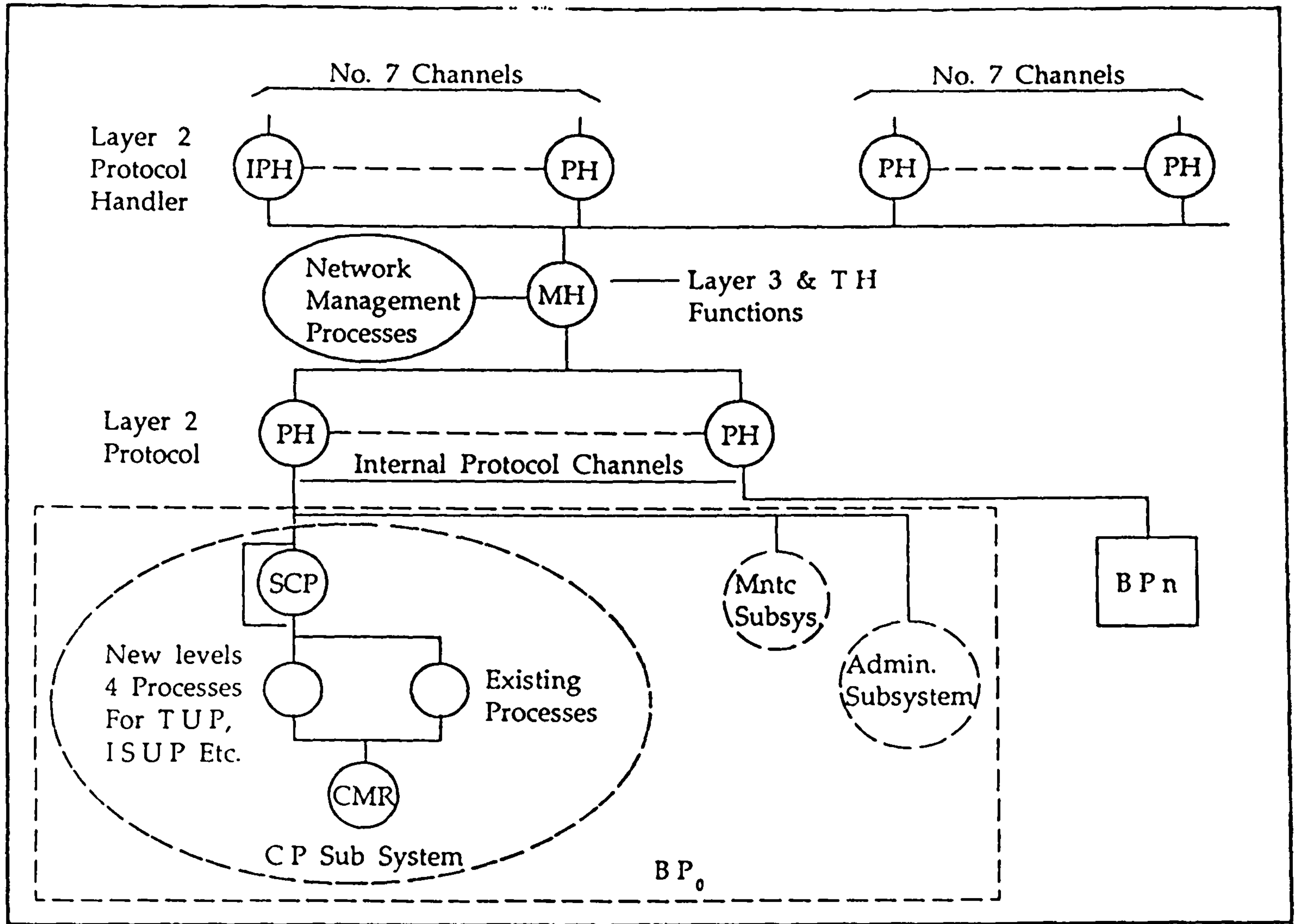
The software is implemented in two parts, one in No 7 SU and the other in BPs and AP.

The No 7 SU software primarily consists of message handler (MH) and network manager. The MH implements the level 3 functions of routing, discrimination, distribution, etc. It polls the protocol cards, analyzes the header and route the message on the outgoing channel.

At regular intervals the MH is interrupted and the control is passed on to a network management process which takes the subsequent action under the control of C-DOS ported on toNo 7 SU.

Software in BP containing different processes with the sub-system software, implements the level 4 user part. The TUP includes incoming No 7 tarnsfer part with CP.

ISUP is implemented with ICISTP, OGISTP and rest of the CP.



NO. 7 S S SOFTWARE SCHEME

CHAPTER 5

ON LINE DATA UPDATION COMMANDS USED FOR MAN_MACHINE COMMUNICATION

The Online commands are issued bu an operator in a standard format. These are received by Command Recognition Process (CRP) and communicated to Administrative Command Interpreter (ACI) through Command Execution Process(CEP).

ACI analyzes the command class and command code to know the destination processor and required database to operate upon. The commands dealt for No 7 Signalling System are related to either

- Network features
- Circuit features

ACI after analysis sends the message either to Data Update Process(DUP) or to Base Processor Data Processor (BDUP) for the updation at the respective databases. In case when the Data is required to be modified at the No 7 SU, the BDUP sends the message to No.7 Data Update Process (NDUP). These procsses after updation send an acknowledgement, which in case of NDUP goes to BDUP and finally to ACI for backward confirmation to the operator.

5.1 CIRCUIT RELATED COMMANDS:

1. Create Self Point Code	- cre_spc
2. Create Circuit Group Set	- cre_cgs
3. Modify Circuit Group Set	- mod_cgs
4. Delete Circuit Group Set	- del_cgs
5. Alloc. Circuit Identity Code	- aloc_cic
6. De-alloc. Circuit Identity Code	- daloc_cic

5.2 NETWORK RELATED COMMANDS:

1. Create LINK Set	- cre_sls
2. Modify Link Set	- mod_sls
3. Delete Link Set	- del_sls

4. Create Link Set Bundle	- cre_lsb
5. Modify Link Set Bundle	- mod_lsb
6. Delete Link Set Bundle	- del_lsb
7. Create Signaling Route Set	- cre_srs
8. Modify Signaling Route Set	- mod_srs
9. Delete Signaling Route Set	- del_srs

OTHER NETWOK RELATED ONLINE COMMANDS :

*1. Add Data Link	- add_dl
*2. Delete Data Link	- del-dl
*3. Create Data Link Group	- cre_dlg
*4. Modify Data Link Group	- mod_dlg
*5. Delete Data Link Group	- del_dlg

* - these modifications are not reflected in NDUP.

CHAPTER 6

NUMBER 7 DATA UPDATE PROCESS (NDUP)

6.1 PROCESS DESCRIPTION

The NDUP is implemented as a single state process. This means that the process waits for all possible messages in a single wait state. The processing is done for every message received, at the end of which NDUP goes back to the original wait state.

Each job for the data updation is identified by a code byte. The job entries, in the form of messages are maintained by a linked list called NDUPQ.

The commands handled by the NDUP are organised into a collection of command segments. Each segment consists of the command codes used for different database actions. The correponding command segment gains control in the 'forever' function call and execute the command.

The NDUP also maintains the history of past few commannd executed, in the history buffer, pointed by the hist_buf_pointer. This helps in avoiding duplicate command execution and in a way checking the redundancy in the database updations.

The NDUP process is a single state process, therefore, after sending a message to an external process for record fetching or to restart, it waits in original state where it again gains control for the incomplete commands. The status of each command therefore, is stored by the NDUP maintained in the form of a doubly linked list called NPNODE for this purpose.

The NDUP interfaces with other processes also. It remains in the receive message state before it is invoked by

- a command message from Administration Command Interpretor (ACI).
- a command message from DUP.
- a command message from BDUP.

CHAPTER 7

DESIGN ASPECTS

7.1 LOCAL DATA STRUCTURES

7.1.1

Name	: command_ptr(Command pointer table).
Purpose	: To hand over control to the corresponding command segment to process a input command message
Location	: SU memory. (Read only)
Memory size	: 2.0 K Assuming 25 command classes, each class having a maximum 20 command codes(provision for extra command classes is taken into consideration).
Organization	: a two dimensional array of pointers to functions.
Access mechanisim	: indexed by command class and command code.

Accessed by : NDUP

NDUP accesses this data structure only once in passing control to the command segment handling the specific command message.

Initialization :

NDUP initialization routine. This is called when the process first comes up. The elements of the command pointer array are made to point to the various command segments.

Audit : A back up is on disk. Periodically the audit verifies data consistency in memory and on the disk.

7.1.2.

Name : NPNODE

Purpose :To maintain a link list of all pending command messages including their present status and other information.

Fields :Each node contains the following fields:

Status : gives the current status of the command.

Header : contains message mnemonic, message type, subfield 2 and sender process id.

Command header: contains crp_id, size, status, session id, job id, user id and device id.

Mesg_ptr : points to the message.

Left :Points to the previous node in NDUPQ.

Right :Points to the next node in NDUPQ.

oldval_ptr :points to the old value of data

Rslt_ptr : points to the combined result

Location	: SU memory
Memory size	: Each node occupies 40 bytes.
Organization	: a double linked list of nodes.
access mechanism:	sequential search. (user id,job,id & session id together serve as control key)

Initialization	: NDUP initialization routine does the initialization. This is called when the process just comes up. The header of the link list is created.

7.2 MESSAGES USED

7.2.1 Messages for DUP -> NDUP

```
typedef struct  {
                Ulong   crp_id;
                Ushort  user_id;
                Ushort  size;
                Uchar   status_byte;
                Uchar   output_dev;
                Uchar   session_id;
                Uchar   job_id;
                Execution_num   exe_num;
        } Cmdhdr;
```

result header message:

```
typedef struct  {
                Ushort  user_id;
                Ushort  size;
                Uchar   no_of_entries;
                Uchar   output_dev;
                Uchar   session_id;
                Uchar   job_id;
                Uchar   dummy_byte;
                Uchar   status_byte;
                Long    crp_id;
                Execution_num   exe_num;

        } Rslthdr;
```

7.2.2 Messages for circuit related commands:

cgs related message

```
typedef struct
   {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         cgs_num;
   Uchar          pcm_type;
   Ushort         dpc;
   Ushort         byte1_slot_cic;
   Par_slot_cic   slot_cic[VMPOS(byte1_slot_cic)];
   }Ncktcre07;
```

cic related mesage

```
typedef struct
   {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         byte1_rcic_info;
   Par_rcic_info  rcic_info[VMPOS(byte1_rcic_info)];
   }Ncktcre09;
```

7.2.3 Messages for Network related commands

cgs and dlg related message

```
   typedef struct
      {
      Hdr            hdr;
      Cmdhdr         cmdhdr;
      Ushort         dlg_num ;
      Ushort         cgs_num ;
      Ushort         dpc ;
      Uchar          sgnl_net_id;
      Uchar          stdby_dl ;
      Uchar          trmn_flg ;
      Uchar          unuse_dl ;
      Ushort         byte1_dl_lst ;
      Par_dl_info    dl_info[VMPOS(byte1_dl_lst)] ;
      }Nnetcre01;
```

lsb related message

```
typedef struct
   {
   Hdr           hdr;
   Cmdhdr        cmdhdr;
   Ushort        lsb_num;
   Uchar         sw_opt;
   Ushort        other_end_pc ;
   Ushort        byte1_stp_users;
   Par_stp_users stp_users[VMPOS(byte1_stp_users)];
   }Nnetcre02;
```

ls and lsb related messages

```
typedef struct
   {
   Hdr                  hdr;
   Cmdhdr               cmdhdr;
   Ushort               ls_num ;
   Ushort               lsb_num ;
   Uchar                max_act_links ;
   Uchar                min_act_links ;
   Uchar                max_avl_links ;
   Uchar                min_avl_links ;
   Uchar                act_ls ;
   Uchar                recovery_opt ;
   Ushort               byte1_pc_served_lst ;
   Ushort               byte1_link_lst   ;
   Par_pc_served_lst    pc_served_lst[VMPOS(byte1_pc_served_lst)] ;
   Par_link_id          link_id[VMPOS(byte1_link_lst)] ;
   }Nnetcre03;
```

dlg related messages

```
typedef struct
   {
   Hdr           hdr;
   Cmdhdr        cmdhdr;
   Ushort        dlg_num ;
   Ushort        byte1_dl_lst ;
   Par_dl_info   dl_info[VMPOS(byte1_dl_lst)] ;
   }Nnetcre05;
```

```
typedef struct
   {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         dlg_num ;
   Uchar          stdby_dl ;
   Uchar          unuse_dl ;
   }Nnetcre06;
```

lsb related messages

```
typedef struct
   {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         lsb_num;
   Uchar          sw_opt;
   Uchar          sgnl_net_id;
   Ushort         other_end_pc ;
   Ushort         byte1_stp_users;
   Par_stp_users  stp_users[VMPOS(byte1_stp_users)];
   }Nnetcre07;
```

ls and lsb related messages

```
typedef struct
   {
   Hdr                   hdr;
   Cmdhdr                cmdhdr;
   Ushort                ls_num ;
   Ushort                lsb_num ;
   Uchar                 max_act_links ;
   Uchar                 min_act_links ;
   Uchar                 max_avl_links ;
   Uchar                 min_avl_links ;
   Uchar                 act_ls ;
   Uchar                 recovery_opt ;
   Ushort                byte1_pc_served_lst ;
   Ushort                byte1_link_lst  ;
   Par_pc_served_lst     pc_served_lst[VMPOS(byte1_pc_served_lst)] ;
   Par_link_id           link_id[VMPOS(byte1_link_lst)] ;
   }Nnetcre08;
```

dl related message

```
typedef struct
   {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
```

```
        Ushort        dlg_num ;
        Ushort        byte1_dl_lst ;
        Par_dl_info   dl_info[VMPOS(byte1_dl_lst)] ;
        }Nnetcrel0;


       typedef struct
        {
        Hdr                 hdr;
        Cmdhdr              cmdhdr;
        Ushort              ls_num ;
        Ushort              lsb_num ;
        Uchar               sgnl_net_id;
        Ushort              dpc ;
        }Nnetcrel2;


     typedef struct
        {
        Hdr           hdr;
        Cmdhdr        cmdhdr;
        Ushort        lsb_num;
        Uchar         sgnl_net_id;
        Ushort        other_end_pc ;
        }Nnetcrel3;

     typedef struct
        {
        Hdr           hdr;
        Cmdhdr        cmdhdr;
        Ushort        dlg_num ;
        }Nnetcrel4;

  typedef struct   {
                   Hdr      hdr;
                   Cmdhdr   cmdhdr;
                   Ushort   rel_name;
                   Ushort   key_size;
                   Ushort   rec_size;
                   Uchar    key_type;
                   Uchar    code_byte;
                   Uchar    dberr_cnt;
                   Uchar    bm_no;
                   Uchar    key_val[VMPOS(key_size)];
                   Uchar    rec_val[VMPOS(rec_size)];
        } Ndbrec01;
```

7.2.4 Messages for record fetching:

```
typedef struct  {
                Hdr      hdr;
                Cmdhdr   cmdhdr;
                Ushort   rel_name;
                Ushort   key_size;
                Uchar    key_type;
                Uchar    code_byte;
                Uchar    dberr_cnt;
                Uchar    bm_no;
                Uchar    key_val[VMPOS(key_size)];
        } Nreqrcrd01;
```

7.2.5 Acikill message

```
  typedef struct  acikill01
                {
                Hdr             hdr;
                Cmdhdr          cmdhdr;
                Uchar           reason;
                Uchar           dummy;
        } Acikill01;
```

7.2.6 Ack. timeout message

```
typedef struct  {
                Hdr      hdr;
                Uchar    code;
                Uchar    timer_type;
        } N_ackto01;


typedef struct {
                Hdr      hdr;
                Cmdhdr   cmdhdr;
                Char     date[20];
        } Asettim;

typedef struct {
                Hdr      hdr;
                Rslthdr  rslthdr;
                Uchar    err_typ;
                Uchar    dummy1;
        } Asettimr;
```

```
typedef struct  {
                Hdr     hdr;
                Rslthdr rslthdr;
                Ushort  error_typ;
                union {
                        Ushort  error_no;
                        Ushort  par_no_lpt;
                      } par_error;
                Char    rel_file_par_value[2];
        } Acmderr01;
```

7.2.7 Result messages for Network and Circuit related commands

```
typedef struct
   {
   Hdr           hdr;
   Cmdhdr        cmdhdr;
   Ushort        suc_typ ;
   Ushort        dummy ;
   }Ncktcrer07;

typedef struct
   {
   Hdr           hdr;
   Cmdhdr        cmdhdr;
   Ushort        suc_typ ;
   Ushort        dummy ;
   }Ncktcrer09;

typedef struct
   {
   Hdr           hdr;
   Cmdhdr        cmdhdr;
   Ushort        suc_typ ;
   Ushort        dummy ;
   }Nnetcrer01;

typedef struct
   {
   Hdr           hdr;
   Cmdhdr        cmdhdr;
   Ushort        suc_typ ;
   Ushort        dummy ;
   }Nnetcrer02;
```

```
typedef struct
   {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         suc_typ ;
   Ushort         dummy ;
   }Nnetcrer03;

typedef struct
   {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         suc_typ ;
   Ushort         dummy ;
   }Nnetcrer05;

typedef struct        {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         suc_typ ;
   Ushort         dummy ;
   }Nnetcrer06;

typedef struct        {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         suc_typ ;
   Ushort         dummy ;
   }Nnetcrer07;

 typdef struct
   {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         suc_typ ;
   Ushort         dummy ;
   }Nnetcrer08;

typedef struct
   {
   Hdr            hdr;
   Cmdhdr         cmdhdr;
   Ushort         suc_typ ;
   Ushort         dummy ;
   }Nnetcrer10;
```

```
typedef struct
    {
    Hdr             hdr;
    Cmdhdr          cmdhdr;
    Ushort          suc_typ ;
    Ushort          dummy ;
    }Nnetcrerl2;

typedef struct
    {
    Hdr             hdr;
    Cmdhdr          cmdhdr;
    Ushort          suc_typ ;
    Ushort          dummy ;
    }Nnetcrerl3;

typedef struct
    {
    Hdr             hdr;
    Cmdhdr          cmdhdr;
    Ushort          suc_typ ;
    Ushort          dummy ;
    }Nnetcrerl4;
```

CHAPTER 8

FUNCTION CALLS

Functionol Description

Following is the list of functions used for implementation of NDUP process.

	File name	Function name
1	NPNNDCS_XXXX_ACIKILL.C	nnd_xxxx_acikill()
2	NPNNDCS_XXXX_ARG.C	nnd_xxxx_arg()
3	NPNNDCS_XXXX_CHKHIST.C	nnd_xxxx_chkhist()
4	NPNNDCS_XXXX_COPYSTR.C	nnd_xxxx_copystr()
5	NPNNDCS_XXXX_CRENODE.C	nnd_xxxx_crenode()
6	NPNNDCS_XXXX_DELNODE.C	nnd_xxxx_delnode()
7	NPNNDCS_XXXX_FOREVER.C	nnd_xxxx_forever()
8	NPNNDCS_XXXX_HNTHIST.C	nnd_xxxx_hnthist()
9	NPNNDCS_XXXX_HNTNODE.C	nnd_xxxx_hntnode()

10	NPNNDCS_XXXX_INITCMD.C	nnd_xxxx_initcmd()
11	NPNNDCS_XXXX_ENTER.C	
12	NPNNDCS_XXXX_PRCINIT.C	nnd_xxxx_prcinit()
13	NPNNDCS_XXXX_ROLLBACK.C	nnd_xxxx_rollback()
14	NPNNDCS_XXXX_EXTERN.H	
15	NPNNDCS_XXXX_DSELECT.C	nnd_xxxx_dselect()
16	NPNNDCS_XXXX_INSRCRD.C	nnd_xxxx_insrcrd()

8.1 Flow of Commands in NDUP

The important steps involved in the command processing in NDUP is as follows:

The NDUP is created by CDOS using system calls. The initialization could be due to either of these reasons:

POWER ON (first time initialisation)
SYSBOOT (Stable restart due to S/W failyre)
STBCLR (Stable clear)
PARTINIT (Partial initialisation)
SOFTSTRT (S/W restart)
ROLLED_BACK (Process roll back)

At the time of initliasation nnd_xxxx_enter() function is called with an argument for one of the above mentioned reasons.

This argument is passed on to the function nnd_xxxx_arg()

. depending on the type of initialisation nnd_ xxxx_prcinit() ic called.

. nnd_xxxx_prcinit() initialises all important data structures i.e., map table, history buffer, timer structures etc.

nnd_xxxx_arg() also calls nnd_xxxx_forever() for entering into an ongoing forever loop.

Forever loop is the basic characterstics of NDUP which makes it a single state process. It receives message in this state and calls nnd_xxxx_chklist() or nnd_xxxx_ackto() depending on the opcode in the message.

In nnd_xxxx_chklist it makes a search in NPNODE list using nnd_xxxx_hntnode and in history buffer using nnd_xxxx_ _hntlist() .

. In nnd_xxxx_hntnode() it passes control further through a switch statement. The case values is N_WT_RSTRT or N_WT_DBREC.

. In the case of N_WT_DBREC, the command execution might have been in suspension state due to non-availability of record, therefore for the opcode DBREC, nnd_xxxx_insrcrd() is called.

In case of spurious message received, the node from npnode list is deleted through nnd_xxxx_delnode and nnd_xxxx_rollback is called.

. If the case value is N_WT_RSTRT the main updation function is called through map table.

. In the nnd_xxxx_checkhist function if the hunt node doesnot succeed then the history bbuffer is searched through nnd_xxxx_hnt list. The parameter for hunting is user_id, job_id and session_id.

. If the search succeeds and status stored in history is N_EXEC_OVER then a result message is prepared and sent to DUP. In other case the message buffer is deallocated.

. If the check in history buffer doesnot succeed and process is not waiting on DBREC then the nnd_xxxx_initcmd function is called.

. In init_cmd function the main updation function is called through map table. For passing the control to the main routine command class is used for distinguishing circuit related and network related commands.

. Main updation consists of following steps:

- do database updations.
- fill result message.
- check for node in history buffer.
- if exists then :
    - set status to EXEC_OVER.
    - fill reult_ptr.
- send result message to destination.
- delete node from the list.

8.2 FUNCTIONAL DESCRIPTION

8.2.1 NPNNDCS_XXXX_ACKILL.C

Function name : nnd_xxxx_acikill()

Purpose : When the message is receveid by NDUP at SU for the database updations, it may not be possible to take the actions immediately due to nonavailability of record or some other reason. In such cases acikill message is sent to DUP to inform it to go to Defferd mode i.e. not to wait for the acknowledgement immediately. The acknowledgement is sent later on throgh OOD.

Actions :

- allocate sspace correspnding to Acikill01.
- prepare message for Acikill using typedef struct Acikill01
- fill command related details using user_id, job_id, session_id, crp_id.
- fill reason, opcode, sender_id.
- send Acikill message pointed to by the n_msg_pointer.

8.2.2 NPNNDCS_XXXX_ARG.C

Function name : nnd_xxxx_arg()

Purpose : This function does the initialisation through switch statement. It is called by nnd_xxxx_enter.c

Action: . In the case of power up, system boot, stable clear, partial initialisation, it calls nnd_xxxx_prcinit() and nnd_xxxx_forever().

. In the case of rollback it calls nnd_xxxx_forever() only.

8.2.3 NPNNDCS_XXXX_CHKHIST()

Function name : nnd_xxxx_chkhist()

Purpose : It checks the history buffer and passes the contol to corresponding command segment.

Action :

. calls nnd_xxxx_hntnode() for hunting npnnode list.

. Depending on the status it passes the control to switch statement for N_WT_DBREC , opode is NDBREC.It calls nnd_xxxx_insrcrd(), otherwise in the case of error it calls nnd_xxxx_delnode() function and rolls back.

. If status is N_WT_RSTRT, it checks the opcode for NCKTCRE or NNETCRE and calls the main updation function through map table.

8.2.4 NDNNDCS_XXXX_COPY STR.C

Function name : nnd_xxxx_copystr()

Purpose : It copies a string of characters from one location to another.

Action :
. Takes copy_from, copy_to and length on input parameters and copies the contents pointed to by these pointers while incrementing these till conunter = length.

8.2.5 NPNNDCS_XXXX_CRENODE.C

Function name : nnd_xxxx_crenode()

Purpose : This function adds a new entry into the linked list called npnode maintained by NDUP.

Action :
. Checks for the value of n_codecount with the code pointed to by the node_ptr.
. In the case of non_availability, it creates the node and fill up the relevant values.

8.2.6 NPNNDCS_XXXX_DELNODE.C

Function name : nnd_xxxx_delnode()

Purpose : This function deletes an entry from ndupq after the completion of command execution.

Action :
. Deallocates the npnode pointed to by the n_temp_ptr -->store_pt. While deallocation it takes care whether the node is an intermediate node or an end node.

8.2.7 NPNNDCS_XXXX_EXTERN.C

This file includes external definitions of global variables and data structures used at NDUP process level.

8.2.8 NDNNDCS_XXXX_FOREVER.C

Function name : nnd_xxxx_forever()

Purpose : This function module receives the message and passes the control to the relevant command segment.

Action :
. Receives the message in an ongoing loop.
. Based on the hdr.opcode pointed to by the n_msg_ptr, it calls nnd_xxxx_chkhist() for NCKTCLE and NNNETCRE opcodes and calls nnd_xxxx_ackto() in the case of N_ACKTO opcode.

8.2.9 NPNNDCS_XXXX_HNTHIST.C

Function name : nnd_xxxx_hnthist()

Purpose : This function hunts in the history buffer for given user_id, job_id, session_id pointed to by the msg_ptr.

Action :

- Takes msgptr as input parameter and hunts the history buffer based on user_id, job_id and session_id.

8.22.10 NPNNDCS_XXXX_HNTNODE.C

Function name : nnd_xxxx_hntnode()

Purpose : This function hunts in npnode list based on the option given to it.

Action :

- It takes option as input parameter
- It passes option to switch statement for hunting using N_HNT_CODE or N_HNT_ID.
- In the case of N_HNT_CODE it checks the opcode for N_ACKTO or NNETCRE or NCKTCE and hunts the node list.
- In the case of N_HNT_ID it hunts using user_id, job_id and session_id.
- Return value is SUCCESS / FAILURE.

8.2.11 NDNNDLS_XXXX_INITCMD.C

Function name : nnd_xxxx_initcmd()

Purpose : This function calls relevant main command function for database updations. The control is passed through a map table.

Action :
. Checks the opcode passed to it for NCKTCRE and NNETCRE.

. In both the cases it calls nnd_xxxx_crenode() and call the function through nsmap table based on command class and command code.

8.2.12 NPNNDCS_XXXX_ENTER.C

Function name : nnd_xxxx_enter()

Purpose : This is the first function called in NDUP. It calls the nnd_xxxx_arg() for further processing.

Action :
. It takes arg as input parameter for the initialisation.

. It calls nnd_xxxx_arg and passes arg as parameter

8.2.13 NPNNDCS_XXXX_PRCINIT.C

Function name : nnd_xxxx_prcinit()

Purpose : This function initialises the local data structures used by NDUP mainly command pointer and npnode.

Action :

- Obtains the n_self_id for the message
- Initialises the history buffer
- Initialises the command pointer table based on command class and command code. This is used as map table for passing the control to main updation funtion.
- Initialises the timer messages

8.2.14 NPNNDCS_XXXX_ROLLBACK

Function name : nnd_xxxx_rollback

Purpose : This function is used as an error reccovery method. In the case of spurious performance it deallocates the buffers engaged and sends the message for partial initialisation.

Action :

- Depending on n_rollvar value it deallocaates the buffer pointed to by n_msg_ptr.

. it calls CDOS system call start() and passes the parameter n_self_id, ROLLED_BACK for reason.

. finally it calls terminate() system call.

8.2.15 NPMNNDCS_XXXX_EXTERN.H

This file includes hash definedd external declartions.

CHAPTER 9

NETWORK AND DATA MANAGEMENT FOR No 7 SS

9.1 GENERAL

The Signslling Network is a part of either local trunk or international network known by it's identity called network_id. In such a network an axchange is assigned a point code called self point code (SPC). The network together with the point code uniquely identifies the exchange.

The exchanges using #7 Signalling System are connected by digital PCM links. A standaed PCM link has 32 multiplexed channels. A group of PCM links is normally selected for one type of seevice, which can be an incoming circuit group or outgoing circuit group or ptiority circuit group ,etc. A circuit group has 4096 circuits. Therefore, 128 pcm links are used for one type of service between two point codes. Any of the 4096 circuit can be used for either voice or signalling purpose. These circuits between two point codes constitute a circuit group set CGS). The circuits in a circuit group is uniquely identified by the circuit identification code(CIC). The CIC is base on the time slot within a 2048 kb/s digital path and the identity of the PCM itself.

The point code with the No 7 SS relation is constituted of originating point code(OPC) and destination point code (DPC).

Two exchanges generally have a single CGS but more than one CGS may be used in the case when the requirement for inter exchange circuits is greater than 4096 or the circuits are routed distinctly on different communication links such as satellite or terestrial link. Each CGS in that case is assigned a unique signalling relation.

9.2 DATA MANAGEMENT AT SU

The data is maintined in the distributed manner in C-DOT DSS. It is distributed over Administarative processor, Base Processor and Signalling Unit. Initially yhe information is downloaded from IOP at AP and later on the modifications are carried out through MMC commands.

The data relevant to SU for signalling and network management is in the form of arrays. The required memory is allocated at the time of initialisation only. This helps in faster accsss and providing real time services. ' C ' language provides the flexibility of data structures, therefore these are used extensively for data storage. These data structures are accessed using " id's " as a key element. Different database function calls are available to operate upon the records in database.

Following is the data information maintained at the AP and BP in the form of data structures. The important fields which get modified are also mentioned here.

/* Data link related typedef structure */

```
Ddl_data:

        dd_sts;          /* status of data link at any time */
        dd_dummy;        /* dummy */
        dd_cic;          /* CIC corr. to data link */
        dd_lnk_typ;      /* specifies terrs. or satellite */
        dd_bm_id;        /* BM where data link is terminated */
```

```
          /* Data link group related typedef */

Ddlg_data:

   dd_cgs_id;       /* cgs id. */
   dd_data_lnk_cnt; /* total no. of data links info. */
   dd_no_of_stdby;  /* no. of standbys to be kept */
   dd_trmn_flg;     /* terminated on same node as cgs or not*/
   dd_trmn_dpc;     /* pc of datalnk termination if flag=YES */
   dd_sgnl_net_id;  /* signalling network id */
   dd_unuse_dl_flg; /* data link to be returned for voice */



     /* link set related typedef structure */



Dls_db_inf:

   dl_act_lnk_cnt;         /* active links */
   dl_dummy;


Dls_com_inf:

   dl_linkset_bundle;
   dl_min_act_links;
   dl_max_act_links;


Nls_su_inf:

   dl_dpc[DMAX_PCS_SERVED];
   dl_link[DMAX_LINKS];
   dl_parent_route;
   dl_min_avl_links;
   dl_max_avl_links;
   dl_recovery_opt;
   dl_pc_served_cnt;


Dls_data:


   dl_ls_db_inf;
   dl_ls_com_inf;
   dl_ls_su_inf;
```

```
    /* circuit group set related typedef structure */


Dsgnl_reln:

   ds_opc; /* originating point code */
   ds_dpc; /* destination point code */

Dcgs_db_data:

   dc_tel_net_id;          /* telecom network identity */
   dc_sgnl_net_id;         /* signalling network identity */
   dc_su_id;              /* SU7 identity */
   dc_ckt_cnt;             /* no. of ckts in the cgs */
   dc_voice_trf;           /* voice traffic ON or OFF */
   dc_cg_cnt;              /* no. of ckt gps in the cgs */
   dc_cg_id[DMAX_CG_CGS];
   dc_sgnl_reln;           /* signalling relation */

Dcgs_iop_inf:

   dc_remx_code;
   dc_remx_name[DMAX_REMX_SIZE];
   dc_dlg_id[DMAX_DLG_CGS];

Dcgs_data:
   dc_cgs_db_data;
   dc_cgs_iop_inf;



           /* Self Point Code related typedef structure */



Dself_pc_data

   ds_self_pc;
   ds_network_id;
   ds_pc_allocated_flg;
```

The important data structures maintained at SU are as follows. The important fields are also given hare.

```
Nconfig_struct:

                sw_config;
                num_of_cgs ;
                num_of_sig_routeset ;
                num_of_linkset_bundle ;
                num_of_linkset ;
                num_of_link ;

Nself_pc_struct:

                pc_num;
                self_pc;
                network_id;
                pc_allocated_flg;

Ncicinfo_struct:

                timer_tag ;
                proc_num;
                msg_adr;
                id_recv_flg;
                iam_flg;
                dst_ter;
                flush_msg_cnt;


Nrouteset_info_struct:

        srs_num;
        hi_prio_rt_tbl[NRT_TBL_SIZE];
        Nhi_prio_rt_tbl_org[NRT_TBL_SIZE];
        lo_prio_rt_tbl[NRT_TBL_SIZE];
        lo_prio_rt_tbl_org[NRT_TBL_SIZE];
        opc;
        dpc;
        rt_tbl[NRT_TBL_SIZE];
        available_routes[NPRIORITY_CLASSES]; /* lo_prio_rts first ! */
        route[NPRIORITY_CLASSES][NMAX_LOAD_SHARERS]; /* lo_prio_rts fi
        route_status[NPRIORITY_CLASSES][NMAX_LOAD_SHARERS];  lo_prio_rt
        current_rt_grp_priority;
```

```
Nlinkset_bundle_info_struct:

        lsb_num;
        stp_user[NMAX_STP_USERS];
        other_end_pc;
        links[NMAX_LINKS];
        act_links;
        max_act_links;
        dl_count;
        datalink[NDL_SEARCH];
        bm_id[NDL_SEARCH];


Nlinkset_info_struct:

        ls_num;
        dpc[NMAX_PCS_SERVED];
        other_end_pc;
        link[NMAX_LINKS];
        parent_route;
        linkset_bundle;
        act_links;
        avl_links;

Nlink_info_struct:

        link_num;
        status;
        timers;
        act_type;
        state;
        datalink;
        parent_linkset;
        terminal;
        seqnl;
        linkset_bundle;


Nterminal_info_struct:

        trml_num;
        type;
        link;
        ter_status;
        empty_chn_req;  /* from level 3 */
        bm_id;
        reset_time[NTIME_STR_LEN];  /* reset time */
        mrecv_cnt;      /* msg recv. since last reset */
```

CHAPTER 10

SOFTWARE RECOVERY FOR NDUP PROCESS

The key factor in system reliability is the ability of the system to continue functioning in the presence of hardware or software faults. This chapter deals with how the system detects and respond to software faults. In general, software reliability is obtained at the expense of system real time performance. These two factors are evaluated for the purpose of implementation and following methods are discussed here -

- defensive checks,
- Maintenance-clear,
- recovery functions,
- use of Macros for C.S. and Database primitives,
- process roll-back routine,etc.

10.1 Defensive Checks :

Defensive checking is defined as a method of providing execution time checks on a unit of code. Simple algorithms are inserte in the code to perform these checks. The use of defensive checks could be costly in terms of real time. For Online commands thes checks are implemented at the level of CEP. Following are the important points for the purpose of defensive checks:

i. Parameter checks on messages received, especially those which are received from other processors.

ii. Parameter checks in the beginning of all high level functions to guarantee the integrity of the input data to lower level functions.

iii. Range checks on indices and pointers which are used to access array elements so that read or writes can only be done within a range.

iv. Checking for O.S. error return codes (from O.S. primitives) on wrong parameters and failures.

v. Introduce time-outs on all critical Receive Messages or phases of the processes.

vi. Build a layer above the protocol level to avoid duplicate messages and lost messages. This could be handled by having a sequence number and formal replys to messages instead of just relying on the protocol level Acks.

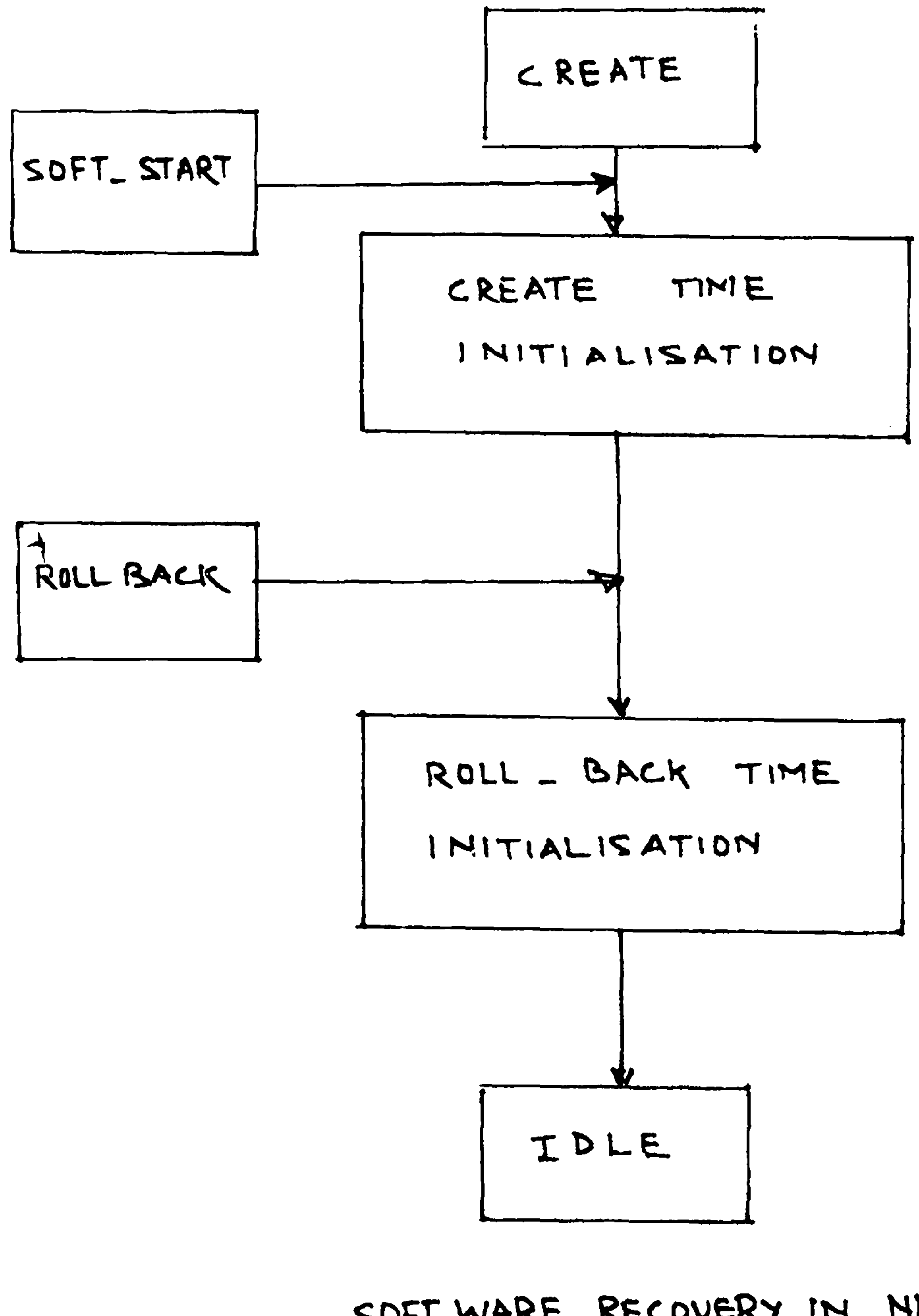
10.2 Recovery strategy :

On occurance of any inconsistent situation the system is returned to a safe state. This is the objective of the recovery strategy in NDUP. Following steps are considered for recovery here:

- process level recovery is initiated ; No other process is effected.
- the relevant process should rollback. In turn it will release all the resources held by it.
- Reboot OS with NO_LOAD option. This will result in the process creation.
- System reboot, start initialization of the unit again.

In NDUP the recovery strategy is as follows:

i. The NDUP at it's single state should provide for receipt of a Maintenance Clear message. On receipt of such a message,the process should abandon the current activity, release all resources held by the process, clear itself up and go to the safe state.



SOFT WARE RECOVERY IN NDUP

ii. The NDUP should be able to handle error situation on its own in the following cases:

- Time-outs on receive message when expecting a message.
- Any error on operator initiated action,operator should be informed of the event by giving a message on the Operator terminal.

iii. Use of Roll-back.

10.3 Roll-back routine description :

Corresponding to NDUP process there is a roll-back routine called nnd_roll_back. It is invoked by the process on receipt of a Maintenance Clear message. The nnd_roll_back may also be called on CPU interrupts after which the process may not be able to proceed.

The Roll-Back consists of the following:

- If the process is in CLEAR STATE then calling the OS primitive RECREATE which will release all the O.S resources held by the process, delete the process and create it again with the same process-id as before. The parameters passed while recreating the process is SOFTSTRT. This ensures releasing of all O.S. resources.
- If the process is not in the clear state the following things are required to be done.

(a) Releasing all the resources held-up by the process.

(b) Informing other related processes, if necessary, that this process is getting restarted.

(c) Restarting this process.

The restart should be done by calling the O.S. primitive "Start" followed by "Trmt" (Start request is queued and is honoured only after the process has been terminated. While doing Start, a parameter (ROLLED_BACK) is passed to the process to indicate that it should skip the create time initialization.

The Roll-back function has one parameter, the pointer the PDB of the process. All the necessary information about the resources held by the process and any information about related communicating processes is either stored in PDB or in a global area. The roll-back accesses these areas and releases the resoures and informs other processes, if necessary.

Roll-back ~~s~~$\overset{s}{\cancel{p}}$eudocode:

```
BEGIN nnd_roll_back (pdb_ptr)

        IF (counter EQ 0) /* Same counter as used for clear state
            RECREATE(SOFTSTRT);
        ELSE
```

```
            IF(called from initiator or terminator leg)

                counter = counter - 1;
                IF(counter LT 0)
                    counter = 0;
                ENDIF

                IF(counter EQ 0)
                    SETCLR(XXXPUP)
                ENDIF

            ENDIF
            Release resources held up for the current activity;

            /* Resources include O.S. resources like
            /*  ALOC buffers
            /*  MALOC buffers
            /*  timers (MSGORD, MSGCYC)
            Undo changes in certain global data structures;
            If required, inform related processes of failure;
            /* The related process can repeat
            /* the message which started the activity
            START (NOCOOR+INHPTY, pid, ROLLED_BACK, 0, 0, 0, 0);
            TRMT  ();

        ENDIF

END nnd_roll_back
```

* The initial argument to the main of the process can have one of the following values:

POWRON	0	For power-on initialization -- System is coming up for first time
SYSBOT	1	For System Bootup -- Complete reload of all code and data is done
STBCLR	3	For process creation only
SOFTSTRT	5	For software restart -- Data is reloaded
ROLLED_BACK	6	When the process is rolled back

Clear State :

Clear state as used by PUP and Maintenance has significance only for eternal processes. For single state eternal processes, clear state is that idle state when the process does not have any knowledge of past events, i.e., the state is same as that when it originated.

10.4 MODULE RECOVERY PROCESS

A process MRP would exist to handle messages from the other maintenance processes. These processes can trigger MRP to initiate recovery and then give up control. MRP on request, can do sytem recovery, kill processes etc.

10.5 DATABASE FAILURES

As far as possible these would be taken care of by the process recovery routine. Depending on the state of the process and the message available in the pdb the process should be able to undo the changes in the database done so far. If all of it cannot be undone then audits can be initiated. In both the case process roll back would be called.

CHAPTER 11

NETWORKING FEATURES OF SS 7

General

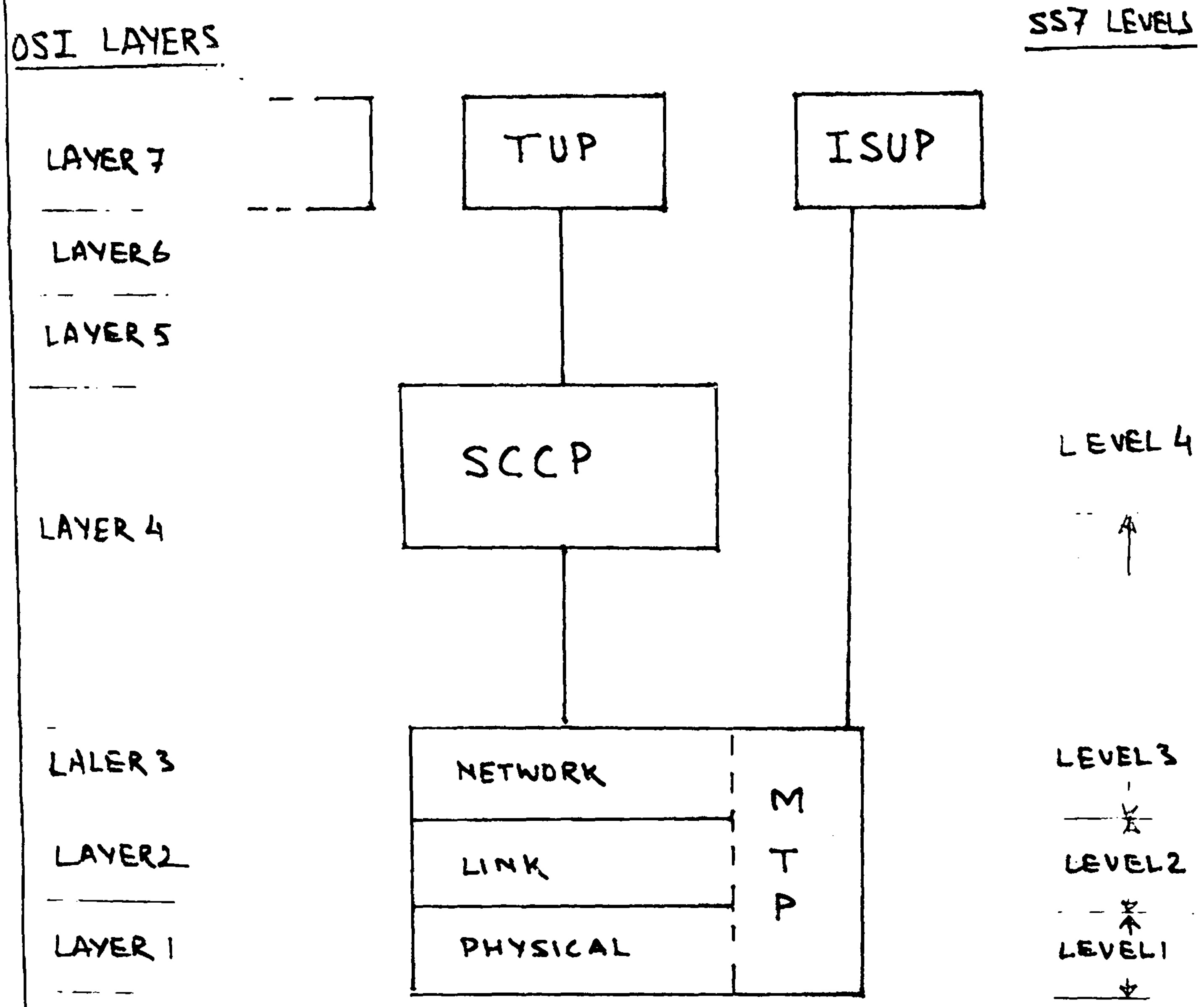
The Signalling System No 7 network architecture and its introduction constitutes an important step in providing useful services to a subscriber. These services mainly of credit card, call forwarding, mobile phones, introduction of Intigrated Services Digital Network etc., may be possible with the implementation of SS 7 network.

SS 7 Layer Architecture

SS 7 is projected to have four layers. These are as follows:

LEVEL 1 : Signalling data link functions.

This level is concerned with providing a two way communication path between two adjacent signallig points(SP).



MTP	:	MESSAGE TRANSFER PART
TUP	:	TELEPHONE USER PART
ISUP	:	ISDN USER PART
SCCP	:	SIGNALLING CONNECTION CONTROL PART

SS 7 ARCHITECTURE AND OST MODEL

LEVEL 2 : Signalling link functions.

These functions are also called link control functions.These support reliable delivery of messages between two adjacent SPs and correspond to layer 2 of OSI datalink layer.

LEVEL 3 : Signalling network functions.

Level 3 is also called common transfer function. This level enables data and control informations of the signalling messages to be exchanged between two non adjacent SPs.

LEVEL 4 : User or Application part.

This level includes user parts such as Telephone User Part(TUP), ISDN User Part(ISUP), Signalling Connection Control Part(SCCP) etc.

The lower three levels of SS 7 are collctively called Message Transfer Part(MTP). The MTP provides datagram services between two SPs.

Comparision of SS 7 Architecture with OSI model

The four layer architecture of SS 7 is functionally analogous to layered model of OSI. However, the analogy fails on many accounts.

The signalling data link and signalling link functions correspond directly to the OSI physical and data lnk layers . The signalling network functions are divided int two categories of signalling message handling and the signalling network management functions. THe former category corresponds to the network layer of OSI model, though the network management functions i.e.,link management, route management, and traffic management makes it different from the OSI model. The management functions include selection, activation/ deactivatin, switch-over etc. of data links.

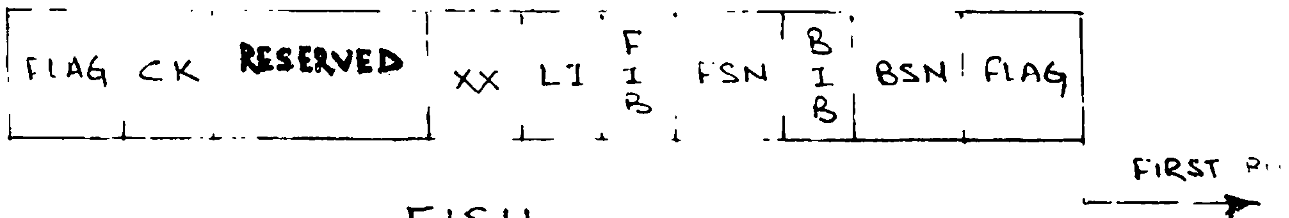
The fourth level of Signalling Connection Control Part(SCCP) corresponds to the transport layer of OSI model. It provides basic and sequenced connectionless services and three classses of connection oriented services viz. basic, flow control and error recovery with flow control.

The upper layer provides user and application service parts which map the upper three layers of OSI model.

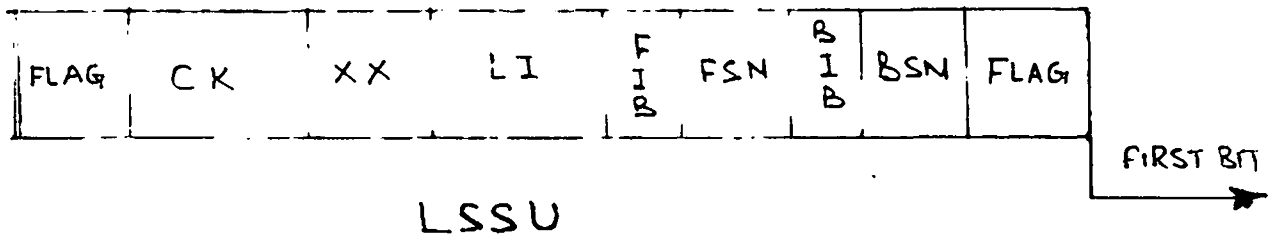
Signal Units

For the implementation of SS 7 messages signal units are used. These signal units are of three types.

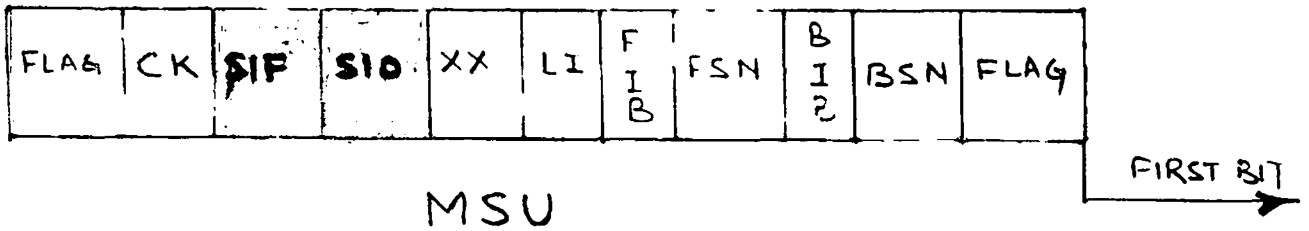
- Fill in Signal Unit (FISU)
- Link Status Signal unit (LSSU)
- Message Signal Unit (MSU)



FISU



LSSU



MSU

FIELD		BIT WIDTH
FLAG :	FLAG -	8
BSN :	BACKWARD SEQUENCE NO. -	7
BIB :	BACKWARD INDICATOR BIT -	1
FSN :	FORWARD SEQUENCE NO. -	7
FIB :	FORWARD INFORMATION BIT -	1
LI :	LENGTH INDICATOR -	6
XX :	NOT USED -	2
RESERVED :	-	n×8
SIO :	SERVICE INFO. OCTET -	8
SIF :	SIGNALLING INFO. FIELD -	$2 \leqslant SIF \geqslant 272$ bytes
CK :	CHECK BITS -	16

SU FORMAT

FISU . FISUs are transmitted on the line when no other signal unit to be sent is pending. These signal units naintain the connection, although no information is transmitted.

LSSU : LSSU is same as FISU but it also contains status information such as establishing a connection, checking normal alignment, lost alignment etc.

MSU : MSUs contain all the information transmitted on the line.The formats of these are shown in he figure.

Conclusion

The four layer model of SS 7 though fully maps the standard OSI model, the usefulnass is still limited for the networking use The difference lies in the application context. The OSI model was formulated to enable end users to interconnect in a standard manner. The user remains external to a communication subnrtwork. Whereas, the Signalling System No 7 performs to create a communication subnetwork for a 'network end user'

Certain fields in the SS 7 packets are shared among different levels. This adds efficiency but makes it impossible to replace one protocol with another, although this rigidity reflects its existence as a self-contained signalling architecture. The

Service information Octet of MSU, which identifies the type of message, its priority level, network identity etc. is a part of level 2 but it is useful for level 3 and level 4 also. Similarly the Signalling link selection(SLS), Originating Pont cCode(OPC), and Destination Point Code(DPC) are used at level 3 and are equally important for the SCCP for transport functions. This property is a unique feature of SS 7.

The SS 7 also has highly developed reliability features. In the Telephone network the accssibility to the services and short time response is very important which may not be so important in a normal communication network.

The SS 7 is capable of adopting to the failures and service degradations dynamicaly by communicating to adjacent SPs. It has a provision of declaring "link failure" and route the calls on different routes. It can also provide better services in degraded situations of congestion.

This way it can be seen that the SS 7 , though does not resembles completely with OSI model but it has efficient networking capabilities and can be recognised as a computer communcation network for a few applications.

APPENDIX - I

I. NAMING CONVENTIONS

This part of the Appendix describes the conventions followed in naming the messages and also the file structures, while designing a man machine command.

Message Naming convention :

All man machine commands are divided into classes. Normally command having some logical utility will be in the same class such as create_cgs and allocate_cic command will be in the same class.

Message Name

subsystem type	- 1 char	
class name	- 3 char	
category	- 3 char	
type	- 1 char	(Not present in all msg)
subfield ident.	- 2 char	(optional)

Subsystem type

All the commands related to #7 will have subsystem type as N

Following class id have been identified -

1. ckt - for all ckt related command (call processing)
2. net - for all network management related command

Category

1. cre - for all create type of commands
2. mod - for all modify type of commands
3. dis - for all display type of commands

Type

This field is specified to identify the direction of the flow of the messages

1. I for all input messages to cep form crp
2. O for all output messages from cep to crp
3. R for all messages returned by ACI to cep

Messages from CEP to ACI will not have this field present.

Subfield

This identifies the sub field of the message.

Note :

The direction type field :

CRP -> CEP	type is I
CEP -> ACI	type is absent
ACI -> CEP	type is R
CEP -> CRP	type is O
ACI -> NDUP	type is absent
NDUP-> ACI	type is R

Example -

Message from crp to cep for creating a cgs - Ncktcrei01

File Naming conventions

File names for Man Machine command will follow the conventions given below.

File Naming format

sub system name	:	1 char
CEP process code	:	4 char
hyphen char(_)	:	1 char
file identifier	:	4 char

Process Code Naming conventions

Process codes for the commands are 4 char long , only first char has a convention other 3 identifies the command

R	- remove type of command
C	- create type of command
D	- display type of command
M	- modify type of command
G	- general purpose command

Function naming conventions

There are two types of functions -

a. Library Functions

b. Process level Functions

Library functions will use nl_ as a prefix for a function name , support functions of these function will also use this prefix

Process level functions will use following conventions -

Sub system name	- 1 char
Process name	- 4 char
hyphen char(_)	- 1 char
function identifier	

Example: Following process codes are fixed as -

Command name	Process code	Command code
CRE_CGS	CCGS	CGS
CRE_DLG	CDLG	DLG
CRE_LSB	CLSB	LSB
CRE_SLS	CSLS	SLS
ALOC_CIC	CCIC	CIC
CRE_SRS	CSRS	SRS
CRE_SPC	CSPC	SPC

Following are the System Calls used in NDUP. These calls are in the form Macros written for CDOT Distributed Operating System (CDOS).

File name : NSNXXCH_MACRO.H

```
#if     _NDUP
/*.....................................................*/

#define         AMAC_SEND(pid, ptr)     {\
    _send(pid, ptr);\
    ptr = NULL; \
    }

/*.....................................................*/

#define         AMAC_ALOC(ptr, size)     {\
    while ((ptr = _aloc(size)) == NULL) \
        _pause(SEC, ONE); \
    }

/*.....................................................*/

#define         AMAC_MALOC(ptr, size)     {\
    while ((ptr = _maloc(size)) == NULL) \
        _pause(SEC, ONE); \
    }

/*.....................................................*/

#define         AMAC_DALOC(ptr)     {\
    _daloc(ptr); \
    ptr = NULL; \
    }

/*.....................................................*/
#endif
```

Following is the list of function calls used for NDUP process.

Function name	File name
nnd_xxxx_arg()	NPNNDCS_XXXX_ARG.C
nnd_xxxx_ackill()	NPNNDCS_XXXX_ACIKILL.C
nnd_xxxx_chkhist()	NPNNDCS_XXXX_CHKHIST.C
nnd_xxxx_copystr()	NPNNDCS_XXXX_COPYSTR.C
nnd_xxxx_crenode()	NPNNDCS_XXXX_CRENODE.C
nnd_xxxx_delnode()	NPNNDCS_XXXX_DELNODE.C
nnd_xxxx_dselect()	NPNNDCS_XXXX_DSELECT.C
nnd_xxxx_insrcrd()	NPNNDCS_XXXX_INSRCRD.C
nnd_xxxx_forever()	NPNNDCS_XXXX_FOREVER.C
nnd_xxxx_hnthist()	NPNNDCS_XXXX_HNTHIST.C
nnd_xxxx_hntnode()	NPNNDCS_XXXX_HNTNODE.C
nnd_xxxx_initcmd()	NPNNDCS_XXXX_INITCMD.C
nnd_xxxx_prcinit()	NPNNDCS_XXXX_PRCINIT.C
nnd_xxxx_rollback()	NPNNDCS_XXXX_ROLLBACK.C

File Name : NPNNDCS_XXXX_ARG.C

```
Void nnd_xxxx_arg(arg)
    Uint    arg;

    {

    switch (arg)
        {
            case POWRON:
                    /* system is coming up for first time */
            case SYSBOT:
                    /* system restart due to s/w fail */
            case STBCLR:
                    /* stable clear */
            case PARTINIT:
                    /* partial init */
            case SOFTSTRT:
                    /* s/w restart */
                nnd_xxxx_prcinit();
                    /* calling the function which does process initialisati
                    /* fall thru */
            case ROLLED_BACK:
                    /* process rolled back */
                nnd_xxxx_forever();
                    /* the actual processing of messages */
                break;
            default:
                _recre(SOFTSTRT);
             /* switch (arg) completes */
```

File Name : NPNNDCS_XXXX_CHKHIST.C

```
Void    nnd_xxxx_chkhist()


  {
    Rshort      opcode;
    Ulong       ret_val;
    Rshort      size;
```

```
opcode = n_msg_ptr->hdr.opcode;
ret_val = nnd_xxxx_hntnode(N_HNT_ID);
if (ret_val == N_RET_SUC)
  {
    switch ((Int)n_temp_ptr->status)
      {
        case N_WT_DBREC:
             if (opcode == NDBREC)
               {
                 if (((Ndbrec01 *)n_msg_ptr)->dberr_cnt
                                 == n_temp_ptr->dberr_cnt)
                     nnd_xxxx_insrcrd();
               }
             else
               {
                 Nreqrcrd01  *reqrcrd_ptr;

                 if (n_temp_ptr->reqrcrd_ptr == NULL)
                   {
                     nnd_xxxx_delnode();
                     nnd_xxxx_rollback(NULL);
                   }
               }
             AMAC_DALOC(n_msg_ptr);
             break;

        case N_WT_RSTRT:
             AMAC_DALOC(n_msg_ptr);
             if (opcode != NDBREC)
               {
                 Ushort      cmnd_cls;

                 switch (opcode)
                   {
                     case NCKTCRE:
                          cmnd_cls = N_CKTCRE_CLS;
                          break;

                     case NNETCRE:
                          cmnd_cls = N_NETCRE_CLS;
                          break;
                   }
                 (*nsmap[n_cmnd_ptr[cmnd_cls][n_temp_ptr->hdr.subfield]]
               }
             break;
        default:
             AMAC_DALOC(n_msg_ptr);
      }
  }   /*  end_if  */
 else
  {
    ret_val = nnd_xxxx_hnthist(n_msg_ptr);
    if (ret_val == N_RET_SUC)
      {
        if (nnd_hist_ptr->cmnd_stat == N_EXEC_OVER)
          {
            Rslt_hdr      *dum_ptr;
```

```
                    Uchar           *rslt_ptr;
                    Uchar           *tobm_ptr;

                    dum_ptr = (Rslt_hdr *)(nnd_hist_ptr->rslt_ptr);
                    if (dum_ptr != NULL)
                      {
                        AMAC_MALOC(rslt_ptr, (size = dum_ptr->rslthdr.size));
                        nnd_xxxx_copystr(dum_ptr, rslt_ptr, size);
                        _send(nnd_hist_ptr->dest_id, rslt_ptr);
                      }
                  }
                AMAC_DALOC(n_msg_ptr);
              }
            else
              {
                if (opcode != NDBREC)
                  {
                    Cmd_hdr         *dum_ptr;

                    dum_ptr = (Cmd_hdr *)n_msg_ptr;
                    nnd_hist_ptr                 = &nnd_hist_buf[nnd_hist_index+
                    nnd_hist_ptr->session_id     = dum_ptr->cmdhdr.session_id;
                    nnd_hist_ptr->user_id        = dum_ptr->cmdhdr.user_id;
                    nnd_hist_ptr->job_id         = dum_ptr->cmdhdr.job_id;
                    nnd_hist_ptr->dest_id        = dum_ptr->hdr.sndr.id;
                    if (nnd_hist_ptr->rslt_ptr != NULL)
                      {
                        AMAC_DALOC(nnd_hist_ptr->rslt_ptr);
                        nnd_hist_ptr->cmnd_stat = N_WT_DBREC;
                                   /* some value which is not N_EXEC_OVER */
                      }
                    if (nnd_hist_index == N_MAX_HIST)
                      nnd_hist_index = 0;
                    nnd_xxxx_initcmd();
                  }
                else
                  {
                    AMAC_DALOC(n_msg_ptr);
                  }
              }
        }
    }
```

File Name : NPNNDCS_XXXX_COPYSTR.C

```
Void copystr(copy_from, copy_to, length)

    register    Uchar   *copy_from;
    register    Uchar   *copy_to;
    register    Ushort  length;

  {

    Rint    i;

    for (i = 0; i < length; i++)
        *(copy_to++) = *(copy_from++);
  }
```

File Name : NPNNDCS_XXXX_CRENODE.C

```
Void    nnd_xxxx_crenode()

  {
    register    struct  npnode  *node_ptr;
    register    Ushort  opcode;
    Uint        check = FAILURE;
    Uchar       time_buf[20];
    Ulong       hours, mints, secs;


    n_clr_counter++;            /* incrementing jobs to be done counter */

    while (check != SUCCESS)
      {
        node_ptr = n_perm_ptr->right;
        while ((node_ptr != NULL) &&
               (n_codecount != node_ptr->code))
            node_ptr = node_ptr->right;

        if (node_ptr != NULL)                   /* code is already existing */
          {
            n_codecount++;
            if (n_codecount >= N_MAX_COD)               /* testing for the limi
                n_codecount = 1;
          }
        else                                            /* code is unique */
            check = SUCCESS;
      }                                                 /* end of "WHILE" check

    AMAC_ALOC(node_ptr, sizeof(struct npnode));

    n_last_ptr->right = node_ptr;
    node_ptr->left    = n_last_ptr;
    n_last_ptr        = node_ptr;
    n_temp_ptr        = node_ptr;

    node_ptr->right       = NULL;
    node_ptr->reqrcrd_ptr = NULL;
```

```
      node_ptr->acikill_ptr = NULL;
      node_ptr->status      = NULL;
      node_ptr->code        = nnd_hist_ptr->code
                            = n_codecount++;
      node_ptr->dberr_cnt   = node_ptr->retry_cnt = 0;

      if (n_codecount >= N_MAX_COD)
          n_codecount = 1;

      opcode = n_msg_ptr->hdr.opcode;

      nnd_xxxx_copystr(n_msg_ptr,&(node_ptr->hdr),(sizeof(Hdr) + sizeof(Cmdhc

      AMAC_ALOC(node_ptr->store_ptr, node_ptr->cmdhdr.size);
      nnd_xxxx_copystr(n_msg_ptr, node_ptr->store_ptr, node_ptr->cmdhdr.size)

      time(time_buf);
      hours = ((time_buf[12] - '0') * 10 + (time_buf[13] - '0'));
      mints = ((time_buf[15] - '0') * 10 + (time_buf[16] - '0'));
      secs  = ((time_buf[18] - '0') * 10 + (time_buf[19] - '0'));

      node_ptr->cre_time = secs + mints * 60 + hours * 60 * 60;

  }
```

File Name : NPNNDCS_XXXX_DELNODE.C

```
Void nnd_xxxx_delnode()

   {

     AMAC_DALOC(n_temp_ptr->store_ptr);

     if (n_temp_ptr->reqrcrd_ptr != NULL)
         AMAC_DALOC(n_temp_ptr->reqrcrd_ptr);

     if ((n_temp_ptr->right != NULL) &&
         (n_temp_ptr->left  != NULL))
                         /* if a_temp_ptr is pointing to some intermediate n
       {
         n_temp_ptr->left->right = n_temp_ptr->right;
         n_temp_ptr->right->left = n_temp_ptr->left;
       }
     else                              /* a_temp_ptr pointing to the last node *
       {
         n_last_ptr        = n_temp_ptr->left;
         n_last_ptr->right = NULL;
       }

     if (n_temp_ptr != n_perm_ptr)
         AMAC_DALOC(n_temp_ptr);
```

```
    n_clr_counter--;
    if (n_clr_counter < 0)
        n_clr_counter = 0;
 /* if (n_clr_counter == 0)    SETCLR(BDUPPUP); */

   }
```

File Name : NPNNDCS_XXXX_FOREVER.C

```
Void    nnd_xxxx_forever()

  {
    for (;;)
      {
        /*      if (n_clr_counter ==0);      PUP requirement     */

        n_msg_ptr = _recv();
        switch (n_msg_ptr->hdr.opcode)
          {
          case  NCKTCRE :
          case  NNETCRE :
                nnd_xxxx_chkhist();
                break;

          case  N_ACKTO:                  /*  time out trigger     */
                nnd_xxxx_ackto();
                break;

          default:
                AMAC_DALOC(n_msg_ptr);
          }                          /*  end_switch  */
      }                              /*  end_for()   */
  }                                  /*  end_forever */
```

File Name : NPNNDCS_XXXX_HNTHIST.C

```
Ulong   nnd_xxxx_hnthist(msg_ptr)

    Cmd_hdr      *msg_ptr;

  {
    Rint        i;

    nnd_hist_ptr = nnd_hist_buf;
    for (i = 0; i < N_MAX_HIST; i++)
      {
        if ((msg_ptr->cmdhdr.session_id == nnd_hist_ptr->session_id) &&
            (msg_ptr->cmdhdr.user_id    == nnd_hist_ptr->user_id)    &&
            (msg_ptr->cmdhdr.job_id     == nnd_hist_ptr->job_id))
            break;

        nnd_hist_ptr++;
      }
    if (i >= N_MAX_HIST)
        return(N_RET_FAIL);
    else
        return(N_RET_SUC);
  }
```

File Name : NPNNDCS_XXXX_HNTNODE.C

```
Ulong   nnd_xxxx_hntnode(option)

    Uchar      option;

  {
    Ulong      ret_val = N_RET_FAIL;
    Uchar      code_byte;

    switch ((Ulong)option)
      {
        case N_HNT_CODE:                              /* hunting node using code
          {
            switch (n_msg_ptr->hdr.opcode)
              {
                case N_ACKTO:
                    code_byte = n_msg_ptr->n_ackto01.code;
                    break;
                case NNETCRE:
                case NCKTCRE:
                    code_byte = n_codecount;
                    break;
                default:
                    ;
              }
            n_temp_ptr = n_perm_ptr->right;
            while ((n_temp_ptr != NULL) && (code_byte != n_temp_ptr->code))
                n_temp_ptr = n_temp_ptr->right;
            if (n_temp_ptr != NULL)
                ret_val = N_RET_SUC;
          }
          break;

        case N_HNT_ID:                     /* hunting node using job,user ids */
          {
            Cmd_hdr    *msg_ptr;

            msg_ptr = (Cmd_hdr *)n_msg_ptr;
            n_temp_ptr = n_perm_ptr->right;

            while ( (n_temp_ptr != NULL) &&
                ((msg_ptr->cmdhdr.session_id != n_temp_ptr->cmdhdr.session i
                 (msg_ptr->cmdhdr.user_id    != n_temp_ptr->cmdhdr.user id)
                 (msg_ptr->cmdhdr.job_id     != n_temp_ptr->cmdhdr.job_id )
              n_temp_ptr = n_temp_ptr->right;

            if (n_temp_ptr != NULL)
                ret_val = N_RET_SUC;
          }
          break;

        default:
                ;
      }

    return(ret_val);
  }
```

File Name : NPNNDCS_XXXX_INITCMD.C

```
Void    nnd_xxxx_initcmd()

  {

    if ( n_msg_ptr->hdr.sndr.id == n_dup_id)
      {
        switch (n_msg_ptr->hdr.opcode)
          {
            case NCKTCRE:
              {
                nnd_xxxx_crenode();
                (*nsmap[n_cmnd_ptr[N_CKTCRE_CLS][n_temp_ptr->hdr.subfield]]
                break;
              }
            case NNETCRE:
              {
                nnd_xxxx_crenode();
                (*nsmap[n_cmnd_ptr[N_NETCRE_CLS][n_temp_ptr->hdr.subfield]]
                break;
            }
          default:
               AMAC_DALOC(n_msg_ptr);
        } /*  end_switch      */
      }
    else
       AMAC_DALOC(n_msg_ptr);
  } /*  end_main        */
```

File Name : NPNNDCS_XXXX_ENTER.C

```
Process enter(arg)
Uint    arg;
/* when process is created this "arg" is passed to the process */
    {
    nnd_xxxx_arg(arg);
    }    /* end of main */
```

File Name : NPNNDCS_XXXX_PRCINIT.C

```
Void nnd_xxxx_prcinit()

 {
    register    Nndhist     *histbuf_ptr;
    register    N_ackto01   *time_ptr;
    Rint        i;

    M_GETPID(n_self_id);

            /*  following statements are for history buffer initialisation

    histbuf_ptr = nnd_hist_buf;
    for (i=0; i < N_MAX_HIST; i++)
        {
        histbuf_ptr->user_id    = 0;
        histbuf_ptr->session_id = histbuf_ptr->job_id   = 0;
        histbuf_ptr->rslt_ptr   = NULL;
        histbuf_ptr->cmnd_stat  = N_WT_DBREC;
        histbuf_ptr++;
        }

        nnd_hist_index = 0;
        /*  following statements are for command pointer initialisatio

    n_cmnd_ptr[N_CKTCRE_CLS][N_CRE_SPC] = MAP_CRE_SPC;
    n_cmnd_ptr[N_CKTCRE_CLS][N_CRE_CGS] = MAP_CRE_CGS;
    n_cmnd_ptr[N_CKTCRE_CLS][N_MOD_CGS] = MAP_MOD_CGS;
    n_cmnd_ptr[N_CKTCRE_CLS][N_DEL_CGS] = MAP_DEL_CGS;
    n_cmnd_ptr[N_CKTCRE_CLS][N_ALC_CIC] = MAP_ALC_CIC;
    n_cmnd_ptr[N_CKTCRE_CLS][N_DLC_CIC] = MAP_DLC_CIC;

    n_cmnd_ptr[N_NETCRE_CLS][N_CRE_SLS] = MAP_CRE_SLS ;
    n_cmnd_ptr[N_NETCRE_CLS][N_MOD_SLS] = MAP_MOD_SLS ;
    n_cmnd_ptr[N_NETCRE_CLS][N_DEL_SLS] = MAP_DEL_SLS ;
    n_cmnd_ptr[N_NETCRE_CLS][N_CRE_LSB] = MAP_CRE_LSB;
    n_cmnd_ptr[N_NETCRE_CLS][N_MOD_LSB] = MAP_MOD_LSB;
    n_cmnd_ptr[N_NETCRE_CLS][N_DEL_LSB] = MAP_DEL_LSB;
    n_cmnd_ptr[N_NETCRE_CLS][N_CRE_SRS] = MAP_CRE_SRS;
    n_cmnd_ptr[N_NETCRE_CLS][N_MOD_SRS] = MAP_MOD_SRS;
    n_cmnd_ptr[N_NETCRE_CLS][N_DEL_SRS] = MAP_DEL_SRS;

        /*  following statements are for time buffer initialisation   *

    n_time_buf[N_CYCLIC_TIMER] = 5 ;



        /*  following statements are for linked list npnode   */


n_clr_counter = 0;              /*  tells how many jobs to done */

n_codecount = 1;                 /* key to the linked list node */

AMAC_ALOC(n_perm_ptr, sizeof(struct npnode));  /* header node of the li

n_perm_ptr->left = n_perm_ptr->right = NULL;
n_temp_ptr = n_last_ptr = n_perm_ptr;
    /* n_perm_ptr always points to the header node */
    /* n_last_ptr always points to the last node in the list */
    /* n_temp_ptr poits to the current processed node */
    /* n_codecount gives the code with which a node will be created */
    /* and that code is the key to identify the node */

AMAC_ALOC(n_ptr_dest_id, sizeof(Pid));

AMAC_MALOC(time_ptr, sizeof(N_ackto01));

time_ptr->hdr.opcode     = N_ACKTO;
time_ptr->hdr.subfield   = ONE;
time_ptr->hdr.sndr.id    = n_self_id;
time_ptr->timer_type     = N_CYCLIC_TIMER;
```

File Name : NPNNDCS_XXXX_ROLLBACK.C

```
Void     nnd_xxxx_rollback(pdb_ptr)

    char     *pdb_ptr;

  {
    if (n_rollvar == 0)
      {
        if (n_msg_ptr != NULL)
            AMAC_DALOC(n_msg_ptr);
        _start(NOCOOR + INHPTY, n_self_id, ROLLED_BACK, 0, 0, 0, 0);
        _trmt();
      }
  }
```

File Name : NPNNDCS_XXXX_ACIKILL.C

```
Void     nnd_xxxx_acikill(reason)
    Uchar    reason;

  {
    register    Acikill01       *acikill_ptr;
    Rshort      size;

    size = sizeof(Acikill01);
    if ((n_temp_ptr->acikill_ptr == NULL) ||
                (reason != (Uchar)(n_temp_ptr->acikill_ptr)))
      {
        AMAC_MALOC(acikill_ptr, size);

        n_temp_ptr->acikill_ptr    = reason;

        acikill_ptr->hdr.opcode    = ACIKILL;
        acikill_ptr->hdr.subfield  = ONE;
        acikill_ptr->hdr.sndr.id   = n_self_id;

        acikill_ptr->cmdhdr.session_id = n_temp_ptr->cmdhdr.session_id;
        acikill_ptr->cmdhdr.user_id    = n_temp_ptr->cmdhdr.user_id;
        acikill_ptr->cmdhdr.job_id     = n_temp_ptr->cmdhdr.job_id;
        acikill_ptr->cmdhdr.crp_id     = n_temp_ptr>cmdhdr.crp_id;

        acikill_ptr->cmdhdr.status_byte = n_temp_ptr->cmdhdr.status_byte;
        acikill_ptr->cmdhdr.output_dev  = n_temp_ptr->cmdhdr.output_dev;
        acikill_ptr->cmdhdr.size        = size;
        acikill_ptr->reason             = reason;

        _send(n_temp_ptr->hdr.sndr.id, acikill_ptr);
      }
```

SUPPLEMENTARY FILES USED FOR NDUP

Following files have been used to support the NDUP process in No 7 SU.

i. NPNNDCH_XXXX_EXTERN.H :

This file includes extern definitions used for NDUP at process level.

ii. NPNNDCH_XXXX_PDEF.H :

This has process definition of NDUP only.

iii. NPNNDCS_XXXX_EXTERN.C :

This has external declaretions of variables at process level.

iv. NSNXXCH_CHASH.H :

This has hash defined constants used for No 7 Software defined at higher level.

v. NSNXXCH_SYSTEM.H :

This is system file which includes relevant files from " toolrel:, dbrel:, mtrel: " directories.

vi. NSNXXCH_TYPDEF.H :

This has typ definitions of data structures used for NDUP process.

APPENDIX IV

REFERENCES

1. Specification for National application of CCITT No 7 Common Channel Signalling System (Part I).
2. CCITT Red Book VI.
3. Overview of CDOT & DSS Projects.
4 ISDN: Architecture and Protocols
: Dr. Maurizio Decina and Dr. A. Roveri.
5. " Overvview of Signalling System No 7 "
: G. Gary Schlanger.