(1348)

MAN MACHINE 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

K. M. K. CHAKRAVARTHY

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

C E R T I F I C A T E

This is to certify that Mr. K.M.K.Chakravarthy who submitted this project report entitled "**Man-Machine Interface for #7 Signalling System**" for his M.Tech degree worked under my guidance and supervision.

This work has not been submitted anywhere for awarding any other degree or diploma.

Shankarnarayan
Mr. S.Shankarnarayan,
Manager
CCITT #7 group
New Delhi

[illegible]
(Dr.K.K.Nambiar)
School of Computer &
Systems Sciences,
J.N.U., New Delhi.

[illegible]
(Prof.N.P.Mukherjee)
Dean
School of Computer &
Systems Sciences
J.N.U, New Delhi

CONTENTS

ACKNOWLEDGEMENTS

I am grateful to Mr. S. Shankaranarayan, Manager CCITT #7 group C-DOT for allowing me to do the project work at C-DOT New Delhi.

I would like to thank my guides Dr. K.K. Nambiar at J.N.U and Mr. Jalaj Swami, Engineer #7 group, at C-DOT for their constant encouragement and guidance throughout the project work. I am also grateful to Mr. A.G. Dixit, Senior group leader, Admin group for his kind help.

I would also like to thank Mr.Anant Ghotkar, Group leader, Engineers Mr.K.K.Das, Mr. Gulshan Kumar, #7 group for their helpful suggestions during my project work.

Finally I thank all engineers of #7 group for making my stay here a memorable & enjoyable one.

K.M.K.Chakravarthy.

P R E F A C E

This is a project report on the Design & Implementation of four Man-Machine commands for implementation of #7 Signalling System in C-DOT DSS. The commands implemented will make signalling terminals available to the exchange and makes the corresponding database updations at IOP. The software accepts operator's command, does corresponding updation at IOP and sends message to update the same within the exchange.

This work has been divided into four parts. Chapter 1 is the Overview of C-DOT DSS architecture. Chapter 2 deals with the details about the #7 Signalling System and its implementation in C-DOT DSS . Chapter 3 is the project overview Chapter 4 is a detail(ED) of the Man-Machine Commands Design and their Implementation.

K.M.K.Chakravarthy.

CHAPTER ONE

OVERVIEW OF C-DOT DSS ARCHITECTURE

C-DOT digital switching system (DSS) has a modular, distributed architecture. It basically consists of four elements.

Administrative module :

The Administration module provides system level functions. These include call processing support such as directory to equipment number translation, software recovery and overall initialization. It also provides interfaces to mass memory and man-machine communication via input output processor (IOP).

Base module :

The base module is the basic growth unit. All the subscriber lines like ordinary, PBX, coin collection box (CCB), line concentrator (LC) etc. and the associated services like tones, announcements and signalling equipment are terminate on this unit.

The base module consists of a time switch which is a time slot interchanger, a base module switch (BMS), Base processor (BP), and memory units with appropriate interfaces to the processor. Presently a Signalling unit based on CCITT #7 specifications is being added to the BM. This SU is designed to handle all the internal protocols along with the #7 protocols.

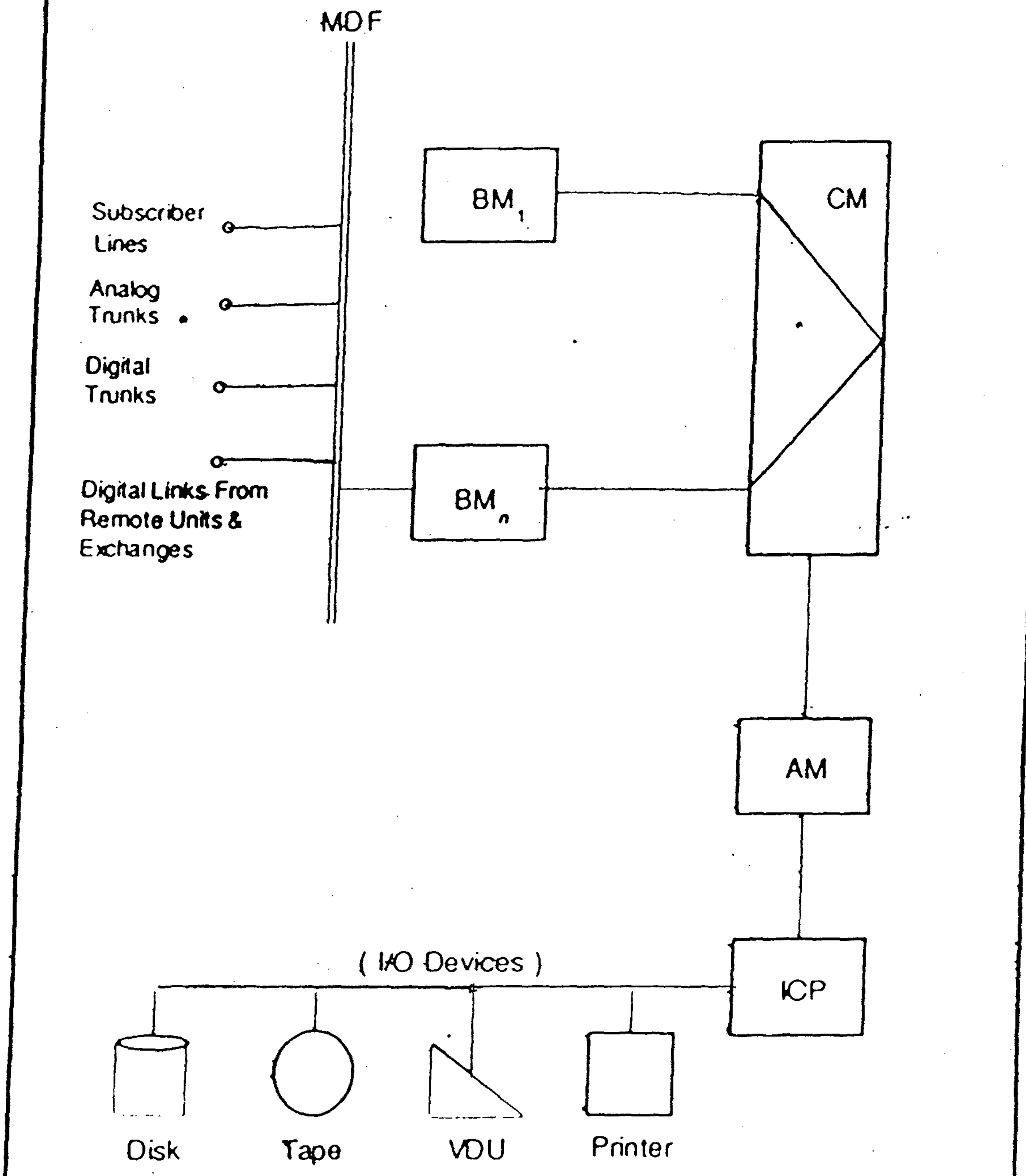
Central module :

The basic function of the Central module is to provide connectivity between BMs, between AM and BMs and between AM and CM itself.

The CM consists of a space switch (SS) controlled by a Space switch controller (SSC) and a central message switch. The SS provides digital paths for switching connections between the BMs. The SSC will get an idle time slot from one of the two channels available and sets up a path in SS on which any two terminals can communicate.

Input Output Processor (IOP) :

The IOP is a front end of the DSS and is based on UNIX V.2 operating system. All the status of the exchange will be maintained in the form of files as a back up and whenever the exchange goes down it can download the data from IOP. THe memory is available in the form of Disk and Magnetic tape transport etc. The IOP provides interfaces to video display units and hard copy printers etc.



CDOT DSS BASIC ARCHITECTURE

CHAPTER 2

#7 SIGNALLING SYSTEM

2.1 Signalling for Telephony

The basic service offered by a telephony network is an end to end (user to user) circuit switched connection. A dedicated pair of lines will be allotted between the subscriber and the exchange for the whole call duration. If it is an inter exchange call, some inter_exchange circuits will also be used. The subscriber makes a call and informs the exchange about the identity of the called party. These signals can be of the form loop (off-hook), no_loop (on-hook) or loop disconnect (digits). Then the first exchange will select a free inter--exchange circuit and signals the next exchange about the identity of the called party and connects the calling party to the outgoing circuit. These signals are also of the form of loop, no-loop at the outgoing end and normal or reversed polarity of the battery for clear forward, answer, called party clear etc. at incoming end. The other exchanges also will do the same to forward the connection upto last exchange. After completion of the call all the inter exchange circuits will be released. So for setting up a call and releasing it certain signalling information needs to be conveyed between the exchange and the subscriber and between the exchanges.

There are different signalling schemes like loop-disconnect # 5 etc. Signalling schemes are also differentiated in two ways. One is Channel Associated Signalling (CAS) type and the other is Common Channel Signalling (CCS) type. This CCS is a modern one which is considered to be a stepping stone in the development of ISDN and is designed to support both voice as well as non-voice into communication networks.

2.1.1 Channel associated signalling (CAS)

There is a dedicated signalling channel for each of the inter exchange circuit. The signal conveyed as change in steady state has to be derived at the other end with the help of time discrimination. In a SPC type of exchange the signals are to be converted in the form of events at one end and need to be converted into steady signals at the other end.

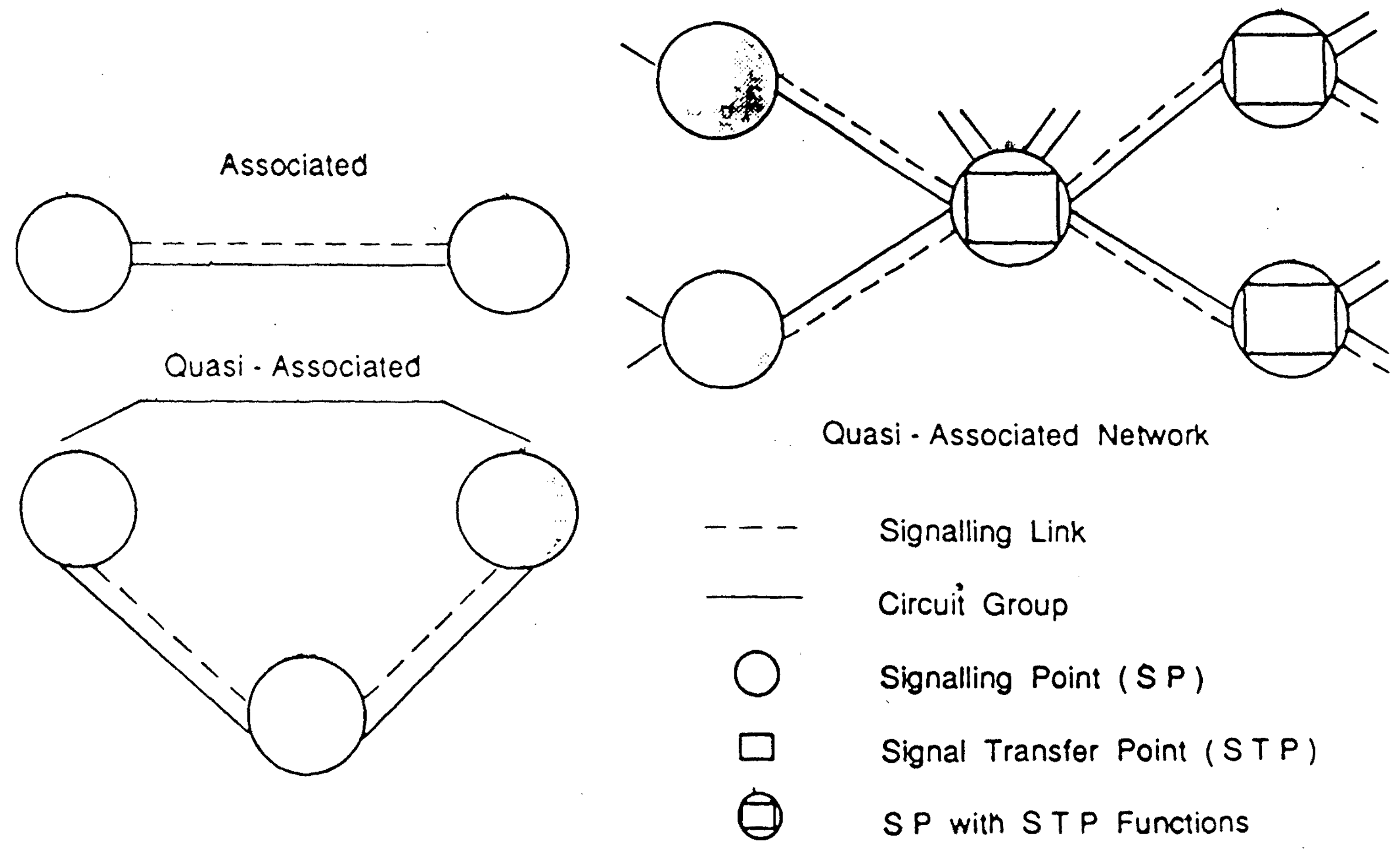
2.1.2 Common Channel Signalling System (CCS)

This system uses a common data link between the processors of two SPC processors of two exchanges. The signalling information of any voice circuit can be sent in the form of messages. The message should identify to which voice circuit it is meant for. If the transmission medium is analog then modems are used to derive the data link.

Advantages of CCS over CAS :

1. Fast and reliable.

2. Elimination of per circuit hardware as in the case of CAS and thus more economical.


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

3. In digital transmission if the time slot 16 channels are unused they can be used for voice traffic.

4. New features like closed User Group (CUG), Redirection of calls, Credit calls from any telephone etc. can be introduced.

2.1.3 Modes of operation of CCS

There are two modes of operation.

1. Associated mode :

There will be a signalling link between any two exchanges.

2. Quasi Associated mode :

Here two exchanges will be connected by a circuit group and they exchange the signalling information via a third exchange known as Signalling Transfer Point (STP) where as the former two will be Signalling Points (SPs). A Signalling Point can be a source or a sink.

Quasi associated working saves inter exchange data-links & signalling terminal hardware. Saved data links can be used for voice connections.

2.2 CCITT #7 SS

This is a CCS type with quasi associated networking features. For reliability two quasi associated SPs will have a minimum of two signalling routes via different STPs &

not more than two STPs in tandem. There are various kinds of signalling messages used in #7 SS.

2.2.1. Signalling messages

There are three types of signalling messages

1. Message Signal Unit (MSU).
2. Link Status Signal Unit (LSSU).
3. Fill In Signal Unit (FISU).

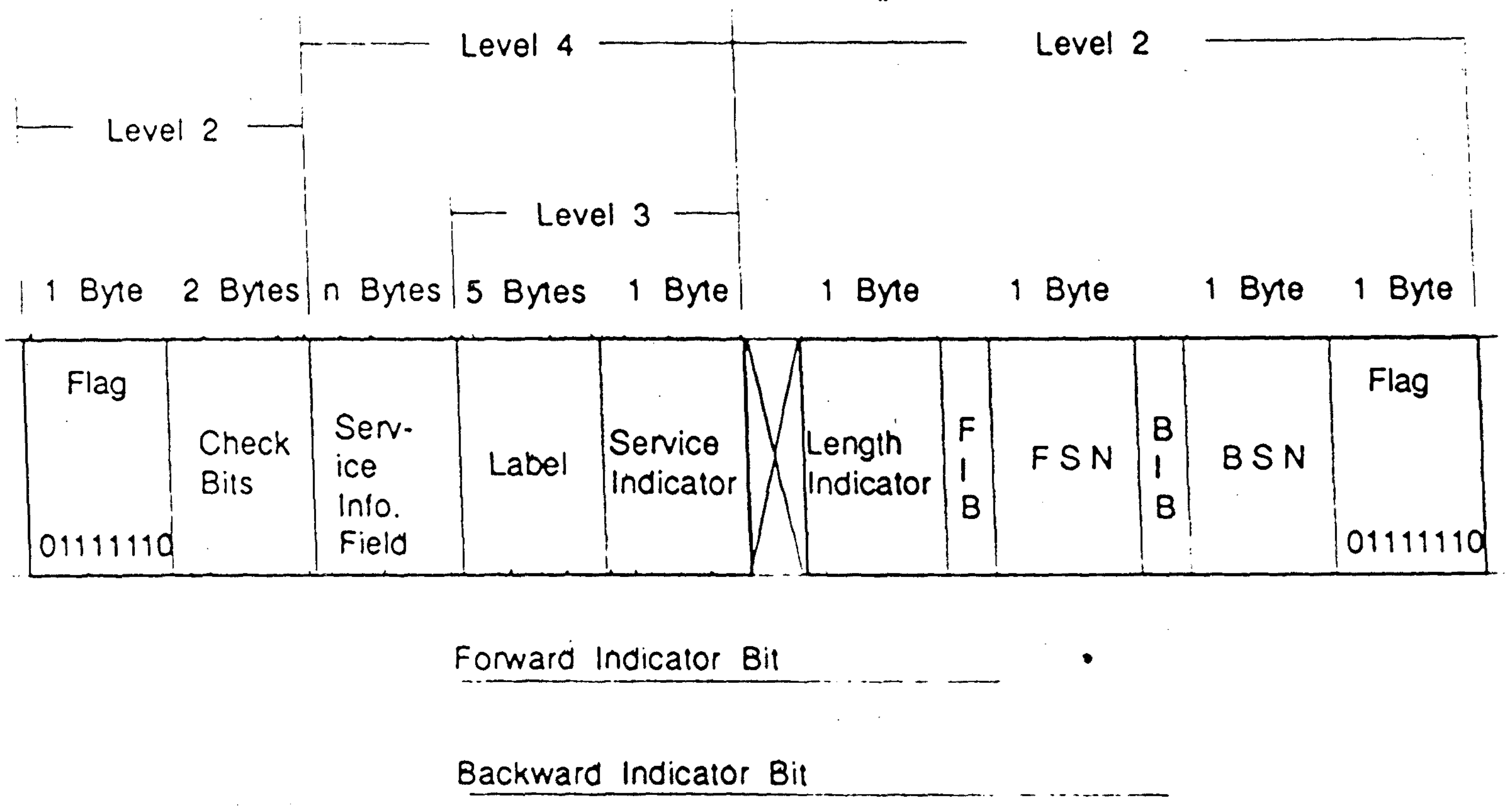
Message signal unit (MSU) is the information carrier over the #7 signalling link. It is divided into number of fields by different levels. It contains level 3 functions like The Originating point code (OPC), Destination Point Code (DPC), and Circuit Identification Code (CIC) for routing and level 2 functions like Forward sequence number (FSN), backward sequence number (BSN), Check bits (CK) etc. to ensure reliable transmission.

Whenever there is an error, it will be corrected by retransmission. Even when there is no message traffic on the channel, FISUs will be transmitted to monitor the error performance of the channel.

LSSUs are sent to convey the status of the link during the alignment.

2.2.2. Structured view of #7 SS

#7 SS has a layered structure. It comprises of four levels. Levels 1 to 3 constitute the message



Level 4
Level 2
Level 2
Level 3
1 Byte
2 Bytes
n Bytes
5 Bytes
1 Byte
1 Byte
1 Byte
1 Byte
1 Byte
Flag
01111110
Check Bits
Serv-ice Info. Field
Label
Service Indicator
Length Indicator
FIB
FSN
BIB
BSN
Flag
01111110
Forward Indicator Bit
Backward Indicator Bit

MSU FORMAT

transfer part which ensures a reliable transmission of the message without loss or duplication. A number of parallel user parts will make use of this MTP to convey the messages.

Level 1 :

It define a Signalling data link layer which may of 64kb/s channel (usually timeslot 16 of PCM).

level 2 :

It defines the functions and procedures for the reliable transmission of variable length message signal units (or packets) received from Level 3. The functions will be like sequence numbering, keeping track of incoming sequence, error checking & correcting by retransmission.

Level 3 :

It defines common functions for all signalling links and to user parts of level 4. Three major categories of transport functions are there.

1. Message transfer functions :
   a) Proper routing of the outgoing messages.
   b) Collecting all the incoming messages by checking the destination point code.
   c) Distribution of incoming messages to different user parts of Level 4.

2. Signalling connection control part (SCCP) :

To establish, to maintain, to dismantle logical data connection for transporting packets from ISDN users to slow speed terminals.

3. Signalling network functions :

a) Signalling link management.

b) Signalling route management.

c) Signalling traffic management.

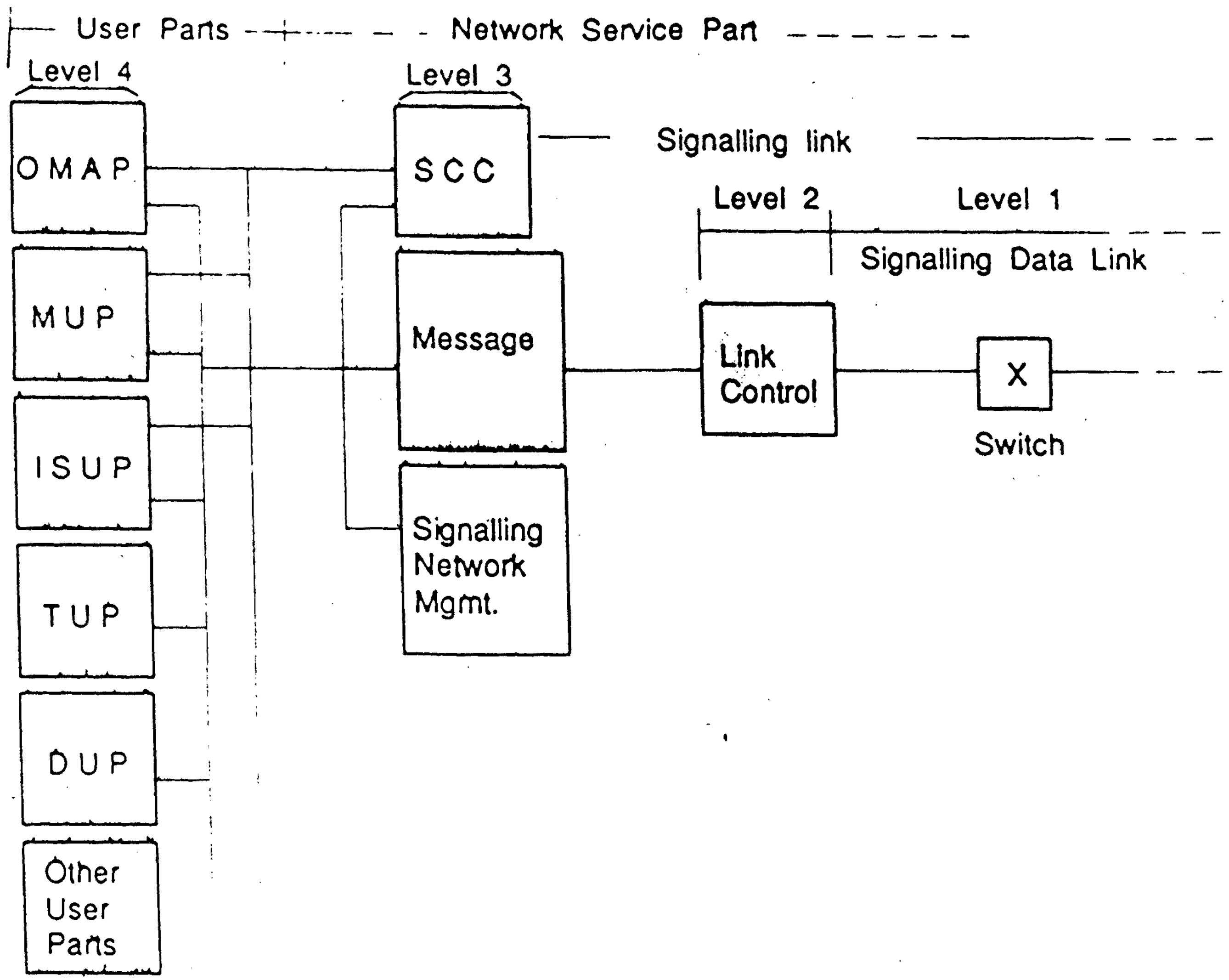
Level 4 :

Depending upon the bits in the field of service indicator one of the sixteen parallel user parts will be selected. Presently there are two user parts. One is Telephone user part and second ISDN user part is in evolution.

2.3 Basis for the Project Work

2.3.1 Over view of #7 SS in C-DOT DSS

The #7 SS is now being implemented in C-DOT DSS. A #7 SU housed in standard terminal frame connects to the time switch of a BM a one of the 8 serial interfaces of 128 timeslots. The internal message channels are different from that if #7 protocol .So the #7 SU is so designed to make both protocol conversion, format translation during routing of messages between #7 links and BMs. Work is in progress to develop a packet switch to implement CCS in PABXs, RSUs and ISDN users.



STRUCTURE OF CCITT NO. 7 SS

2.3.2 Hardware Architecture of #7 SU

#7 SU consists of two cards.

1. Protocol card :

Each protocol card has eight protocol channels called terminals where two protocol channels are controlled by one microcomputer. By changing the firmware different protocols such as #7, I, D etc. can be handled. The firmware of the channel along with the HDLC chip connected to the channel will perform #7 level 2 functions like flag generation/detection, zero insertion/deletion etc. The 64kb/s serial streams of 32 channels of four protocol cards constitute 2048kb/s stream and four of these streams from a total of 16 PCs constitute 8Mb/s stream towards the Time switch.

2. #7 CPU card :

#7 CPU card is basically a 68000 based machine and contains one DMA controller, an EPROM, and peripherals. The CPU card is duplicated. Always one will be in ON-LINE and the other will be in STAND-BYE mode. A serial communication link interfaces both the CPUs.

2.3.3 Software Architecture of #7 SU

The software for implementation of #7 Signalling system is distributed over many processes like BP, AP, IOP etc. It can be viewed as six different parts.

1. Protocol Handler :

It handles all level 2 functions for the #7 SU and resides in Protocol card.

2.Operating system :

It is a CCITT #7 adaptable operating system.

3.Database Subsystem :

It processes the #7SU related data which is stored at AP and BP using standard database routines.

4. Maintenance Subsystem :

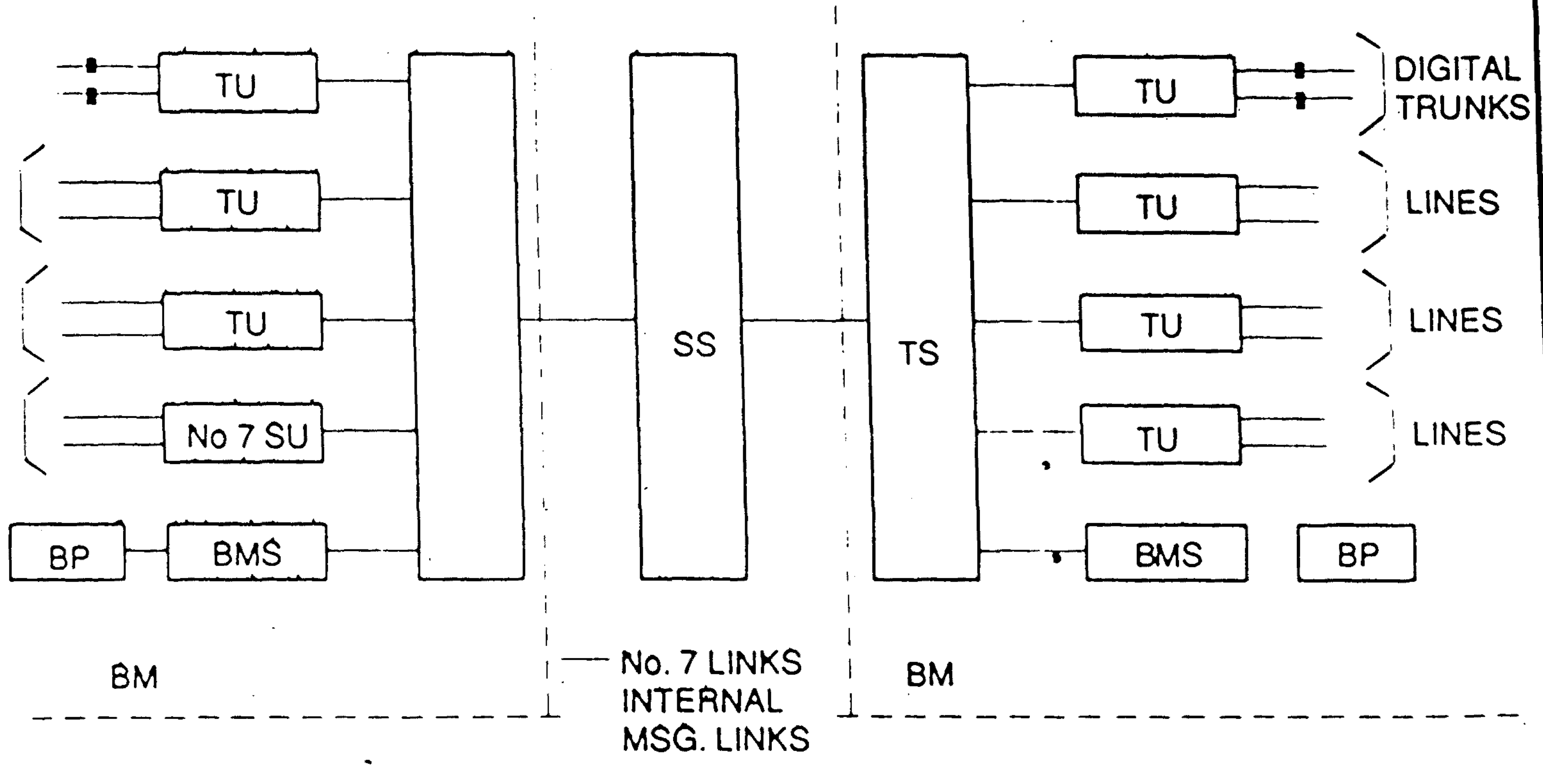
The functions include initialization of #7SU , monitoring its health (including Protocol Cards), running audit tests on database periodically and management of terminals and maintenance of Protocol Cards and I channels.

5. Message Handler :

This is a CPU resident software to route the I, D, #7 messages via the concerned Protocol Cards.

6. Network manager :

The function includes management of link, route, traffic for #7 network using #7 management processes.



NO. 7 SU IN C-DOT DSS

CHAPTER 3

PROJECT OVERVIEW

3.1 MAN - MACHINE INTERFACE

The man-machine language (MML) is used to facilitate the various administrative and Maintenance functions. It provides a transparent interface to the operator to communicate with the exchange and to modify the internal state.

#7 SU maintains a database about circuit group sets, data link groups, link set bundles, signalling route sets, etc.

In our case the database required, is related to the terminals and Protocol Cards. All the subscriber related signalling messages will go via the #7 channel. Level 2 functions will be handled by the terminals in the protocol card. The present status of these terminals and protocol cards will be maintained in the database.

Before using a terminal for signalling, the exchange will check for the status of the terminal whether it is in service or out of service and also checks for the status of the protocol card relating to that terminal is in EQUIPPED or in DEQUIPPED condition.

This information will help the exchange to use proper terminals for signalling. This database will be created, modified by the operator using man-machine commands.

3.1.1 Architecture

The data is stored in different processors depending upon the requirements, generally will be in AP, BP, #7SU and at IOP. If at any time the exchange goes down, all the present status can be downloaded from the IOP. Data global to all BPs will be maintained at AP.

For this interfacing purpose five processes have been recognized.

1. Command recognition process :

CRP is a dynamic process residing at IOP. Whenever the operator logs_in the terminal immediately this process will be created. It is a command interpreter and takes the parameters from keyboard. It performs validation checks like range, set checks etc. on the input parameters. After all the validations it will create a dynamic process called CEP for that particular command and pass the relevant information.

2. Command execution process :

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

1. It checks for consistency of the information.

2. Exhaustive parameter validations.

3. Updations of database at IOP and informs the same to the exchange.

Whenever CEP is created, it gets the arguments from CRP and makes an exhaustive parameter validations. After all the checks it will send a message to ACI in AP where the ACI depending upon the requirements, will send a message to AP in AM, BP in BM and NDUP in #7SU. Based on the Success/Failure message from ACI, CEP will send an output message to CRP.

If the destination of the command is IOP itself CEP will do all the updations. But if it is SU or BP or AP then CEP will send a message to ACI and wait for the response.

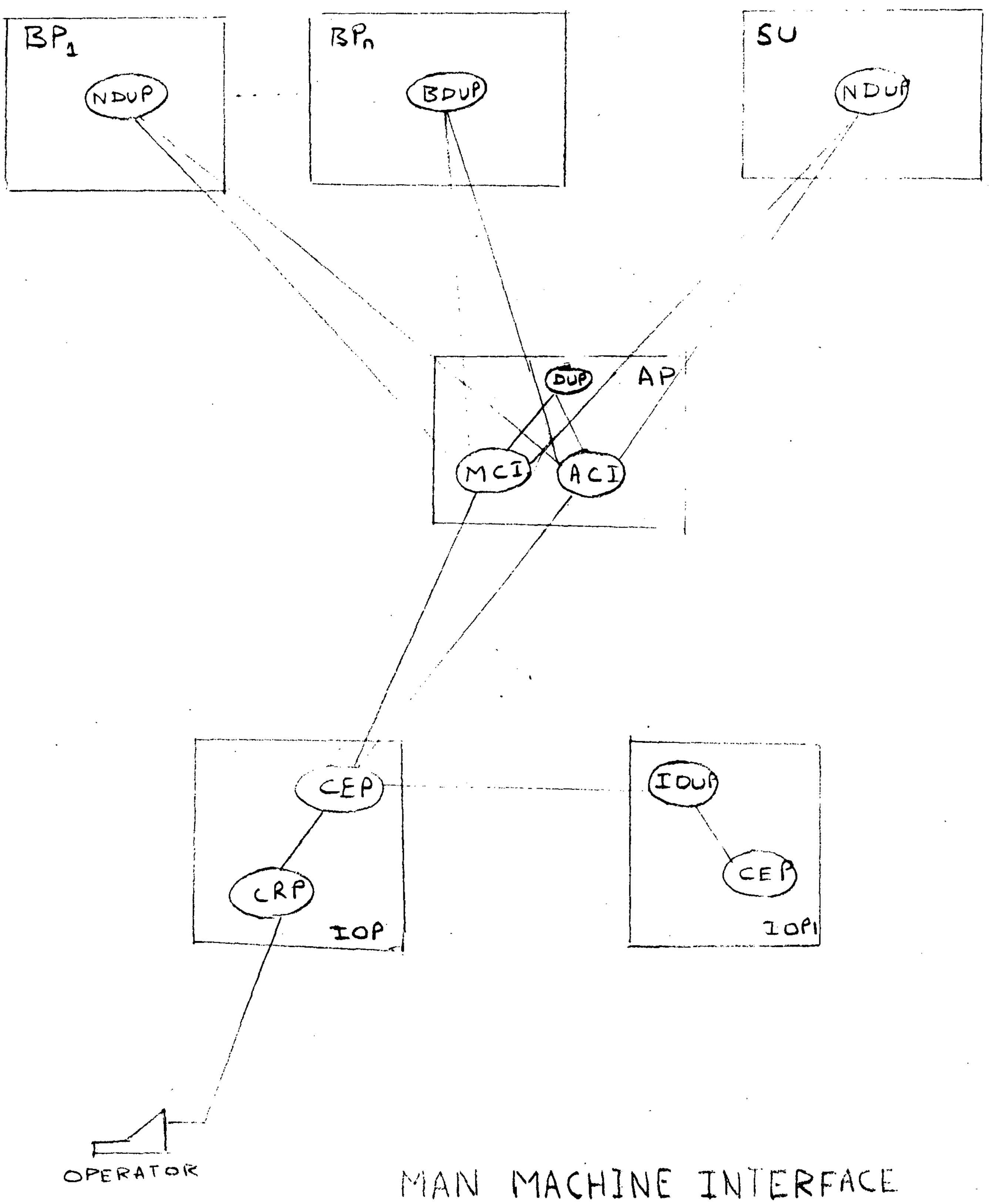
3. Input-Output Data Update Process :

IDUP is an eternal process residing in IOP. For reliability issues IOP work in Duplex environment. The CEP will send a message to IDUP which in turn will create an identical CEP at IOP1 for the data updtion. The CEP of first IOP will wait for the message from CEP of IOP1 indicating the successful completion of updations before proceeding further.

4. Administrative command interpreter :

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 on it the ACI will send messages to Data Updation Process at AP or Base Updation Process at BP or #7 SU data update process at NDUP. Based on the result it will send a Success/Failure message to CEP.

5. Output Outside Dialogue :



MAN MACHINE INTERFACE

The OOD is an eternal process residing in IOP. When the CEP sends back the result message the OOD displays the results in a proper format to the operator.

3.1.2 Modes of operation

The man-machine commands can be executed in two modes.

First is Growth mode also known as Off-line operation. The command will update the data only at IOP and the Disk. It will not affect the data at exchange. The modified data will be periodically downloaded from IOP to the exchange.

The other is On-line mode. Here the data residing both at IOP and the exchange will be modified. These modifications immediately will effect the performance of the exchange. It is also known as Normal mode.

This mode will be sent as status_flag from CRP to CEP .

3.2 Man-machine interface for #7 Signaling system

Every #7 SU unit can have 16 protocol cards . In each Protocol card there will be eight terminals which can handle level 2 protocols . The exchange has to modify its database according to the physical status of the terminal so that it can use proper terminals for signalling.

All the terminals are soft configurable to any type of protocol like #7, I channel, D channel type etc. In the default state all are configured to #7 type.

A terminal can have five different states. EQUIPPED, INS or FREE, ACTIVE, OOS, Non Existent. Terminal status at IOP can take four states.

1. Equipped	:	Configured, before Diagnostics are run.
2. INS	:	Configured, after Diagnostics.
3. OOS	:	Configured & off-line.
4. Non existent	:	Not existing in the pool.

3.2.1 Commands

Depending upon the maintenance requirement four commands are recognized.

First is Equipping a protocol card (EQUIP-PC). This command is to make the protocol terminals available to the exchange. Here the operator physically jacks_in a card in the proper slot and gives a command with the slot identification, the type of card being used, and its version number.

The function of the command is to make an entry for that card and the terminals in that card, in the database at IOP and informs the exchange about it.

In the exchange the SU will identify all the terminals corresponding to that Protocol Card as EQUIPPED and runs Diagnostics on the terminals and all successful terminals will be identified as INS. From then onwards the exchange will use the terminals for signalling purposes.

The second command is Deequipping a protocol card. (DEQUIP-PC).

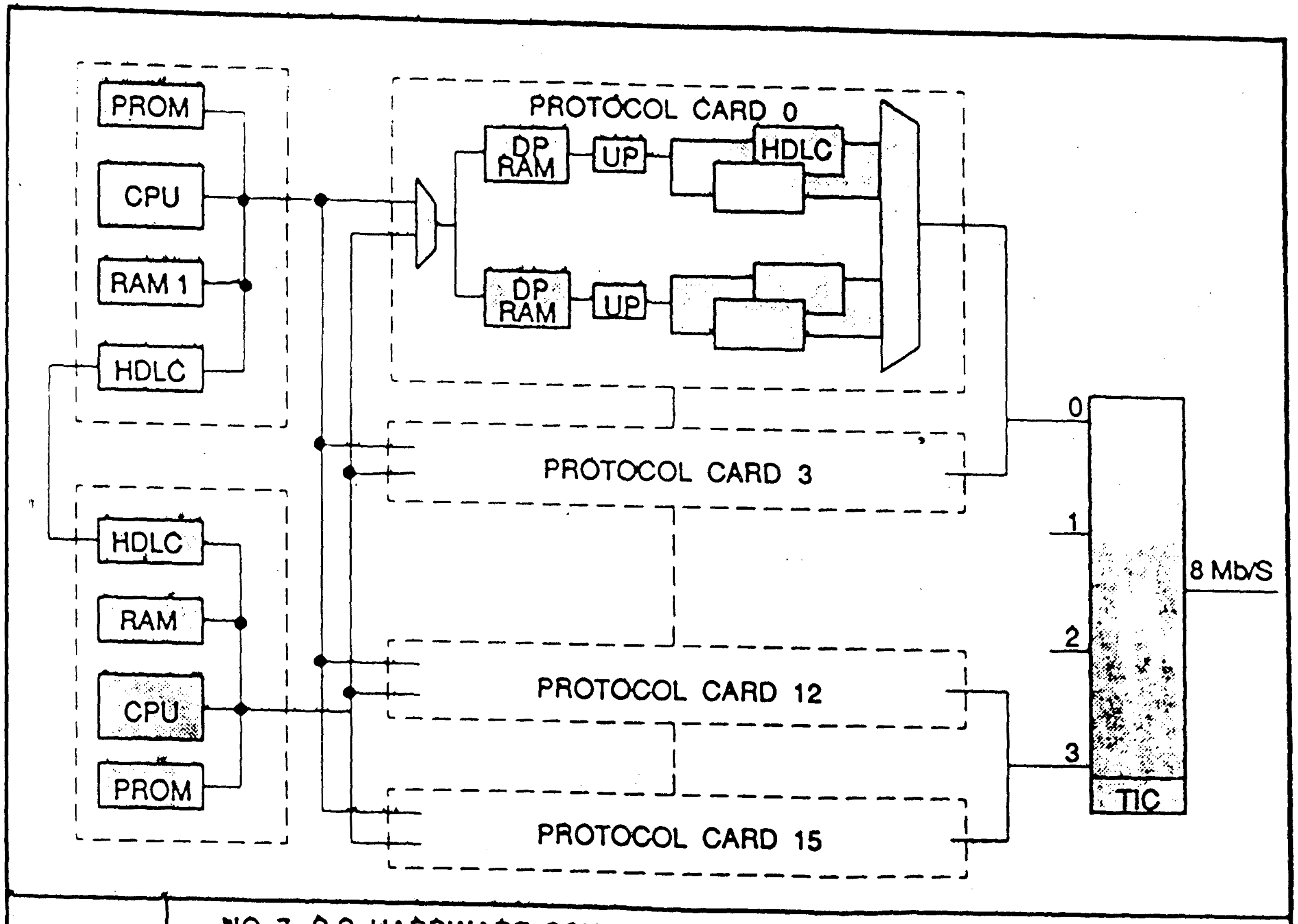
As the name suggests it will delete a protocol from on-line mode. At SU all the active terminals on that PC will be switched over. After executing this command the exchange will recognize the terminals in that card as OOS.

Another command recognized is Put terminal out of service (PUT_TRM_OOS). Here the operator will specify the reason for doing it. The reason can be either for running the diagnostics or for any other reason like redistribution of the load etc. The command will accordingly modify the database at IOP and informs the same to the exchange.

SU will then select another terminal either from the Protocol Card other than the one to which the present terminal is related if the reason is for running diagnostics or from anywhere if the reason is anything else and make a terminal switchover.

The status of the terminal that is being put out of service will be modified as OOS. This terminal further won't be used for signalling purposes.

The last command is Put terminal in service. Here a terminal will be put into service and is informed to the exchange.



NO. 7 S S HARDWARE SCHEME

CHAPTER 4

DESIGN AND IMPLEMENTATION

Whenever the operator logs in, a dynamic process (CRP) will be created by UNIX operating system for that terminal. When the operator inputs a Man-machine command CRP will check whether the command is valid one or not and checks the ranges of the parameters, whether they are within the permitted values or not. After this the CRP forks and creates a new process called CEP and goes into Sleep state.

CEP will perform various validations and syntax checks on the parameters. After successful completion of the validations it updates the files at IOP(s) and sends a message containing the information of updation to the process Administrative command interpreter (ACI). (in on-line mode operation). The ACI mails this message to DUP for data updations at AP and updations at SU.

After all the actions the ACI will send a Success/Failure message to CEP which in turn wakes up the CRP by a message. The CRP will send this result to OOD to display the results to the operator.

4.1 Algorithm

1. CEP will get the parameters from the operator and checks for the validation. It gets two arguments

status_byte and qid (of the message queue being used for the flow of messages between CRP and CEP) from CRP.

2. Validations of opcode and subfield of the command.

3. Exhaustive parameter validations.

4. Allocates space for history buffer. History buffer is needed because whenever the execution fails because of any fault, the recovery routine using history buffer will undo all the updations so far.

In a history buffer, for the record which is going to be modified both the present & previous records, for a record which is going to be deleted, the previous record, and for an insertion record, the current record will be stored.

5. After validations the CEP will send a message to IDUP to do the modifications at the duplicated IOP i.e. IOP1.

6. validations at that IOP1 .

7. Filling the History buffer for IOP1.

8. Updations at the IOP1.

9. CEP of IOP1 sends message to IOP.

10. Updations at IOP.

11. In on-line mode CEP at IOP sends message to ACI and waits for the response.

12. CEP then sends message to CRP and exits.

4.2 Equip protocol card : (EQUIP-PC) CEP.

4.2.1 Input parameters :

1. SLOT-ID (Slot id)

PARAMETER NAME	: SLOT-ID of the protocol card
MNEMONIC	: SLOT-ID
TYPE	: COMPOUND
POSSIBLE VALUES	: 1-1-1-3 TO 32-4-4-24
DEFAULT	: NONE

2. VERSION (version)

PARAMETER NAME	: VERSION NUMBER of protocol card
MNEMONIC	: VERSION
TYPE	:
POSSIBLE VALUES	:
DEFAULT	:

3. HW_TYPE (Hardware type of protocol card)

PARAMETER NAME	: HARDWARE TYPE
MNEMONIC	: HW_TYPE
TYPE	: NUMERIC
POSSIBLE VALUES	: PROTOCOL
DEFAULT	: PROTOCOL

The command execution process contains following modules.

File name	function
nepcd_entr.c	*main()*

nepcd_main.c	*nepcd_main(), nl_rec_validate_ccs(), nl_validate_ccs().*
nepcd_anls.c	*nepcd_rslt_ana()*
nepcd_form.c	*nepcd_form_cmderr_msg()*
nepcd_out.c	*nepcd_out_msg()*
nepcd_rcvr.c	*nepcd_rec_rtn()*

4.2.2 Command flow :

. CEP will be created by CRP using UNIX system calls and status_byte & queue_id will be the arguments from CRP to CEP. These two arguments will be taken by the *main()* function in nepcd_entr.c.

The *main()* function initiates a file table structure which keeps all the files currently open to the process.

The function *a_set_def()* will trap all the UNIX signals like interrupts to make CEP to continue its execution.

Performs checks on the arguments.

The *msg_rcv()* function will receive a message from CRP from the queue specified by queue_id.

Calls the *nepcd_main()* for further execution.

nepcd_main()

It validates the opcode and subfield of the command.

. Creates a history buffer which will store the present and past records for the purpose of recovery if there occurs any problem in execution.

. It gets the slot_id as an input from CRP and gets the Card_id using Slot_id to Card_id conversion tables. These tables will be maintained in the database.

. It checks the database using the routine *nl_validate_ccs()* with card_id as primary key, for the existance of that card.

. At any point during the execution, if any error like ERR_BIT set etc., occurs, a recovery routine *nepcd_rec_rtn()* is called.

. If the card is present, it means that the card is already in the database, so an error recovery routine is called.

. The SU should be in equipped state. A check on this will be made by accessing Fequip_rec with bm_num,rack_num, frame_num as key . The frame_type field of Fequip_rec must be SU_FRAME.

TH-6840

. After all the validations it sends a message to IDUP for validations and updations at IOP1.

. After getting a success message from IDUP, data updations will be done. A record will be added in Fcard_data for each card being equipped. These will be written into the data base using ISAM *write()* operation.

Similarly corresponding to each terminal related to that protocol card, a record will be inserted in the Fterminal_data and each terminal status will be written as EQUIPPED.

Along with the updations history buffer fields will also be filled.

The file structures Fcard_data & Fterminal data are su_id dependent. The su_id is derived from fs_su_id field by accessing Fsu_data Fsu_data with ns_bm_id as the secondary key.

. If the CEP is run in NORMAL (ONLINE) mode then a message (Ncktcre20) is sent to ACI and CEP goes into wait state and waits for the result message (Ncktcreo20) from ACI.

. Depending upon the operating mode and result, the result CEP will send a success or error message (Ncktcrer20) to CRP. This is done by *nepcd_out_msg()* .

. In the case of error the recovery routine is called. The *nepcd_rec_rtn()* function depending on the type of error, will call a function nepcd_form_cmderr(), with different parameters.

. The *nepcd_form_cmderr()* creates an error message (Acmderr01) and sends it to CRP.

CRP will send this message to OOD which displays the result in a proper format.

4.2.3 Data and File Structures :

File structures :

```
typedef struct
          {
           Ushort                  trml_num;
           Uchar                   version;/* Version number
                                           of equipment*/
           Ushort                  hw_type;/* hardware type
                                           of equipment
                                     : not the trml. type */
           Uchar                   ter_status;
      }    Fterminal_data;
```

The ISAM file TRMLFILE contains the records of data related to terminals namely Fterminal_data and the primary key is trml_num.

```
typedef struct
     {
           Fikey          card_id; /* identity of the

                                        protocol card. */
          } Fcard_data;
```

The ISAM file PCRDFILE Here the primary key is card_id ;

```
typedef struct
     {
          Uchar    ns_status;
          Uchar    ns_bm_id;
          Uchar    ns_tic_id;
          Uchar    ns_link_id;
          Uchar    ns_unit_id;
          Uchar    ns_dummy;
     } Nss_su_info;
```

```
typedef struct
     {
          Uchar         fs_su_id;
          Nss_su_info   fs_su_data;
     } Fsu_data;
```

The ISAM file is SUDTFILE Here fs_su_id is primary key and ns_bm_id is the secondary key.

```
     typedef struct SLOT_INFO
          {
               Uchar    card_typ;
               Uchar    card_ver_no;
               Uchar    num_of_ckts;
               Uchar    start_ckt_num;
          } Slot_info;


     typedef struct FEQUIP_REC
          {
                   Uchar          ten_part[3];
                   Uchar          len_part[2];
                   Uchar          frame_typ;
                   Slot_info      slot[DMAX_SLOTS];
                   Char           asgn_byt[DMAX_CKTS];
        } Fequip_rec;
```

The ISAM file is TNENFILE and ten_part[3] is the primary key.

Data Structures at SU

Data structures at SU are either accessed or updated at SU.

```
     typedef struct
          {
                 Ushort                    trml_num;
                 Uchar                     type;
                 Uchar                     link;
                 Uchar                     ter_status;
                 Uchar                     ec_option;
                 Uchar                     msrv_cnt;
                 Uchar                     empty_chn_req;
                                               /* from level 3 */
                 Uchar                     bm_id;
                 Uchar                    reset_time[NTIME_STR_LEN];
                                                /* reset time */
                 Ushort                    mrecv_cnt;
                                           /* msg recv. since
                                                     last reset */
                 Ulong                     dpbadr;
                                           /* base addr of DPRAM */
                 Nl312buf_struct           *ptr1_l312buf;
                 Nl312buf_struct           *ptr2_l312buf;
                 Nl213buf_struct           *ptr_l213buf;
                 Nrx_info_struct           *ptr_rx_info;
                 Ntx_info_struct           *ptr_tx_info;
                 Nidledq_info_struct       *ptr_idledq_info;
          } Nterminal_info_struct;
```

Here   trml_num   is the primary key.

Messages structures

This header is used for messages from CRP -> CEP and CEP -> ACI.

```
typedef struct
       {
             Ushort        app_num;
             Uint          serial_num;
       } Execution_num;


typedef struct
       {
             Ulong         crp_id;
             Ushort        user_id;
             Ushort        size;
             Uchar         status_byte;
             Uchar         output_dev;
             Uchar         session_id;
                 /* Incremented as soon as user logs in or */
                 /* job-id in the same session exceeds 256 */
             Uchar         job_id;
             Execution_num exe_num;
       } Cmdhdr;


  typedef struct
       {
             Uchar   bm_num;
             Uchar   rack_num;
             Uchar   phy_frame_num;
             Uchar   slot_num;
       } Strt_posn;
```

Message from CRP -> CEP

```
typedef struct
       {
            Hdr          hdr;
            Cmdhdr       cmdhdr;
            Ushort       hw_type;
            Uchar        ver_no;
            Uchar        dummy_1;
            Strt_posn    slot_id;
       } Ncktcrei20;
```

Message from CEP -> ACI

```
typedef struct
      {
            Hdr             hdr;
            Cmdhdr          cmdhdr;
            Uchar           su_id;
            Ushort          hw_type;
            Uchar           ver_no;
            Uchar           card_id;
      } Ncktcre20;
```

Header for Result Messages

This header is used for result messages from ACI -> CEP and CEP -> CRP.

```
typedef struct
      {
            short           app_num;
            Uint            serial_num;
      } Execution_num;

typedef struct
      {
            Ushort          user_id;
            Ushort          size; /* size of the  of message */
            Uchar           no_of_entries;
            Uchar           output_dev;   /* V : VDU     */
                                 /* P : PRINTER */
                                 /* D : DISK    */
            Uchar           session_id;
            Uchar           job_id;
                 /* incremented as user logs in. 0 to 255 */
            Uchar           dummy_byte;
            Uchar           status_byte;
             /* Bit 0 : Command initiated from command file */
             /* Bit 1 : Command initiated by calendar       */
             /* Bit 2 : Results not necessarily to be sent  */
            Ulong           crp_id;
            Execution_num exe_num;
      } Rslthdr;
```

Message from ACI -> CEP

```
typedef struct
      {
            Hdr             hdr;
            Rslthdr         rslthdr;
            Uchar           suc_type;
                                /* SUCCESS or not SUCCESS */
            Uchar           dummy;
      } Ncktcrer20;
```

Message from CEP -> CRP

```
typedef struct
     {
          Hdr            hdr;
          Rslthdr        rslthdr;
          Ushort         hw_type;
          Uchar          ver_no;
          Uchar          dummy_1;
          Strt_posn      slot_id;
          Uchar          su_id;
     } Ncktcreo20;
```

4.2.4 EXAMPLE

The following example demonstrates the inputting of the the command and corresponding input parameters. This also displays the result of the EQUIP_PC command.

```
U    < EQUIP_PC

S              Equip Protocol Card
 ________________________________________________________
|                                                        |
|                                                        |
|           VERSION    = 1                               |
|                                                        |
|           HW_TYPE    = 1                               |
|                                                        |
|           SLOT-ID    = 1-1-1-3 & 1-1-1-5               |
|                                                        |
|     < E/R/T    = E                                     |
|________________________________________________________|
```

The result screen will be as follows :

```
                 Equip Protocol Card
 **************************************************
 *                                                *
 *    Version    = 1                              *
 *                                                *
 *    Hw_type    = 1                              *
 *                                                *
 *    Slot id    = 1-1-1-3 & 1-1-1-5              *
 *                                                *
 *                                                *
 **************************************************
```

4.3 Dequip protocol card (DEQUIP-PC) CEP

4.3.1 Input parameters :

1. SLOT-ID (Slot id)

PARAMETER NAME	: SLOT-ID of the protocol card
MNEMONIC	: SLOT-ID
TYPE	: COMPOUND
POSSIBLE VALUES	: 1-1-1-3 TO 32-4-4-24
DEFAULT	: NONE

The preliminary checks on the range/set will be done by CRP.

The command execution process contains following modules.

File name	function
nupcd_entr.c	*main()*
nupcd_main.c	*nupcd_main()*, *nl_rec_validate_ccs()*, *nl_validate_ccs()*.
nupcd_anls.c	*nupcd_rslt_ana()*
nupcd_form.c	*nupcd_form_cmderr_msg()*
nupcd_out.c	*nupcd_out_msg()*
nupcd_rcvr.c	*nupcd_rec_rtn()*

4.3.2 Command flow

. CEP will be created by CRP by calling a UNIX system call 'fork' and status_byte & queue_id will be the arguments from CRP to CEP. These two arguments will be taken by the *main()* function in nupcd_entr.c.

. The *main()* function initiates a file table structure which keeps all the files currently open to the process.

. The function *a_set_def()* will trap all the UNIX signals like interrupts to make CEP to continue its execution.

. Performs checks on the arguments.

. The *msg_rcv()* function will receive a message (Ncktcrei21) from CRP from the queue specified by queue_id.

. Calls the *nupcd_main()* for further execution.

nupcd_main()

. It validates the opcode and subfield of the command.

. Creates a history buffer which will store the present and past records for the purpose of recovery if there occurs any problem in execution.

. It gets the slot_id as an input from CRP and gets the Card_id using Slot_id to Card_id conversion tables. These tables will be maintained in the database.

. It checks the database using the routine *nl_validate_ccs()* with card_id as primary key, for the existence of that card.

. At any point during the execution, if any error like ERR_BIT set etc., occurs, a recovery routine *nupcd_rec_rtn()* is called.

. The card should present in the database, with equipped & jacked-in state. *nl_validate_ccs()* will check for it.

. The SU should be in equipped state. A check on this will be made by accessing Fequip_rec with bm_num,rack_num, frame_num as key . The frame_type field of Fequip_rec must be SU_FRAME.

. After all the validations it sends a message to IDUP for validations and updations at IOP1.

. A record will be deleted in Fcard_data for each card being deequipped.

. All the records of the terminals related to that protocol card in Fterminal_data will be deleted.

. Along with the updations history buffer fields will also be filled.

. The file structures Fcard_data & Fterminal data are su_id dependent. The su_id is derived from fs_su_id field by accessing Fsu_data Fsu_data with ns_bm_id as the secondary key.

. If the CEP is run in NORMAL (ONLINE) mode then a message (Ncktcre21) is sent to ACI and CEP goes into wait state and waits for the result message (Ncktcreo21) from ACI.

. Depending upon the operating mode and result, the result CEP will send a success or error message (Ncktcrer21) to

CRP. This is done by *nupcd_out_msg()* .

. In the case of error the recovery routine is called. The *nupcd_rec_rtn()* function depending on the type of error, will call a function *nupcd_form_cmderr()*, with different parameters.

. The *nupcd_form_cmderr()* creates an error message (Acmderr01) and sends it to CRP.

. CRP will send this message to OOD which displays the result in a proper format.

4.3.3 Data and File Structures

file structures at IOP :

File structures Fterminal_data, Fcard_data, Fsu-data, Fequip_rec are maintained as ISAM files in the database. The structures are defined earlier.

Data Structures at SU

Data structures Nterminal_info_struct is described in the previous process.

Messages structures :

The header messages for command and result are common to all processes and are described earlier.

```
typedef struct
     {
          Uchar    bm_num;
          Uchar    rack_num;
          Uchar    phy_frame_num;
          Uchar    slot_num;
```

```
        } Strt_posn;
```

Message from CRP -> CEP

```
    typedef struct
        {
            Hdr             hdr;
            Cmdhdr          cmdhdr;
            Uchar          dummy_1;
            Strt_posn      slot_id;
        } Ncktcrei21;
```

Message from CEP -> ACI

```
    typedef struct
        {
            Hdr             hdr;
            Cmdhdr          cmdhdr;
            Uchar           su_id
            Uchar           card_id;
        } Ncktcre21;
```

Message from ACI -> CEP

```
    typedef struct
        {
            Hdr             hdr;
            Rslthdr         rslthdr;
            Uchar           suc_type;
                            /* SUCCESS or not SUCCESS */
            Uchar           dummy;
        } Ncktcrer21;
```

Message from CEP -> CRP

```
    typedef struct
        {
            Hdr             hdr;
            Rslthdr         rslthdr;
            Strt_posn       slot_id;
            Uchar           su_id;
            Uchar          dummy_1;
        } Ncktcreo21;
```

4.3.4 EXAMPLE

The following example demonstrates the inputting of the command and corresponding input parameters. This also displays the result of the DEQUIP_PC command.

```
U    < DEQUIP_PC

S              Dequip Protocol Card
 _______________________________________________
|                                               |
|              SLOT_ID = 1-1-1-3 & 1-1-1-5      |
|                                               |
|     < E/R/T   = E                             |
|_______________________________________________|
```

The result screen will be as follows :

```
              Dequip Protocol Card
************************************************
*                                              *
*                                              *
*    Slot_id   : 1-1-1-3 & 1-1-1-5             *
*                                              *
*                                              *
************************************************
```

4.4 Put Terminal Out Of Service (PUT_TRM_OOS) CEP

4.4.1 Input parameters

1. SLOT-CKT (Slot number)

PARAMETER NAME	:	SLOT-CKT of the protocol card
MNEMONIC	:	SLOT-CKT
TYPE	:	COMPOUND
POSSIBLE VALUES	:	1-1-1-3-1 TO 32-4-4-24-8
DEFAULT	:	NONE

2. REASON (Reason :either for running the DIAGNOSTICS or OTHER)

PARAMETER NAME	:	REASON
MNEMONIC	:	REASON
TYPE	:	

POSSIBLE VALUES	: DIAGS or OTHER
DEFAULT	: NONE

The preliminary checks on the range/set will be done by CRP.

The command execution process contains following modules.

File name	function
nrtrm_entr.c	*main()*
nrtrm_main.c	*nrtrm_main(), nl_rec_validate_ccs(), nl_validate_ccs().*
nrtrm_anls.c	*nrtrm_rslt_ana()*
nrtrm_form.c	*nrtrm_form_cmderr_msg()*
nrtrm_out.c	*nrtrm_out_msg()*
nrtrm_rcvr.c	*nrtrm_rec_rtn()*

4.4.2 Command flow

. CEP will be created by CRP by calling a UNIX system call 'fork' and status_byte & queue_id will be the arguments from CRP to CEP. These two arguments will be taken by the *main()* function in nrtrm_entr.c.

. The *main()* function initiates a file table stucture which keeps all the files cuurently open to the process.

. The function *a_set_def()* will trap all the UNIX signals like interrupts to make CEP to continue its execution.

. The *msg_rcv()* function will receive a message (Ncktcrei22) from CRP from the queue specified by queue_id.

. Calls the *nrtrm_main()* for further execution.

nrtrm_main()

. It validates the opcode and subfield of the command.

. Creates a history buffer .

. It gets the ckt_id as an input from CRP and gets the Card_id using slot_num to Card_id conversion tables. These tables will be maintained in the database.

. It checks the database using the routine *nl_validate_ccs()* with card_id as primary key, for the existence of that card.

. At any point during the execution, if any error like ERR_BIT set etc., occurs, a recovery routine *nrtrm_rec_rtn()* is called.

. The card should present in the database, with equipped & jacked-in state.

. The terminal related to that card will be derived by using card_id and ckt_num as (term_num = 8*card_id + ckt_num -1) Using this terminal number as the primary key for Fterminal_data database will be searched for that terminal status.

. The terminal status should be INS.

. Sends a message to IDUP for validations and updations at IOP1.

. Updations of history buffer fields will be done.

. A record will be updated in Fterminal_data for each terminal being evacuated.

. The file structures Fcard_data & Fterminal data are su_id dependent. The su_id is derived from fs_su_id field by accessing Fsu_data Fsu_data with ns_bm_id as the secondary key.

. If the CEP is run in NORMAL (ONLINE) mode then a message (Ncktcre22) is sent to ACI and CEP goes into wait state and waits for the result message (Ncktcre22) from ACI.

. Depending upon the operating mode and result, the result CEP will send a success or error message (Ncktcrer22) to CRP. This is done by *nrtrm_out_msg()* .

. In the case of error the recovery routine is called. The *nrtrm_rec_rtn()* function depending on the type of error,will call a funnction nrtrm_form_cmderr(), with different parameters.

. The *nrtrm_form_cmderr()* creates an error message (Acmderr01) and sends it to CRP.

. CRP will send this message to OOD which displays the result in a proper format.

4.4.3 Data and File Structures

File structures at IOP :

File structures Fterminal_data, Fcard_data, Fsu-data, Fequip_rec are maintained as ISAM files in the database. The structures are defined earlier.

Data Structures at SU

Data structures Nterminal_info_struct is described in the previous process.

Messages structures :

All the headers of command & result messages are same and are described earlier.

Message from CRP -> CEP

```
typedef struct
     {
          Uchar     bm_num;
          Uchar     rack_num;
          Uchar     phy_frame_num;
          Uchar     slot_num;
     } Strt_posn;

typedef struct
     {
          Uchar     bm_num;
          Uchar     rack_num;
          Uchar     phy_frame_num;
          Uchar     slot_num;
          Uchar     phy_ckt_num;
          Char      dummy;
     } Ten;

typedef struct
     {
          Hdr            hdr ;
          Cmdhdr         cmdhdr ;
          Uchar         dummy_1 ;
          Ten           ckt_id ;
          Uchar          reason ;
```

```
        } Ncktcrei22;
```

Message from CEP -> ACI

```
     typedef struct
          {   Hdr           hdr ;
              Cmdhdr        cmdhdr ;
              Uchar        su_id ;
              Uchar        reason ;
              Uchar        card_id ;
              Uchar        ckt_num ;
          } Ncktcre22;
```

Result Message Formats

Message from ACI -> CEP

```
     typedef struct
          {
              Hdr           hdr;
              Rslthdr       rslthdr;
              Uchar         suc_type;
                            /* SUCCESS or not SUCCESS */
              Uchar         dummy;
          } Ncktcrer22;
```

Result Message from CEP -> CRP

```
     typedef struct
          {
              Hdr           hdr;
              Rslthdr       rslthdr;
              Uchar        dummy_1;
              Ten          ckt_id;
              Uchar        su_id;
          } Ncktcreo22;
```

4.4.4 EXAMPLE

The following example demonstrates the inputting of the the command and corresponding input parameters. This also displays the result of the PUT_TRM_OOS command.

```
U    < PUT_TRM_OOS

S               PUT TERMINAL OUT OF SERVICE
```

```
 ______________________________________________________________
|                                                              |
|            SLOT-CKT = 1-1-1-3-7 & 1-1-1-5-2                  |
|                                                              |
|            REASON   = OTHER                                 |
|                                                              |
|      < E/R/T   = E                                           |
|______________________________________________________________|
```

The result screen will be as follows :

```
                    PUT TERMINAL OUT OF SERVICE

**************************************************
*                                                *
*    Slot and circuit number  : 1-1-1-3-7        *
*                               1-1-1-5-2        *
*    Reason                   : OTHER            *
*                                                *
*                                                *
**************************************************
```

4.5 Put Terminal In Service (PUT_TRM_INS) CEP

4.5.1 Input parameters

1. SLOT-CKT (Slot number)

PARAMETER NAME	: SLOT-CKT of the protocol card
MNEMONIC	: SLOT-CKT
TYPE	: COMPOUND
POSSIBLE VALUES	: 1-1-1-3-1 TO 32-4-4-24-8
DEFAULT	: NONE

The preliminary checks on the range/set will be done by CRP.

The command execution process contains following modules.

File name	function
nctrm_entr.c	*main()*

nctrm_main.c	*nctrm_main(), nl_rec_validate_ccs(), nl_validate_ccs().*
nctrm_anls.c	*nctrm_rslt_ana()*
nctrm_form.c	*nctrm_form_cmderr_msg()*
nctrm_out.c	*nctrm_out_msg()*
nctrm_rcvr.c	*nctrm_rec_rtn()*

4.5.2 Command flow

. CEP will be created by CRP by calling a UNIX system call 'fork' and status_byte & queue_id will be the arguments from CRP to CRP. These two arguments will be taken by the *main()* function in nctrm_entr.c.

. The *main()* function initiates a file table stucture which keeps all the files currently open to the process.

. The function *a_set_def()* will trap all the UNIX signals like interrupts to make CEP to continue its execution.

. Performs checks on the arguments.

. The *msg_rcv()* function will receive a message (Ncktcrei23) from CRP from the queue specified by queue_id.

. Calls the *nctrm_main()* for further execution.

nctrm_main()

. It validates the opcode and subfield of the command.

. Creates a history buffer.

- It gets the ckt_id as an input from CRP and gets the Card_id using slot_num to Card_id conversion tables. These tables will be maintained in the database.
- Checks the database using the routine *nl_validate_ccs()* with card_id as primary key, for the existence of that card.
- At any point during the execution, if any error like ERR_BIT set etc., occurs, a recovery routine *nctrm_rec_rtn()* is called.
- The card should present in the database, with equipped & jacked-in state.
- The terminal related to that card will be derived by using card_id and ckt_num as (term_num = 8*card_id+ckt_num -1) Using this terminal number as the primary key for Fterminal_data database will be searched for that terminal status.
- The terminal status should be OOS.
- Sends a message to IDUP for validations and updations at IOP1.
- Updations of history buffer fields will be done.
- A record will be updated in Fterminal_data for each terminal being put into service .
- The file structures Fcard_data & Fterminal data are su_id

dependent. The su_id is derived from fs_su_id field by accessing Fsu_data Fsu_data with ns_bm_id as the secondary key.

. If the CEP is run in NORMAL (ONLINE) mode then a message (Ncktcre23) is sent to ACI and CEP goes into wait state and waits for the result message (Ncktcre23) from ACI.

. Depending upon the operating mode and result, the result CEP will send a success or error message (Ncktcrer23) to CRP. This is done by *nctrm_out_msg()* .

. In the case of error the recovery routine is called. The *nctrm_rec_rtn()* function depending on the type of error,will call a funnction nctrm_form_cmderr(), with different parameters.

. The *nctrm_form_cmderr()* creates an error message (Acmderr01) and sends it to CRP.

. CRP will send this message to OOD which displays the result in a proper format.

4.5.3 Data and File Structures

File structures at IOP :

File structures Fterminal_data, Fcard_data, Fsu-data, Fequip_rec are maintained as ISAM files in the database. The structures are defined earlier.

Data Structures at SU

Data structures Nterminal_info_struct is described in the previous process.

Messages structures :

All the headers of command & result messages are same and are described earlier.

Message from CRP -> CEP

```
     typedef struct
          {
               Uchar    bm_num;
               Uchar    rack_num;
               Uchar    phy_frame_num;
               Uchar    slot_num;
          } Strt_posn;

     typedef struct
          {
               Uchar    bm_num;
               Uchar    rack_num;
               Uchar    phy_frame_num;
               Uchar    slot_num;
               Uchar    phy_ckt_num;
               Char     dummy;
          } Ten;

     typedef struct
          {
               Hdr            hdr ;
               Cmdhdr         cmdhdr ;
               Uchar         dummy_1 ;
               Ten           ckt_id ;
          } Ncktcrei23;
```

Message from CEP -> ACI

```
     typedef struct
          {
               Hdr            hdr;
               Cmdhdr         cmdhdr;
               Uchar          su_id ;
               Uchar         card_id ;
          } Ncktcre23;
```

Result Message from ACI -> CEP

```
     typedef struct
          {
            Hdr                hdr;
            Rslthdr            rslthdr;
            Uchar              suc_type;
                             /* SUCCESS or not SUCCESS */
            Uchar              dummy;
        } Ncktcrer23;
```

Result Message from CEP -> CRP

```
    typedef struct
        {
            Hdr               hdr;
            Rslthdr           rslthdr;
            Uchar            dummy_1;
            Ten              ckt_id;
            Uchar            su_id;
        } Ncktcreo23;
```

4.5.4 EXAMPLE

The following example demonstrates the inputting of the the command and corresponding input parameters. This also displays the result of the PUT_TRM_INS command.

```
U     < PUT_TRM_INS

S                PUT  TERMINAL INTO  SERVICE

 _______________________________________________
|                                               |
|                                               |
|           SLOT-CKT = 1-1-1-3 & 1-1-1-5        |
|                                               |
|                                               |
|     < E/R/T    = E                            |
|                                               |
|_______________________________________________|
```

The result screen will be as follows :

```
                 PUT  TERMINAL INTO  SERVICE

*************************************************
*                                               *
*      Slot ckt  : 1-1-1-3 & 1-1-1-5            *
*                                               *
*************************************************
```

APPENDIX

NAMING CONVENTIONS

This 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

For communication to duplex iop type will have to be identified .

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	

BIBLIOGRAPHY

1. Specifications for National application of CCITT #7 (Common Channel Signalling System part1)

2. CCITT Red Book Vol VI