D814 SYSTEM SOFTWARE MANUAL

Revision: 3

CODEX CORPORATION
July, 1981

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd.)

TABLE OF CONTENTS (Cont'd.)

Section/Title Page

TABLE OF CONTENTS (Cont'd.)

TABLE OF CONTENTS (Cont'd.)

TABLE OF CONTENTS (Cont'd.)

TABLE OF CONTENTS (Cont'd.)

1. INTRODUCTION

1.1 Purpose/Scope

This document describes the D814 software system. Its purpose is to identify all the elements of the software system, to describe their function, and to define the interfaces/relationships between the elements.

1.2 Software System Structure

The D814 software system is structured into several hierarchial levels. The names and definitions of these levels are given below. The D814 system software will be described in terms of these structural levels.

D814 System Software: All software which resides in the D814 product.

Subsystem: A unique collection of software which resides in one place. Subsystems are made up of modules. The Mainframe software and the I/NP software are examples of subsystems.

Module: A unique collection of software which performs a single function and resides in one place. Modules may be used in more than one subsystem. The configuration control software and IPOS are examples of modules.

Submodule: A unique collection of software which performs a logical sub-division of a single system function and resides in one place. The IPOS queue utility and the BIC FIFO control software are examples of submodules.

Routine: A collection of instructions to perform a single operation. Routines have inputs, perform operations, and give outputs. The IPOS enqueue routine and the addressed packet router are examples of routines.

System Data Structure: A data structure common throughout the D814 system software. The addressed packet format is a data structure at this level.

Subsystem Data Structure: A data structure common to several modules in a subsystem.

Module Data Structure: A data structure common to several routines in a module.

1.3 Document Organization

An overview of this document's organization and content is given below:

Section 2: D814 Hardware Summary

This section describes the basic elements of the D814 hardware. Its intent is to familiarize the reader with the D814 hardware so that he can better understand the environment the software runs in.

Section 3: Software Subsystem Structure

In this section, the D814 system is broken down into its component sub-systems. System data structures, subsystem functions, and interfaces between subsystems are defined.

Section 4: Firmware

The PROM's contained in the D814 are defined.

Section 5: Mainframe Module Definitions

Each mainframe software module is defined as to its general algorithms, data structures, and external interfaces.

Section 6: Intelligent Port Module Definitions

Each IP software module is defined as to its general algorithms, data structures, and external interfaces.

2. HARDWARE ENVIRONMENT

2.1 6000 Mainframe Environment

The fundamental hardware elements of a D814 node consist of a modular interconnection of mainframe, power supply, display and control panel, and port nest. Up to 96K bytes of memory can be included, in increments of 16K. A port nest contains the intelligent nest interface card (I/NIC) and some customer designated mix of intelligent network and terminal ports.

2.1.1 Component Parts

Microprocessor

The D814 is a general purpose data communications multi-processor computer configured around the Motorola M6800 Microprocessor. Since the M6800 has no explicit I/O instructions, additional logic has been constructed to augment the basic instruction set, facilitate a multiprocessor shared memory environment, and provide bootstrap, control panel primitives, and control interrupts.

Mainframe Modules

The mainframe system is organized around a common bus system with modular subsystems attached to it. System configurations differ principally by the numbers and types of these modules, as well as in the port nest modules and system software.

Master Controller

This module contains the system master clocks and memory refresh logic; the necessary bus and interrupt arbitration logic to enable other modules to use the bus system; and the master I/O logic for the bus to the port nests, which are driven by this module. The Master Controller occupies two logic cards and is organized around an Intel 3000 series microcontroller.

Some primitives implemented by the Master Controller are:

Read Processor Status
Read Panel Keys
Fork a Task
Terminate a Task
Interrupt Enable/Disable
Control Processor
Read Off-Line (Configuration) Memory

Write Off-Line (Configuration) Memory
Test Port
Create Port Request
Read Option PROM (Node Chip)

Processor Card

This module contains a Motorola M6800 Microprocessor, associated bus access and interrupt logic, and a small amount of local ROM.

RAM Memory

This module contains 16K x 8 bits of semiconductor memory for program, data and buffer stores for the system.

Option Card

This module is used to contain hardware associated with certain options and various system memories. Two such functions on the options card are:

1. Control panel logic
2. Memory for storing the network and terminal configuration information
3. Options PROM (Node Chip)

Customer Configuration/Reconfiguration

The information necessary to specify a customer's initial D814 system consists of configuration data, standard software, and software to support customer purchased options. These three types of information are stored in non-destructive (battery backup power) memory.

Configuration Memory

The configuration memory contains the data necessary to specify the D814 as it is configured for the customer. This data includes such information as number of ports, port types and characteristics, routing parameters, default parameters for threshold monitoring, etc. Whenever the D814 is boot-loaded and brought on-line, its software will use the data in the configuration memory to configure the system.

Port Nest Modules

The port nest is driven by an I/O bus cable from the mainframe. It, in turn, via the Intelligent Nest Interface Card (I/NIC), redrives the I/O bus to the next port nest, if any. In later versions of the D814, the port nest may be interfaced to two mainframes by a Dual Mainframe Interface Card (DMIC) for back-up redundancy use.

Front Panel

The front panel consists of a data entry keyboard, a self-scan display, an array of processor status indicator lights, and a key switch.

2.1.2 Special Features

Hardware Data Spaces

Each software task in the 6000 is given an area of memory, known as its data space, which is mapped into locations X'0000' through X'001F'. These data spaces are assigned by the master controller, which maintains the base registers pointing to the data spaces for the various tasks. There are a maximum of 64 data spaces. The memory mapped into the data space is actual RAM memory and may be accessed directly if the proper base address is known.

Lock Bytes

The 6000 has an area of memory between X'400' and X'4FF' which is known as the lock byte area. This area is special memory which clears to zero when it is read. The purpose of the lock byte area is to synchronize tasks in different processors. One processor reads the lock byte. If the contents are non-zero, it has obtained the resource it is requesting. If the contents are zero, it must wait. When it is through with the resource, it writes some non-zero value into the lock byte to release the resource.

2.1.3 Memory Map

The following is a definition of the address space allocation for software in the D814 mainframe. The mainframe has an address space of 96K bytes. This includes 32K of high bank memory addressable as locations X'8000'-X'FFFF' when the high bank has been activated and 32K of low bank memory addressable as locations X'8000'-X'FFFF' when the low bank is active. On system power-up and during downline load the low memory bank is active. After downline load and during normal system operation the high bank is active and the low bank is unused. The 6000 mainframe includes a bank switch which will allow addresses X'8000' through X'FFFF' to refer to either of the two 32K memory segments.

```
        Start of RAM memory
____
   0    Data space
   |            |
   |            |
  1F            V
____
  20    Local Storage - Memory mapped to one of up to eight 16-byte local
   |    storage areas fron X'0' to X'7F' in physical RAM.  Each processor
   |    has its own dedicated local storage.
   |            |
  2F            V
____
  30    'Trigger' addresses for Master Controller commands - Mapped to
   |    locations in Local Storage and used to activate Master Controller
   |    functions
   |            |
  7F            V
____
  80    Page 0 (defined in file OF$>PGO)
   |    System parameters and scratch storage for mainframe modules
   |            |
  FE            V
____
  FF    Level Request Flags - Each bit corresponds to a priority level and,
        if set, implies some hardware device needs service or some software
        task is pending at that level

____
 100    Self-scan message area
   |            |
 11F            V
____
 120    System Area
   |    More system parameters and scratch area
   |            |
 1FF            V
____
 200    Table of Port Control Block (PCB) addresses
   |            |
 3FF            V
____
 400    Lock bytes
   |            |
 7FF            V
____
 800    Level Queue Area
   |            |
   |            |
 8FF            V
____
 900    Volatile Memory - Data spaces, dynamic buffers, and memory
   |    allocated at system initialization time
   |            |
3FFF            V
____
4000    6000 Program Code - Continues in high bank memory
   |            |
7FFF            V
____
```

```
        Start of HIGH BANK RAM memory
8000    Continuation of system executable code
   |            |
   |            V
MISC$CODEND:HERE-1  Address of last byte of 6000 program code
MISC$CODEND:HERE
   |    Volatile Memory - Dynamic buffers and fixed-size data structures
   |    for system modules
   |            |
FFF7            V
FFF8    Interrupt vectors
   |            |
FFFF            V
        End of high bank memory
```

```
        Start of LOW BANK ROM memory
8000    IPL code, downline load bootstrap code, and debugger, all in ROM
   |            |
FFF7            V
FFF8    Interrupt vectors
   |            |
FFFF            V
        End of low bank memory
```

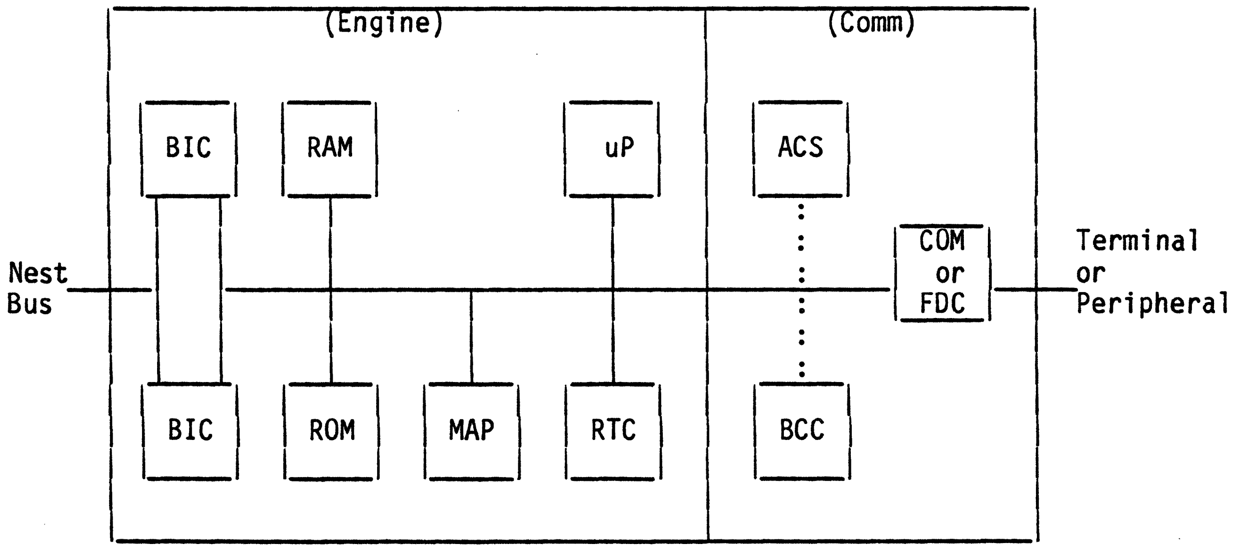
2.2 Intelligent Port Environment

The Intelligent Port (IP) is a microprocessor based I/O or peripheral driver for the D814 system.

Internally, it consists of a bus interconnecting the following hardware elements:

1. A microprocessor
2. RAM memory
3. PROM memory
4. A real time clock
5. 2 Bus Interface Chips (BIC)
6. Communications or floppy disk controller chips
7. A map register
8. An auxiliary control signal register, if present
9. A checksum calculation chip, if present

The BIC is the IP's interface to the D814 mainframe and the communications of floppy disk controller chip is its interface to the I/O device or peripheral. The port as a whole will be divided into 2 sections: an engine (composed of the microprocessor, RAM, PROM, map register, BIC's, and real time clock), and the card which makes each port the specific type of communications device that it is, which is called the Comm card. The Comm card contains the communications chip or device controller, as well as any support circuitry needed, such as Auxiliary Control Signal register, checksum (BCC) calculator, etc. BIC #0 is used for IPL and addressed packets; BIC #1 is used as a data path by TPs and NPs.



The mainframe is capable of deactivating, activating, and resetting an IP by sending commands to BIC #0. The reset indicates the type of reset to be performed, so that the IP can perform a predefined function.

Programs that execute in the IP are loaded from the mainframe via BIC #0. The loading is initiated by using one of the resets listed below.

2.2.1 IP PROM

There are 4 types of resets on the IP, each type causes one of several PROM based routines to be executed. These are descnibed in detail in Section 4.

The PROM in the IP will contain the following routines:

1. On Reset 0 (power-up or nest reset):

    a. Run diagnostic routines
    b. Go to debugger if present
    c. Set port bit and turn off diagnostic LED

2. On Reset 1:

    a. Size RAM
    b. Output Port-ID, RAM size and Processor-ID to BIC
    c. Go to IP Program Load

3. On Reset 2:

    a. Enter BIC loopback test

4. On Reset 3:

    a. Return failure information.

2.2.2 D814 IP Memory Map

The following is a definition of the standard address space allocation for IP programs in the D814 system. All addresses are in Hex. XXXX, YYYY, and ZZZZ are variable depending on the software and hardware requirements for the particular IP. For all currently planned IPs, ZZZZ will be 3FFF, 7FFF, or BFFF.

```
           Start of RAM memory
      0    Mapped Area
      |           |
      V           |
      F           V
     10    Page Zero Variables
      |           |
      V           |
     FF           V
    100    Volatile Memory
      |           |
      V           |
    3FF           V
    400    Program Checksum (Range X'402' to XXXX-1)
    401
    402    Program ID
    403
    404    Program Revision Number
    405
    406    Address of the end of program executable code (XXXX-1)
    407
    408    Entry Point of Program
      |           |
      |    Program Code
      |           |
      V           |
XXXX-1            V
XXXX       Program Permanent Storage (IP$CODEND:HERE)
      |           |
      V           |
YYYY-1            V
YYYY       Program Buffers
      |           |
      V           |
ZZZZ-1            V
ZZZZ       Last byte of RAM (reserved by system)
C000       4K Empty or External Diagnostics
      |           |
      V           |
CFFF              V
D000       4K Empty or Peripheral Diagnostics ROM
      |           |
      V           |
DFFF              V
E000       4K Diagnostic ROM AND M6800 Vectors
      |           |
      V           |
EFFF              V
```

```
F000    3.75K Empty or Debugger or External Diagnostics ROM
   |     (if Debugger, scratchpad RAM from FE7F to FEFF
   V              |
FEFF              V
FF00    224 byte Remote I/O
   |              |
   V              |
FFDF              V
FFE0    16 byte Engine I/O
   |              |
   V              |
FFEF              V
FFF0    16 byte ROM Vectors (mapped to EFF0 to EFFF)
   |
   V
FFFF
```

3. SOFTWARE SUBSYSTEMS STRUCTURE

The D814 functions as a node in data communications network. As can be seen from Section 2, the D814 hardware is architected as a mainframe (central processing element), controlling a hierarchical I/O bus, and up to 127 intelligent ports (satellite processors) which interface to the I/O bus. All external devices and communications lines connect to the D814 via interfaces on the intelligent ports. The D814 software is structured around this hardware architecture.

As mentioned in the introduction, the software system is structured at its highest level by subsystems, subsystem interfaces, and global system data structures. The D814 system consists of PROM subsystems, a Mainframe (MF) software subsystem and a number of intelligent port (IP) software subsystems. Only one IP subsystem exists per physical intelligent port. The intelligent port subsystems can be divided into five basic classes:

* Control Port Subsystems
    1. Intelligent Control Terminal Port (ICTP)
    2. Intelligent Floppy Disk Controller (IFDC)
* Network Link Subsystems
    1. Intelligent Network Port (INP)
    2. Intelligent Group Band Network Port (IGBNP)
* Single Threaded Communications Port Subsystems
    1. Intelligent Spoofed Synchronous Terminal Port - 2780/3780 BSC version (ISSTP-BSC)
    2. Intelligent Bit Oriented Protocol Port (IBOP)
* Multi Threaded Communications Port Subsystems
    1. Intelligent Multichannel Synchronous Terminal Port (IMSTP)
    2. Intelligent Multichannel Asynchronous Terminal Port (IMATP)
    3. Intelligent Mux Protocol Port (IMXP)
* Message Communications Port Subsystems
    1. Intelligent Data Gram Port (IDGP)

Figure 4-1 illustrates the relationship between the mainframe software subsystem and all intelligent port software subsystems. Note that there is no direct interface between IP subsystems; all intercommunications between IP subsystems must be routed through the MF subsystem.

The purpose of the PROM subsystems is to load the appropriate operational software subsystem into RAM for execution in the mainframe and IPs.

D814 SUBSYSTEM RELATIONSHIPS

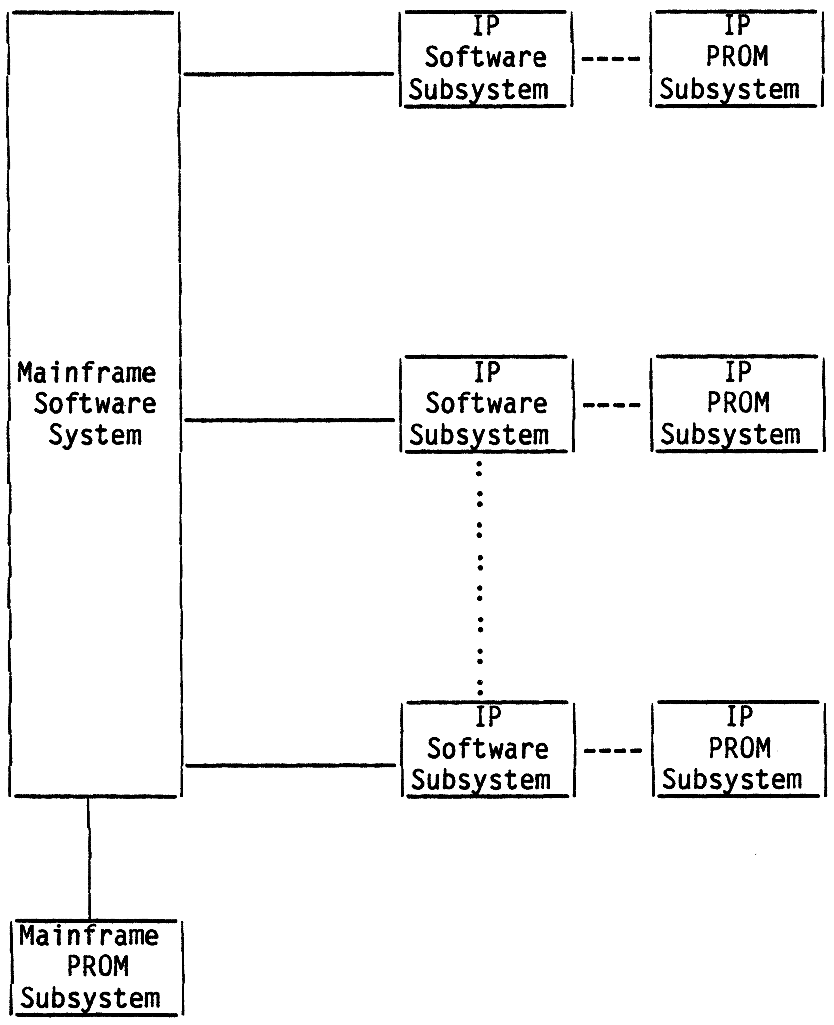


Figure 3-1

The interface between the MF software subsystem and each IP software subsystem physically consists of the D814 master controller, the hierarchical I/O bus, and the IP's BIC(s). Each IP must have a BIC#0, and may have a BIC#1 depending on the type of IP it is.

IP's operate in one of three modes:

1. Diagnostic Mode
2. Program Load Mode
3. Normal Mode

In diagnostic mode, the IP is under control of its internal PROM and performs BIC loopbacks for diagnosability of IP function from the MF software subsystem. In program load mode, the IP is also under control of its internal PROM. In this mode the IP will load a software subsystem from the MF into its onboard RAM via BIC#0. On command from the MF, the IP will leave program load mode and enter normal mode by starting execution of the loaded software subsystem.

When an IP is running a D814 software subsystem in normal mode, BIC#0 is used to support an addressed packet interface between the MF software subsystem and the IP software subsystem. BIC#1 is required only for the following IP subsystem classes: network link subsystems, single threaded communications port subsystems, and multithreaded communications port subsystems. For each of these subsystem classes, BIC#1 is used to support a high speed data interface between the IP software subsystem and the MF software subsystem.

3.1 Software Subsystems

3.1.1 Mainframe PROM Subsystem (MFPROMSS)

The MFPROMSS resides on a PROM card in the D814 mainframe. Its function is to load the mainframe RAM with the mainframe software subsystem which is the operational software for the D814 mainframe. The MFPROMSS gets this software from a local floppy disk or via an I/NP from an adjacent node. This subsystem is detailed in Section 4.1.

3.1.2 IP PROM Subsystem (IPPROMSS)

The IPPROMSS resides in a PROM chip on all IP's. It is activated whenever the D814 mainframe issues an IP master reset to the IP. Its basic function is to load IP software subsystems into IP RAM for execution. For all IP's except the I/FDC, software is loaded from the mainframe via BIC#0. For the I/FDC, a bootstrap program is loaded from the mainframe and the I/FDC loads its software subsystem directly from the disk. This subsystem is detailed in Section 4.2 and 4.3.

3.1.3 Mainframe Software Subsystem (MFSS)

The MFSS consists of a number of software modules. The mainframe is the central controller for a D814 node. It performs the following functions:

1. Addressed Packet Control
2. Statistics and Monitoring
3. Configuration Control
4. Node Path, Routing, and Congestion Control
5. IP Program Load Control
6. Network Link Frame Assembly and Disassembly
7. Multithreaded Port Frame Assembly and Disassembly
8. Network Boot Control
9. Down Line Loading of Adjacent Nodes
10. Front Panel Control
11. Node Level System Service Support

3.1.4 I/CTP Software Subsystem (ICTPSS)

The ICTPSS runs on an I/CTP module which consists of a 48K IP engine card and a Control Terminal Card (CTC). The subsystem is loaded to the I/CTP from the mainframe using the IPPROMSS resident on the IP engine card.

The I/CTP running the ICTPSS implements the man-machine interface between the D814 operator and the D814 network. A CRT is the primary human interface and an optional printer using asynchronous RS232 protocol can be used to log hardcopy reports. The I/CTP provides the following functions:

1. Network Configuration Control
2. Network Report Monitoring
3. Network Statistics Collection
4. System Service Control Point
5. Inter-operator Communications
6. Device Control via RS232 Interface

3.1.5 I/NP Software Subsystem (INPSS)

The INPSS runs on the I/BIT module (card) with 48K of RAM. The software is loaded to the IP from the mainframe using the IPPROMSS resident on the I/BIT card.

The I/BIT module running the INPSS implements the low speed (<19.2K bps) network link functions for the D814. This function includes:

1. Link Initialization.

2. Receiving internal network link frame from D814 MF and sending them over the network link.

3. Receiving external network link frames from the network link and sending them to the D814 MF.

4. Managing the network link ARQ protocol.

5. Measuring and reporting statistics about the communications link performance and loading characteristics.

6. Responding to system service commands.

3.1.6 I/SSTP-BSC Software Subsystem (ISSTPSS-BSC)

The ISSTPSS-BSC runs on the I/BYTE module (card) with 48K of RAM. It is loaded to the IP from the D814 mainframe using the IPPROMSS resident on the I/BYTE card.

The I/BYTE module executing the ISSTPSS-BSC implements the RS232 synchronous communications interface function for the Binary Synchronous Communications protocol for one COMM line on the D814. This function includes:

1. Receiving/transmitting data via RS232 interface.
2. Monitoring/driving control signal lines.
3. Call management.
4. Flow control.
5. Adaptive data compression.
6. Throughput enhancement via BSC protocol intervention.
7. Configuration management for interface.
8. Measuring and reporting COMM line statistics.
9. Responding to system service commands.

3.1.7 I/SSTP-HASP Software Subsystem (ISSTPSS-HASP)

The ISSTPSS-HASP runs on the I/BYTE module (card) with at least 48K of RAM. It is loaded to the IP from the D814 mainframe using the IPPROMSS resident on the I/BYTE card.

The I/BYTE module executing the ISSTPSS-HASP implements the RS232 synchronous communications interface function for the HASP protocol for one COMM line on the D814. This function includes:

1. Receiving/transmitting data via RS232 interface.
2. Monitoring/driving control signal lines.
3. Call management.
4. Flow control.
5. Adaptive data compression.
6. Throughput enhancement via HASP protocol intervention.
7. Configuration management for interface.
8. Measuring and reporting COMM line statistics.
9. Responding to system service commands.

3.1.8 I/BOP Software Subsystem (IBOPSS)

The IBOPSS runs on the I/BIT module (card) with 16K of RAM. It is loaded to the IP from the D814 mainframe using the IPPROMSS resident on the I/BIT card.

The I/BIT module executing the IBOPSS implements the RS232 synchronous bit oriented communications interface function for one COMM line on the D814. This function includes:

1. Receiving/transmitting data via RS232 interface.
2. Monitoring/driving control signal lines.
3. Call management.
4. Flow control.
5. Adaptive data compression.
6. Configuration management for interface.
7. Measuring and reporting COMM line statistics.
8. Responding to system service commands.

3.1.9 I/MATP Software Subsystem (IMATPSS)

The IMATPSS runs on a 48K byte I/ENG card and supports up to 4 QBYTE cards to interface up to 16 asynchronous RS232 devices to a D814. It is loaded to the IP.

The IMATPSS executing on an I/ENG card implements a multi-channel asynchronous interface function for the D814.

3.1.10 I/MXP Software Subsystem (IMXPSS)

The IMXPSS runs on a 48K I/BIT card and supports one Codex "Mux Port" interface as defined in the Codex Multiplex Protocol Specification; as well as the Codex Single Line Interface as defined in the Codex SLI Functional Interface Specification. Communication is bit synchronous via RS232 interface.

The IMXPSS executing on the I/BIT implements a multi-channel synchronous interface function for the D814. This function includes communications with Codex 6010, 6030, and 6040 products as well as Codex FEP's.

3.1.11 I/FDP Software Subsystem (IFDPSS)

The IFDPSS runs on a 48K I/ENG and a Floppy Disk Controller card (FDC). The subsystem is loaded to the I/FDP from an attached floppy disk drive on command from the mainframe via BIC #0. The mainframe accomplishes this by loading a jump instruction in normal load format (see Section 3.2.1). This jump instruction transfers control to a ROM on the FDC card which loads the correct software from a specified drive into the I/FDP RAM memory. For further details see Section 4.3.

The I/FDP running the IFDCSS implements the D814 disk file system and file system AP protocol. A single I/FDP may control up to 4 disk drives. The I/FDP provides the following functions:

1. File management.
2. File system protocol control.
3. Disk control.
4. Statistics and monitoring.
5. Configuration management for disk interface.

3.1.12 I/DGP Software Subsystem (IDGPSS)

The IDGPSS runs on an I/CTP module which consists of a 48K I/ENG card and a Control Terminal Card (CTC). The subsystem is loaded to the I/DGP from the mainframe using the IPPROMSS resident on the I/ENG.

The I/DGP running the IDGPSS implements the "datagram" function and the man-machine interface necessary to implement it. A CRT is the primary human interface and an optional printer using asynchronous RS232 protocol can be used to log hardcopy listings of messages. The I/DGP provides the following functions:

1. Inter-operator communications.
2. Statistics and monitoring.
3. Device control via RS232 interface.

3.2 Subsystem Interfaces

D814 software subsystem interfaces fall into three categories:

1. Program Load
2. Address Packets
3. High Speed Data Streams

3.2.1 Program Load Interface

The program load interface is used for two functions. First, it is used to load IP software from the mainframe into an IP. Secondly, it is used to load the mainframe when a downline load of the mainframe is done.

BIC #0 is always used for the program load interface. The format of the program load data is as follows:

Load Header:

Bytes 0 - 1:	Start address (0 if not last load block)
Bytes 2 - 3:	Load address
Bytes 4 - 5:	Byte count of load block including header.

Data:

Bytes 6 - n:	Object code in binary.

Checksum:

Bytes n+1 - n+2:	16 bit end-around carry checksum of load block including header.

It is up to the sending device to block the data into this format.

3.2.2 Addressed Packet Interface

Addressed Packets (APs) are the primary method of command and control within the D814. They provide a flexible method of communicating information between any two modules in a D814 network.

BIC #0 is used for the AP interface after the program load function is complete. The format of an AP is as follows:

Byte	Contents
0	Total length of packet in bytes
1	Packet Destination Node Number
2	Packet Destination Port Number
3	Packet Destination Module Number
4	Packet Source Node Number
5	Packet Source Port Number
6	Packet Source Module Number
7-n	Packet Data

In bytes 2 and 5, port zero is the mainframe at the specified node. In byte 1, if the high order bit (X'80') is set, the packet has experienced an error condition and is being returned to the sender (source and destination having been interchanged). When an error is flagged, a byte containing an error code is appended to the end of the packet and the byte count (byte 0) is incremented by one.

3.2.3 High Speed Data Interface (HSDI)

The high speed data interface is used for transferring "user data" between IP's and the D814 mainframe for single-threaded, multi-threaded, and network link communications subsystems. There are three basic data formats used, one for each class of communications subsystem. BIC #1 is always used for this interface.

3.2.3.1 Single Threaded High Speed Data Interface (STHSDI)

The STHSDI is stream oriented and passes data in 4-bit "chunks" called nibbles. Nibbles are segments of encoded user data. Since the BIC is byte oriented transfers across the BIC may be packed. The following coding of the STHSDI is the system standard.

X'00'	Not allowed.
X'0a'	Single nibble ($a \neq 0$)
X'ab'	Two nibbles (a through b, $a \neq 0 \neq b$)
X'b0'	System In-channel-signal ($b \neq 0$)

Note that a nibble may not take the value 0 (zero). This is a consequence of the in-channel-signal (ICS) scheme used in the D814. ICS's are used for system control of virtual channels (paths). They are not coded and can be interpreted by any entity which processes a data stream. Their detailed use will be explained in later sections of this document.

3.2.3.2 Multi Threaded High Speed Data Interface (MTHSDI)

The MTHSDI is a multiplexed stream of STHSDI data. It is used to send multiple single threaded data streams through one BIC to interface with a multi-threaded terminal port. The following format is used for the MTHSDI data:

```
                       Slot
 _______________________________________________________________
  . . . |   A   |MTEOS| d | d | d | ... | d | A |MTEOS| d | d | ...
 _______________________________________________________________
```

The data stream is composed of a series of "slots" each of which contains address and data for one single-threaded data stream. Each slot begins with an A (address) field containing the thread number used by the port to identify the single-threaded data stream. Dot data follows the A field and is in the same format as the STHSDI. Each slot is terminated by a special Multi-threaded End-of-Slot (MTEOS) ICS, defined as X'FO', which is not allowed to occur in the STHSDI format.

3.2.3.3 Network Link High Speed Data Interface (NLHSDI)

The NLHSDI is a multiplexed stream interface standard for sending multiple single threaded data streams, addressed packets, and control messages to an I/NP for transmission over a network link. Network link transmission uses a frame (block) based HDLC-like protocol. The NLHSDI is also frame based with three possible frame types as follows:

1. Data Frame
2. Address Packet Frame
3. Control Message Frame

All NLHSDI frames have a common general format:

```
 _________________________________________________________
 . . . . . | EOF | Frame Data | FTI | EOF | . . . >
 _________________________________________________________
           | <-- NLHSDI Frame ---|->|     |
                                 |        '--> "End-of-Frame" Character
                                 '---------> "Frame-Type-Identifier"
```

Each frame begins with a "Frame-Type-Identifier":

FTI	Frame Type
X'40'	Data Frame
X'80'	Addressed Packet Frame
X'CO'	Control Message Frame

The EOF character for all NLHSDI frame types is X'01'.

3.2.3.3.1 Data Frame Format

The NLHSDI format for data frames is as follows:

```
|X'01'|EOS| d | d | d |A|...| EOS | d | d | d |A|X'80'|X'01'| ...>
   |                                              |         |
   |      Slot N ...     Slot 1                   |         '-> EOF Prev. Frame
   '--EOF                                         '--> FTI = Data Frame
```

Slot Format:

```
 . . . | A | EOS | d | d | d | A | EOS |  >
             |                 |    |
             |      Data       |    '--End-of-Slot ICS
             |                 |
             '--End-of-Slot    '-------Slot Address (2≤A≤255)
```

Each slot is address and data for one single threaded data stream. The format of the data segment of each slot is the same as the STHSDI. Three "End-of-Slot" (EOS) ICSs are defined for network link data streams. They are:

X'F0'	=	End slot normal
X'E0'	=	End slot & kill channel (Failure)
X'D0'	=	End slot switch channel

It should be noted that the MTEOS used in the MTHSDI and the normal EOS used in the NLHSDI have the same value. This causes no confusion since neither the MTEOS nor the normal EOS is allowed to occur in the STHSDI format.

Slot addresses may not have the values 0 and 1. Therefore, the EOF value of X'01' after the last slot cannot be confused with the beginning of a new slot.

3.2.3.3.2 Addressed Packet Frame Format

The NLHSDI format for Addressed Packet Frames is as follows:

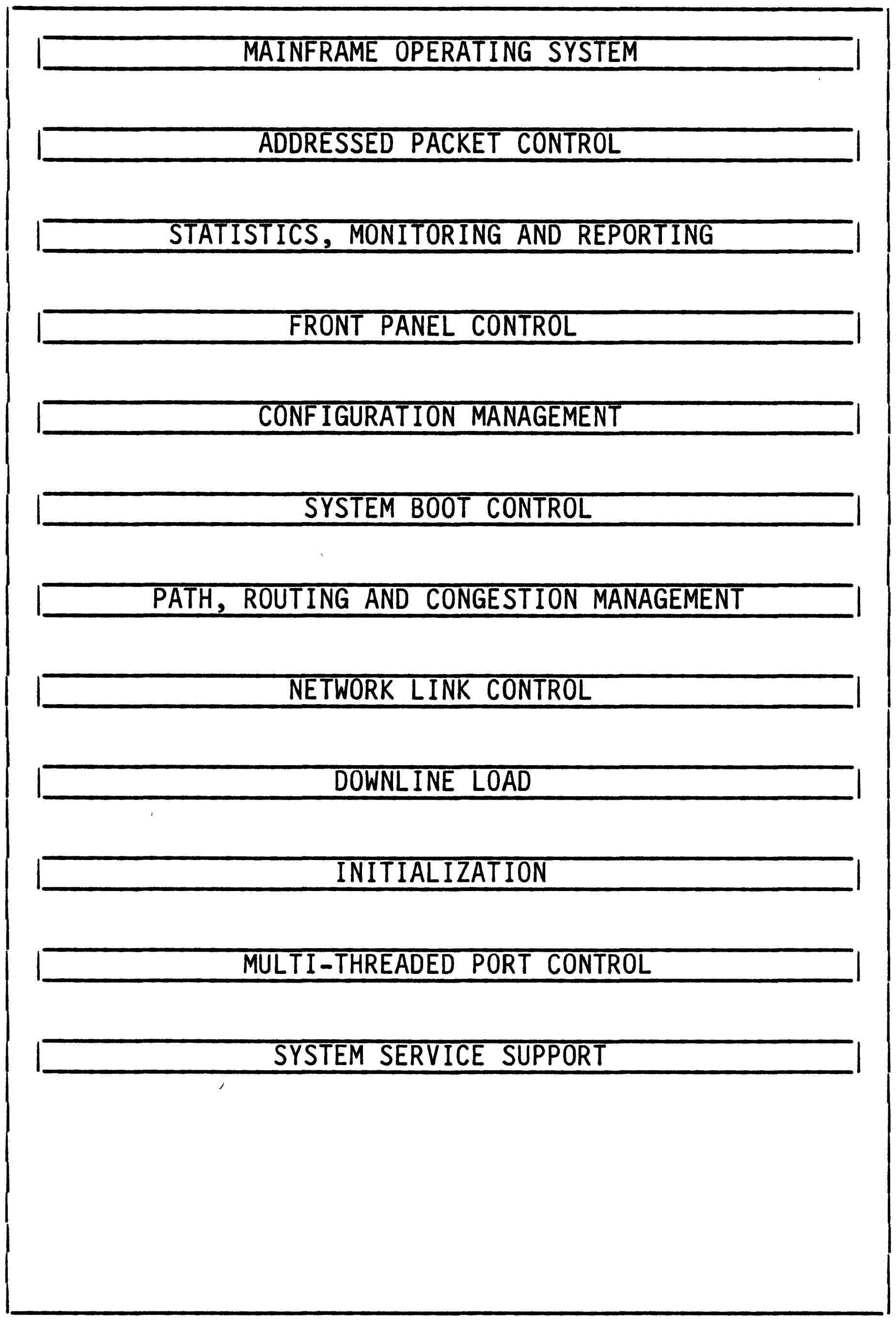
```
   X'01' |... d d d | Ln|...| d | d | d | d | L1| X'80' | X'01' |....>
  |                                                        |
  |     Addressed Packet ... Addressed Packet              '-> EOF Previous
  |                      n                  1                    Frame
  '--> EOF
```

The addressed packet format is as described in 4.2.2. Note that L = AP length = 1 is illegal for an AP and thus is a valid EOF character.

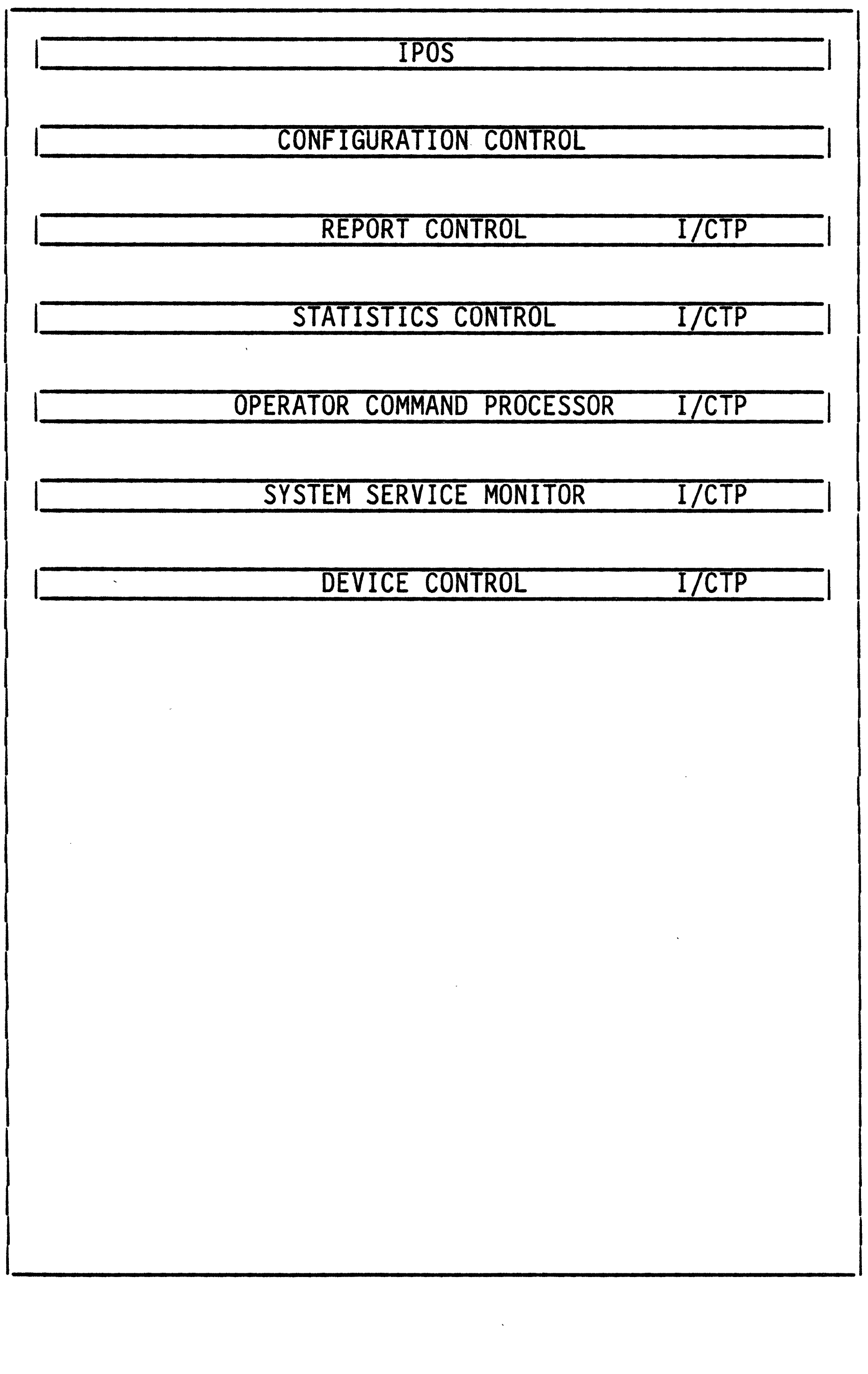
3.2.3.3.3 Control Message Frame Format

The NLHSDI format for Control Message Frames is identical to the format for Addressed Packet Frames. Each control message must be greater than two so the EOF = X'01' is still valid. Control messages are special node-to-node messages that are handled at high priority. Their detailed use and format will be explained later in the section on the MFSS data structures.

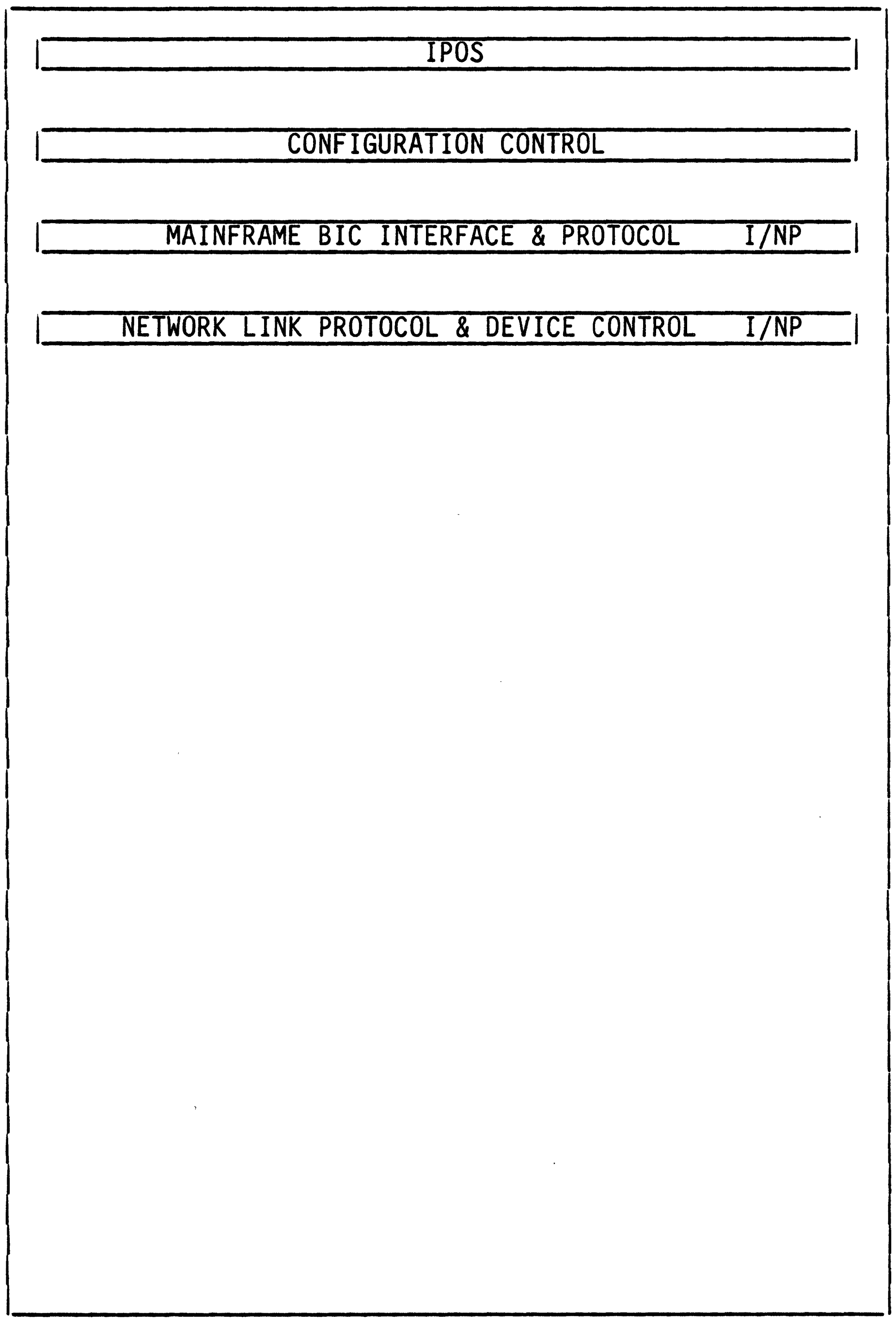
MAINFRAME SUBSYSTEM



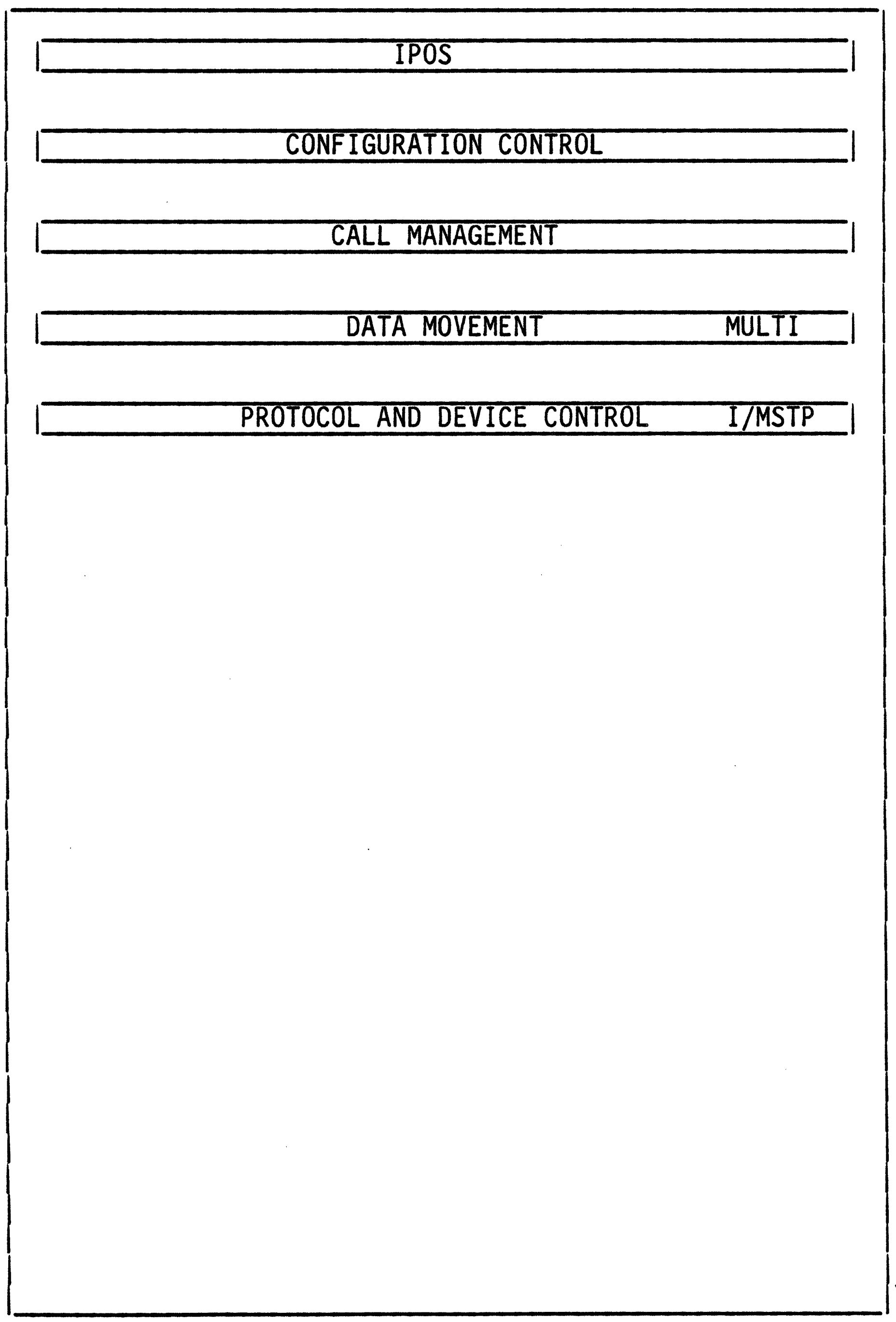
I/CTP SUBSYSTEM



I/NP SUBSYSTEM


IPOS
CONFIGURATION CONTROL
MAINFRAME BIC INTERFACE & PROTOCOL I/NP
NETWORK LINK PROTOCOL & DEVICE CONTROL I/NP

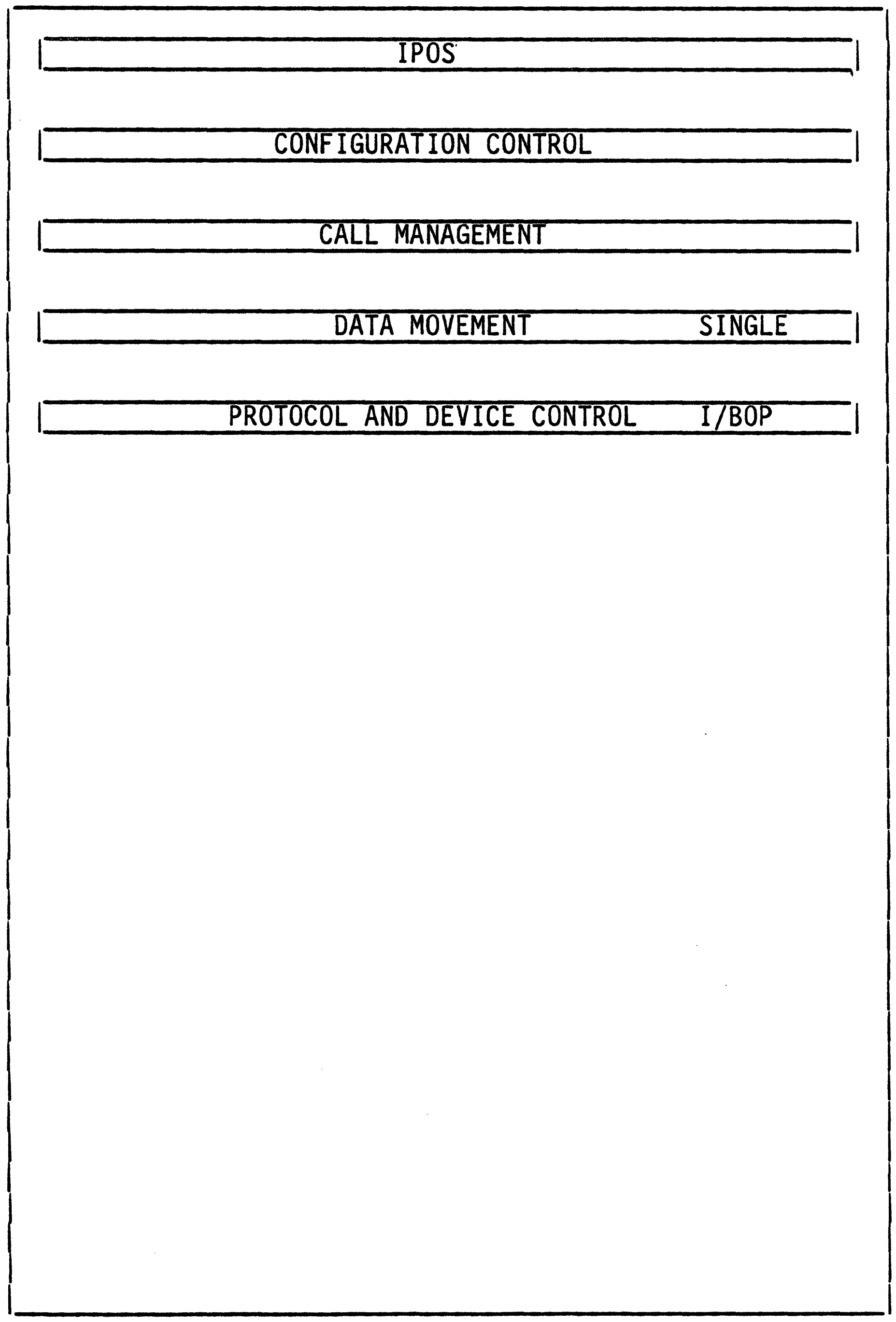
I/MSTP SUBSYSTEM



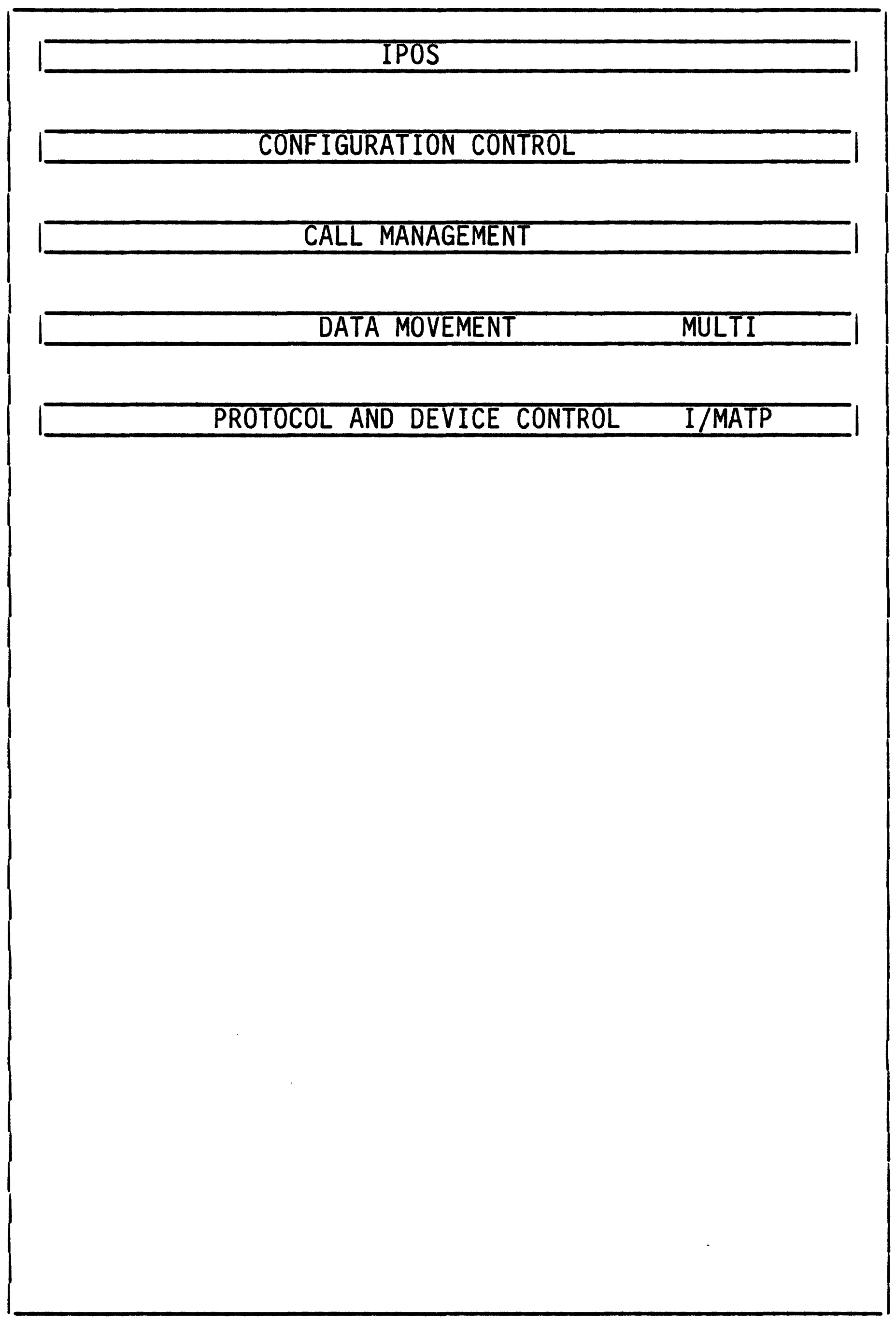
I/SSTP-BSC SUBSYSTEM

```
IPOS

CONFIGURATION CONTROL

CALL MANAGEMENT

DATA MOVEMENT                    SINGLE

PROTOCOL AND DEVICE CONTROL  I/SSTP-BSC
```

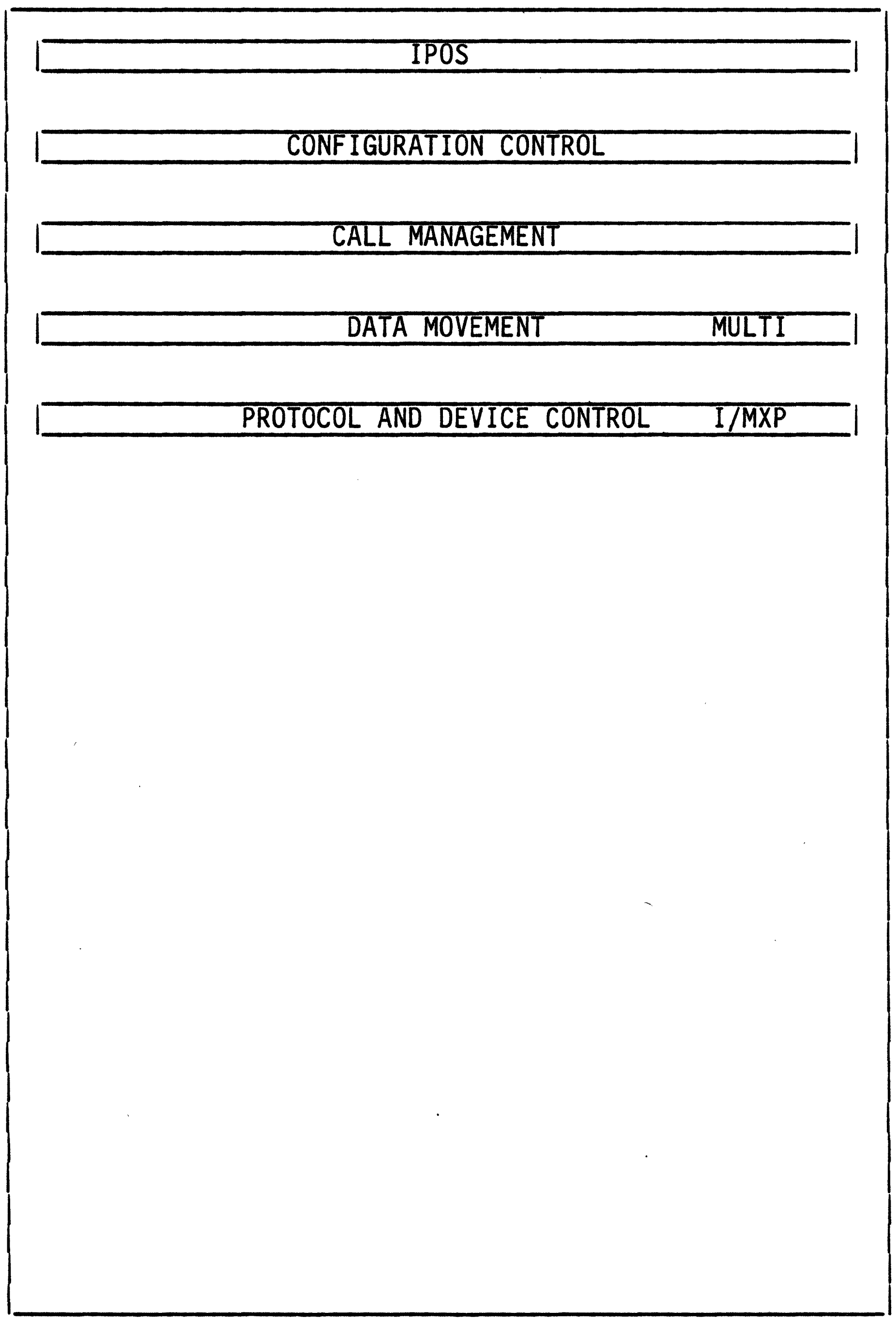
I/BOP SUBSYSTEM



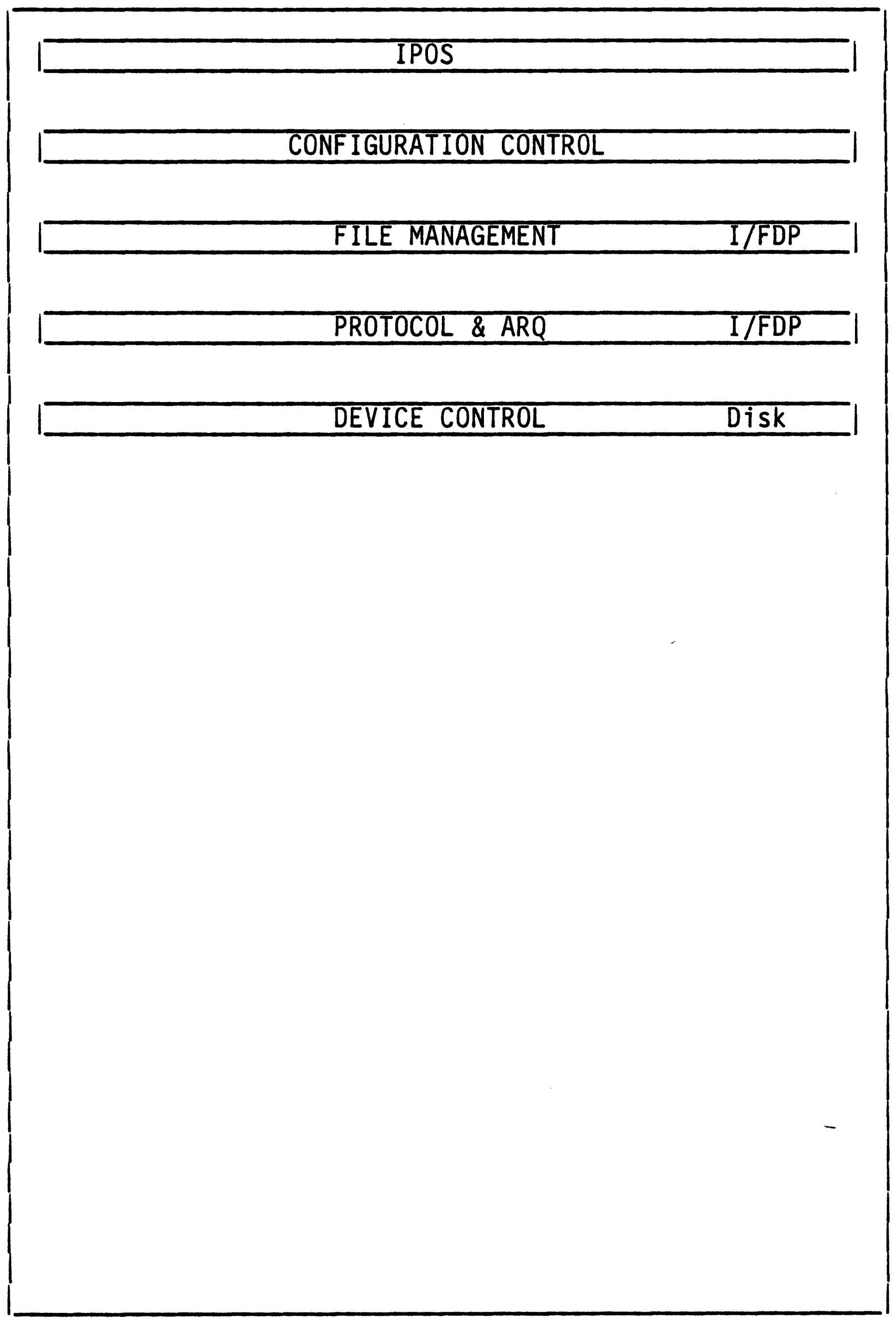
I/MATP SUBSYSTEM



I/MXP SUBSYSTEM



I/FDP SUBSYSTEM



3.3 Bus Interface Chip (BIC) Operation

The Bus Interface Chip (BIC) is used to interface the mainframe nest bus to the individual I/Ps in the nest(s). A detailed hardware description of the BIC may be found in the D814 Hardware System Specification. The design of the software operating the BICs from the mainframe (controller) side may be found in Sections 5.2, 5.8, and 5.11 of this document and in Sections 6.1.6, 6.4, and 6.5 from the port side. The specifics of the detailed design of the BIC operating code may be found in the detailed design specs for the modules described in the sections noted above.

3.3.1 BIC Operations from the Controller (Mainframe) Side

This subsection provides a broad overview of the mainframe side of the Bus Interface Chip (BIC). The data formats used for addressed packet and data transmission over the BIC have already been described.

3.3.1.1 BIC Packet FIFO

All I/Ps have a BIC number 0. This BIC is used for both downline loading of operating and diagnostic software and addressed packet communication between the port and the mainframe (see subsection on Mainframe Addressed Packet (MAP) Module).

The data format for downline load data is described in Section 3.2.1. Interrupts are not used when downline loading, and the load block described there is loaded directly into the outbound FIFO without using any coding for zero bytes.

The data format for addressed packets is also described in Section 3.2.2. The BIC #0 FIFOs (called the packet FIFOs) are used to transmit packets in segments small enough to fit entirely in the FIFO. The reader's and sender's flags are used to ensure that a segment is not read until it has been completely loaded. In other words, reading and writing of addressed packet segments (unlike user data in BIC #1) is strictly synchronous. This is all described in detail in the subsection on the Mainframe Addressed Packet (MAP) Module.

After downline load the mainframe packet FIFO control registers are set to generate service segments only when the port's sender's or reader's flag is set, and the addressed packet logic is interrupt driven (see subsection on MAP).

3.3.1.2 BIC Data FIFO

BIC #1 at an I/P (if it exists) is used for user data transmission using the High-Speed Data Interface format discussed in Section 3.2.3. Since this format does not allow the zero byte to be sent, the BIC may be determined to be empty whenever a zero is read (see BIC Design Specification). This allows software on both the mainframe and the I/P side to disperse with most BIC FIFO status checks and therefore run more efficiently.

BIC data FIFOs associated with single-threaded and multi-threaded terminal ports are accessed as needed by the Mainframe Network Link (MNL) Module and the Mainframe Multi-Threaded Port Control (MMT) Module; interrupts are not used in accessing the I/TP data FIFOs.

BIC data FIFOs associated with I/NPs, on the other hand, are interrupt driven. To minimize context-switching overhead at the expense of increased queueing delay, the BIC inbound and outbound FIFO control registers for an active port are set to cause mainframe service requests (see subsection on Mainframe Tack Control (MTC) Module) when the inbound (mainframe-bound) FIFO goes more than half-full and when the outbound FIFO goes less than half-full. In addition, an I/NP may force a mainframe data FIFO interrupt using the inbound FIFO sender's flag on the outbound FIFO reader's flag.

3.3.2 <u>BIC Operations from the Port Side</u>

This subsection provides an overview of the BIC presented from the port side. The presentation is in two parts: BIC #0 (packet BIC) which is manipulated by the IPOS operating system, and BIC #1 (data BIC) which is manipulated by the IP's to move network user data.

3.3.2.1 <u>IP Packet BIC (BIC #0)</u>

The IP's use BIC-0 to download port software and online diagnostics and to transmit and receive IPOS addressed packets between the port and mainframe. Address packet formats are described in Section 3.2.2.

Downline load data formats are described in Section 3.2.1. The port downline load algorithm does not require the BIC-0 interrupt flags. Data is moved directly from the BIC to port RAM without decoding attempts for zero value bytes.

Exercising facilities and diagnostic information are provided using BIC-0. See Section 4.1, IPL ROM.

When used for addressed packets by IPOS, BIC-0 runs under interrupt control using the senders and receivers flags.

3.3.2.2 IP Data BIC (BIC #1)

All IP's have a data BIC but some, such as the I/CTP, I/FDP and I/DGP, which do not move network data, do not use this BIC. When utilized by the IP, BIC 1 transfers data blocks (as described in Section 3.2.3) using the interrupt-on-half-full and senders/receivers flag interrupts.

The former allows both the port and mainframe to interleave processing while the latter insures that all data blocks are processed as soon as possible. This methodology requires encoding of hexadecimal zeroes to distinguish real zeroes from those read from an empty FIFO. An exception to the encoding scheme occurs in the I/NP which uses hexadecimal zero as a transparent line pad character.

4. FIRMWARE

The term "firmware" in this document refers to the IPL code in ROM in the mainframe or IP.

There are four distinct firmware subsystems in D814:

1. General Port IPL ROM
2. I/FDP IPL ROM
3. Mainframe IPL ROM
4. Mainframe ROM-Resident Diagnostics

Each of these are discussed in the following sections.

4.1 D814 Port IPL ROM

The IPL ROM used in the D814 ports supports four IPL functions. The reset bits in the packet BIC designate what type of reset is being performed. Any reset causes the port diagnostic LED to go on. The functions of the resets are:

RESET 0: (Master Reset)

This reset occurs during power up and under software request. When a reset-Ø is detected, the engine executes an instruction set test, a ROM checksum test and two RAM tests, leaving RAM cleared. When these are completed successfully, control goes to the port debugger if one is present. Otherwise, the port turns off the diagnostic LED, sets the port bit in the packet BIC, and then waits for another reset.

RESET 1: (Software Load)

Reset 1 is issued by the mainframe software (MSI, MDL, MIL) when software is to be loaded into a port. Memory is sized, and the communication card ID, high order address byte of the highest page of port RAM, and processor ID (00-6800, 01-6809) are passed to the mainframe using the packet BIC. The port then awaits software from the mainframe (including a start of load location, a start of execution address, and a 2 byte checksum).

Multiple individual loads may be performed. ~~The port continues to expect~~ software ~~until it receives a load~~ block with a start address other than zero. Execution then begins at the specified address. If the port debugger is attached, the start address is saved and control passed to the debugger ROM.

RESET 2: (Limited BIC Test)

Reset 2 is designed to test the BICs. Half of this test resides in the mainframe ROM. Reader and sender flags, FIFOs, and port interrupts are tested. Upon successful completion, the port turns off the diagnostic LED, indicating that the port is minimally capable of loading software through its BIC.

RESET 3:

Reset 3 is used to retrieve information from a port which has failed. The first 32 bytes of port memory are loaded into the packet BIC, and the sender's flag is set.

4.2 D814 I/FDP IPL PROM

In addition to the standard D814 IPL ROM, the I/FDP has a 256 byte PROM on the second card. When a mainframe wishes to boot an I/FDP, it performs a reset 1 and loads a standard jump instruction as software. The jump, when executed, will transfer control to the PROM which will bootstrap the port.

The bootstrap PROM will read one byte from the IPL BIC which it uses to determine on which drive system software should be located. If the drive is non-existent, a NAK is sent to the mainframe with a cause code indicating such.

Otherwise, the IPL PROM reads a standard record from the floppy disk and executes the code read. If, in the process of bootstrapping, an error occurs, a NAK and appropriate cause code are sent to the mainframe.

If the bootstrap completes without error, the ACK followed by the software release and level are stored in the BIC FIFO. The I/FDP is prepared to start execution; however, execution will not begin until the mainframe sends a startup indication over the IPL BIC.

The IPL PROM is customized to load the standard record from a particular location on the floppy. Each type of I/FDP therefore requires a customized PROM and a boot record on the floppy. Note that while I/FDP boot software is expected from a software disk, every D814 floppy has the IPL software in the same standard fixed locations.

4.3 Mainframe IPL Module

4.3.0 Introduction

The mainframe IPL module (MIL) is the ROM resident mainframe system component which takes control after any D814 node restart. A firmware restart causes a nest master reset and causes all mainframe processors to jump to mainframe ROM through a restart vector. There are four ways to initiate a firmware restart. These four 'restart types' are:

A. Power-up - Power is restored to the mainframe generating a power-up sequence.

B. NP Restart (hardware boot) - Interrupt to the master controller generated by hardware. Triggered by a "restart sequence" received by a network port.

C. Software Restart - Generated by a master controller instruction under software control (see Mainframe Module MSB).

D. Reconfig Restart - Type of software restart, used to reload port software when changing configurations.

When mainframe ROM is entered, Mainframe ROM diagnostics are executed, after which the MIL entry point MIL$INIT:ENTRY is jumped to.

MIL determines the local source port for mainframe software and supervises the loading of that software. The source depends on the restart type and on parameters passed to MIL. The source of software port may be a floppy disk port local to the IPL'ing node (node running MIL), a floppy disk emulator port local to the IPL'ing node, or a local network port. Software loaded through a network port is passed by MDL (see Mainframe Downline Load) of a running node over a network link.

The following terminology will be used for this discussion:

adjacent node	- (or neighboring node) - Node running mainframe software which is linked to the IPL'ing node through a network link.
IPL'ing node	- Node running MIL.
preferred software	- The software level passed to MIL in page Ø of RAM or in CMEM, if specified.
preferred link	- The network link which communicated the hard boot command for an NP restart.

chosen software	- The software level (revision # and release #) which is actually loaded.
chosen link	- Network link actually used for loading, when loading through an NP.
loading port	- Port used to load software (may be I/NP, I/GBNP, I/FDP, or I/FDPE). Same as source of software port.
RNP	- ROM Network Port. Network Port code running in NP (network port) local to IPL'ing node. RNP performs functions of an NP to enable IPL'ing node to get software from adjacent node. RNP code resides in mainframe ROM, and is downloaded to the NP by MDL or MDM.

Note that NP refers to both I/NP's and I/GBNP's unless otherwise stated.

4.3.1 Functional Overview

MIL is invoked in the following situations:

A. Power-up

When power is restored to the mainframe after a power interruption.

B. NP Restart (Hardware Boot)

A command by the ICTP operator directs a node to send a special code over a network link from an operator-specified network port. This causes an NP restart of the remote mainframe. A hardware boot is done to cause new software to be loaded into a specified node over a specified link. It attempts the restart of a node regardless of the node's state at the time of the boot.

C. Software Restart

A command by the I/CTP operator initiates the restart of a node and all nodes of the same connected network (nodes running inconsistent software are not part of the same connected network). This is used for two purposes:

1) Network Reboot - To change the software of the entire network.

2) To restart a connected network which is joined to the network with the I/CTP. For example, if the two joined networks are running different software versions, the I/CTP operator can issue a software boot command and restart all of the nodes in the adjoining network. They will come up with the proper software and become part of the ICTP operator's network. The I/CTP operator communicates the command to the NP in his network which has a network link to the adjoining network. The command is communicated over the link by means of a code in the initial link protocol (see section on MSB).

These two cases are indistinguishable to MIL. The Mainframe System Boot Module (MSB) communicates to every node in the connected network the new software release and revision level and configuration. Each node then does a software node restart. Software configuration parameters are passed in page Ø of RAM.

D. Reconfig Restart

A reconfig restart occurs when a node is rebooted to change configurations without changing software levels. This may happen automatically or as a result of an I/CTP operator command. This restart causes all ports to be reset and subsequently reloaded (see MDM section). Mainframe software is not reloaded unless the mainframe checksum is invalid. An invalid checksum is treated like a system error in which case mainframe ROM diagnostics are restarted.

E. System Error (Special case of a software restart)

A system error is a fatal mainframe error. If the mainframe debugger is present when a system error occurs, the mainframe software traps to the debugger from MDM without causing a restart (see sections on MDBG and MDM). If the debugger is not present, a SYSERR code is saved and a software restart is done.

When MIL discovers a SYSERR it delays the number of minutes specified in CMEM (at EQ$MCM:OF_DELAY) while broadcasting SYSERR messages on the front panel and in HELP messages which are sent over all RNP's. When the delay is complete, it clears the error code and resumes the normal load sequence. Note that while in delay, the operator can cause a jump to the debugger from the front panel keyboard (see subsection on front panel interface).

Error Handling:

Two types of errors may occur during MIL: fatal errors and non-fatal (or recoverable) errors. Error messages are communicated by displaying them on the front panel and sending error codes in HELP messages.

A. Fatal Errors

A fatal error is a serious mainframe error which occurs during MIL. When a fatal error occurs, the normal load sequence is interrupted, and a HELP message with an error code is sent across a network link. The IPL'ing node waits for commands from an adjacent node.

B. Non-fatal Errors

A non-fatal error is an error which does not interrupt the normal loading sequence. A HELP message with error code is sent. If the error is a port error, a new loading port is found (see error handling under loading software).

4.3.2 Operational Overview

After a restart, the node is in the following state:

- All ports have been reset and are running ROM-resident engine diagnostics (these are started up automatically).
- All NIC's are in loopback mode.
- All processors are reset, lowbank of memory is selected, all processors jump to mainframe ROM diagnostic routines (MDAG).

  (See section on Mainframe ROM Diagnostics.)

MDAG routines are executed and MIL is jumped to. At entry to MIL, the following has occurred:

1) Mainframe diagnostics have been successfully completed if run (diagnostics not run if mainframe does not require reloading).

2) NIC loopback tests are completed and NIC's are no longer in loopback.

3) Processor Ø jumps to MIL$INIT:ENTRY (entry point to MIL). All other processors are halted until restarted by MIL.

MIL does the following:

1) Compute CMEM checksum, compare to value stored in CMEM, store valid/ invalid parameter.

2) Read port bit (for engine diagnostics completion) from BIC-Ø of each engine and keep table of error codes for all ports.

3) Call BIC loopback routine for each working engine and add failures to port table.

4) Send Reset 1 to each working engine. Keep loading port table of ID information for I/FDP's, I/FDPE's, I/GBNP's and I/NP's.

5) Determine Restart type. (Details below)

6) Determine chosen software, chosen link and loading port. (Details below)

7) Start up loading port and load mainframe software. Monitors loading port for errors. If an error occurs goes to step 6 to find another loading port. (Details below)

Upon successful completion of MIL the IPL'ing node is in the following state:

- mainframe software is loaded
- all ports have completed engine ROM diagnostics and BIC loopback tests

Note that no ports are loaded with system software during MIL.

The parameters saved for the mainframe are defined in the section on interfaces.

At this point the mainframe software is started up by jumping to the load blocks start execution address.

Steps 5, 6 and 7 are explained in more detail:

Step 5: Determining Restart Type

There are four different restart types:

1. Power Up
2. NP restart (or hardware boot)
3. Software
4. Reconfig - (This is a software restart which changes the configuration without changing software levels.)

MIL uses the following variables to determine restart type:

1. Location EQ$MC:RSRT of Master Controller local memory.
2. The reconfig parameter OF$MIL:RECONFIG stored in RAM, page Ø (by MSB).
3. The attention sequence registers on the NP (passed to MIL by the RNP).

MIL reads location EQ$MC:RSRT of master controller local memory. The contents of this location allows MIL to distinguish between a boot generated by software (either software or reconfig restart) and a boot generated by hardware (either power-up or NP restart).

If the boot was generated by software, MIL reads OF$MIL:RECONFIG to determine if restart type was software (EQ$MIL:SFTWR_RSTA) or reconfig (EQ$MIL:CONFIG_RSTA).

For a reconfig restart, mainframe software is not reloaded. MIL checksums mainframe code. If the checksum is invalid, MIL stores the reconfig parameter for software restart and a system error code; and causes another node restart (in order to run mainframe diagnostics).

If the boot was generated by hardware, MIL polls each network port to determine if a network port received a remote boot sequence (see RNP section). If no network port received a remote boot sequence, then the restart type is power-up (EQ$MIL:POWER_RSTA). Otherwise, the restart type is hardware boot (EQ$MIL:HRDWR_RSTA).

<u>Step 6</u>: Determining Chosen Software, Chosen Link, Loading Port

The algorithm used to determine these parameters depends on the restart type (determined in step 5) and is described below.

In the process of selecting a loading port, loadable port diagnostics may be executed. A floppy disk port or floppy disk emulator port always executes loadable diagnostics.

When a loading port is not specified (not a hardboot) MIL selects a loading port by referencing the loading port table. This table is created by MIL and contains the following information for each possible source of software port (I/FDP, I/FDPE, I/GBNP, I/NP):

1) TPORTNUM	- the port address
2) TCOMCARD	- the communications card ID code
3) TERROR	- error information
4) TFDP_DISP	- address of 32 character display buffer (valid for floppies only)
5) TNXT_PORT	- address of next port in table (EQ_LAST_PORT if this is last port)

I/FDP's are the first entries in the table, this ensures that MIL tries to get software from its floppy disks before trying to load through the network. The very first entry in the table is the floppy disk port emulator, if there is an emulator local to the IPL'ing node. A node may have at most one I/FDPE. MIL determines that there is a floppy emulator local to the node by reading a CMEM node parameter (EQ$MCM:OF_FEAS) which contains the address of the floppy emulator. If the floppy emulator address is non-zero and the port plugged into that address is working and is a valid emulator port, then MIL assumes this is an I/FDPE. MIL tries to load through this port before trying to load through any other source of software port. A valid emulator port is a port whose hardware ID is either CTC or I/BYTE. MIL passes through the table trying to find a port to load software from. The error information indicates a fatal or non-fatal error condition. If a fatal error is associated with a port, that port is not used to load software.

Load Any Software:

There are two ways of specifying 'load any software available':

1) If the restart type is power up and CMEM is invalid, then preferred software is EQ$MIL:ANY_REV, EQ$MIL:ANY_REL. This instructs the node to accept any software available. This request is not propagated to other nodes.

2) If the operator inputs 'no preferred software' from the front panel the node loads any software available. In addition, the instruction to load any software is propogated through HELP messages to any adjacent nodes which are IPL'ing. If an IPL'ing mainframe receives a HELP message which specifies no preferred software it loads any software available and continues propogating the instruction through HELP messages.

If the restart type is hardware boot then MIL only tries to load software from the network port which received the remote boot sequence.

Otherwise, MIL tries each possible source of software port until it finds a port with an acceptable version of software available. MIL finds the next possible source of software port by looping through the loading port table and trying the next port entry which does not have a fatal error.

MIL determines chosen software and loading port depending on the restart type, as follows:

1. Restart type is power-up:

   a. Chosen Software

      If CMEM is valid, the chosen software is the last running version of software (read from CMEM).

      If CMEM is invalid, MIL loads any software available.

   b. Loading Port

      The loading port is the first port found with the chosen software available.

2. Restart type is software restart:

   a. Chosen Software

      Chosen software is the software revision and release passed to MIL in page Ø of RAM.

   b. Loading Port

      The loading port is the first port found with the chosen software available.

3. Restart type is hardware boot:

   a. Chosen Software

      Chosen software is the software available through the loading port.

   b. Loading Port

      The loading port is the network port which received the remote boot sequence. (In this case the loading port is known, so MIL goes to step 7.)

MIL determines which port is the next possible source of software port. Then MIL determines if this port has acceptable software as follows:

A. If the load port is an I/FDPE or an I/FDP then the following is done:

   1) Floppy diagnostics are run on the port (if this is an I/FDP the diagnostics are ROM resident. If this is an I/FDPE, diagnostics and a bootstrap program are loaded from the host computer and run).

      If the diagnostics fail, the error code is stored in the loading port table and a new loading port is found.

   2) MIL reads the floppy directory information to find what drives are mounted and what level of software is available.

      - If no software disks are present, MIL finds a new loading port.

      - If MIL is looking for any software then the highest software level available is the chosen software. This floppy is the loading port.

      - If the floppy has the chosen software available then the floppy disk port is the loading port.

      - Else the floppy does not have the chosen software. MIL finds a new loading port.

B. If the loading port is an I/NP or an I/GBNP then MIL does the following:

   1) Reads the BIC to determine if the network port has exchanged link inits (LI) with the remote NP (MIL had previously loaded all local network ports with loading port software, and sent STREAM_LI commands to them.)

      If NP did not receive an LI from the remote node, MIL finds a new loading port.

2) MIL sends a command to the local network port instructing it to send a HELP message to the remote network port, and waits for network port to get an IPL frame from the remote node. The local network port passes software level to MIL.

- If the local RNP is talking to another RNP (received a HELP message) then MIL finds a new loading port.
- If the software level is not the preferred software, then MIL finds a new loading port.
- Else the software level available is acceptable. This is the loading port.

Step 7 Start Up Loading Port and Load Software

This step depends on the type of loading port (floppy disk or network port). If for any reason it becomes impossible to load because of a loading port failure, the port is marked as non-working in the loading port table, and MIL goes back to the beginning of Step 6 (determining chosen software and loading port).

A. Loading Software from a Floppy Disk Port or a Floppy Disk Port Emulator

If the loading port is an I/FDP or an I/FDPE, the local node determines the load sequence. (See subsection on Interfaces.)

MIL receives the software level of each drive. MIL specifies the drive # followed by one of the following commands:

'Load Mainframe'	(EQ$MIL:LOAD_MF)
'Load Floppy'	(EQ$MIL:LOAD_FDP)
'Start Floppy'	(EQ$MIL:START_PORT)

The normal load sequence when loading through an I/FDP or an I/FDPE is:

1. Load mainframe software
2. Start mainframe

The loading port is loaded under MDL.

B. Loading Mainframe through a Network Port:

If the loading port is a network port (either I/NP or I/GBNP) then the loading sequence is determined by the adjacent node in response to a software request in the form of a HELP message.

MIL sends HELP, requesting mainframe software without specifying a file (unless Diagnostics Monitor is requested). In response the adjacent node can send the following commands followed by appropriate software to the IPL'ing node:

(See RNP section on HELP messages.)

1. Load NP
2. Load and Start NP
3. Load and Start MF

When loading the mainframe through a network port the normal sequence involves loading and starting mainframe software only. The loading port is loaded under MDL.

See RNP section for the action taken for each of the three commands.

Error Handling

A. If an error associated with a loading port occurs while loading and the restart type is not EQ$MIL:HRDWR_RSTA (hardware boot), the following is done:

   1. If the error is non-fatal,

      a) display error message on front panel
      b) go to Step 6 to find new loading port and chosen software.

      Note that this port may be tried again as a loading port.

   2. If the error is fatal,

      a) store error code in loading port table
      b) display error message on front panel
      c) go to Step 6 to find new loading port and chosen software.

B. In the case of a hardware restart, there is only one possible source of software port. If any error occurs while loading, fatal or non-fatal, the IPL'ing node sends a HELP message with error code across the link and waits for a new command from the running node.

4.3.3 External Interfaces

A. Parameters:

Depending on the type of restart, MIL may access the following information:

1. Mainframe code: On a RECONFIG or SOFTWARE restart, mainframe code is left intact in RAM. Location OF$MIL:MF_CHCKSUM contains a checksum (two-bytes with wrap-around carry) for the mainframe code.

2. Software restart parameters. The following parameters are left for MIL in page Ø of RAM on a SOFTWARE or RECONFIG restart:

OF$MIL:CONF	configuration #
OF$MIL:SWLV_REV	software revision #
OF$MIL:SWLV_REL	software release #
OF$MIL:RSTA_INPT	initiating port
OF$MIL:RSTA_INND	initiating node
OF$MIL:RECONFIG	reconfiguration code
OF$MIL:SYSERR	system error code
OF$MIL:START_ADDR	mainframe code start address
OF$MIL:CODE_END	mainframe code end address
OF$MIL:MF_CHKSUM	mainframe checksum

3. Configuration Memory. MIL always checks the validity of the checksum in CMEM. If CMEM is valid the following parameters may be accessed:

   a) Software level (EQ$MCM:OF_REV, EQ$MCM:OF_REL) - Used on power-up to determine last running software.

   b) Delay (EQ$MCM:OF_DELAY) - Used to determine number of minutes to delay if a SYSERR has occurred.

   c) Floppy Emulator Port Address and IPL Speed (EQ$MCM:OF_FEAD and EQ$MCM:OF_FEIS). - These are used to determine the port address of a local floppy disk emulator port, and the line speed for the emulator during IPL.

4. Master Controller. A restart mode is stored in firmware accessible Master Controller Memory. This parameter is read using the Master Controller RLM command (see 6000 Logic Design Spec.). This parameter indicates (a) NP or powerup restart, or (b) software or reconfig restart.

5. Remote Boot register on I/NP or VBIT card. Register readable (and clearable) by port software. The contents of these registers are used to determine the link initiating a hardware boot. If more than one hardware boot was received, the NP with the lowest port number is the one used for loading.

6. System Initialization Parameters. The following parameters are left in page Ø of RAM for mainframe system software, on termination of MIL:

OF$MIL:CONF	Configuration Loaded
OF$MIL:SWLV_REV	Software Revision Loaded
OF$MIL:SWLV_REL	Software Release Loaded
OF$MIL:RSTA_INPT	Port Initiating Restart
OF$MIL:RSTA_INND	Node Initiating Restart
OF$MIL:RSTA_TYPE	Type of Restart
OF$MIL:LOAD_PT	Port # used to load mainframe
OF$MIL:CMEM_CHK	CMEM checksum status
OF$MIL:ERROR	ERROR code for load
OF$MIL:RECONFIG	Reconfiguration Parameter
OF$MIL:START_ADDRESS	Code start address for reconfig restart
OF$MIL:CODE_END	Code end address for reconfig
OF$MIL:SYSERR	System error code
OF$MIL:MF_CHKSUM	Checksum of load block

7. Port Status Table. One byte of status information for each of the 126 possible ports in a node is saved in tabular form for the mainframe system software. This port table starts at OF$MIL:PORT_TBL. The status of port PORTNUM is located at OF$MIL:PORT_TBL + (PORTNUM/2). The status byte has the following possible values:

EQ$MIL:GOOD_PORT	No errors in initial diagnostics
EQ$MIL:NO_PORT	No port present at this address
EQ$MIL:ENG_FAIL	Port bit was not set
EQ$MIL:BICØ_FAIL	BIC-Ø failed loopback test
EQ$MIL:BIC1_FAIL	BIC-1 failed loopback test
EQ$MIL:BICS_FAIL	Both BICs failed loopback test

8. Floppy Port Status Table. Six bytes of status information for each floppy disk port in a node is saved for up to EQ$MIL:FLOPPY_MAX floppy disk ports. This floppy table starts at OF$MIL:FLOP_TBL. The status of each floppy consists of:

OF$MIL:FLOP_PORTNUM	
OF$MIL:FLOP_STAT	Status information from diagnostic termination packet
OF$MIL:FLOP_DRIVE Ø	
OF$MIL:FLOP_DRIVE 1	
OF$MIL:FLOP_DRIVE 2	
OF$MIL:FLOP_DRIVE 3	
OF$MIL:FLOP_READ	Error while trying to read from floppy

B. Load Block Format:

Load Data is sent to MIL in load block format, regardless of the loading port type (I/NP, I/GBNP, I/FDP or I/FDPE).

Load data is received by MIL encoded.

The load block format (before encoding) is:
(See Section 3.2.1, Program Load Interface.)

Load Header:

Bytes 0 - 1:	Start address (0 if not last load block)
Bytes 2 - 3:	Load address
Bytes 4 - 5:	Byte count of load block including header.

Data:

Bytes 6 - n:	Object code in binary.

Checksum:

Bytes n+1 - n+2:	16 bit end-around carry checksum of load block including header.

The data field of the load block is encoded by the sender according to the scheme:

X'00' --> X'FFFF', X'FF' --> X'FFFE'

This encodes zero bytes in the data. The reason for this is to allow zeroes to be used as escape characters across a network link and to facilitate sending data in the BIC.

MIL must decode any load data it receives. (Note that the RNP must also decode NP load data - see RNP section.)

C. Diagnostics Interface

1) Mainframe ROM resident diagnostics (MDAG). MIL is entered upon successful completion of mainframe ROM diagnostics. If the mainframe fails diagnostics, MIL is never entered. No parameters are passed. Before jumping to MIL, diagnostics halts all processors other than processor Ø. (See section on Mainframe ROM Resident Diagnostics.)

2) ROM resident engine port diagnostics. When a restart occurs, engine diagnostics are run. Upon successful completion, the Port Bit in BIC Ø is set (other bits may also be set). MIL reads the port bit for all engines. If the port bit is not set, MIL does not use this port for loading.

3) BIC loopback. MIL calls a diagnostic routine which does a BIC loopback test on the port passed to it and returns an error code. (See section on Mainframe ROM Resident Diagnostics.)

4) FDP diagnostics. The floppy disk port used to load software will be instructed to run ROM resident diagnostics under MIL (see I/FDP section on IPL ROM). Floppy status information (results of diagnostic) are displayed on the front panel. If the floppy diagnostics report a fatal error the I/FDP will not be used as a source of software port. If the floppy diagnostic reports a non-fatal error condition MIL may retry this port as a source of software. Floppy status information is saved for the mainframe system software (Floppy Port Status Table).

   The floppy disk port emulator runs diagnostics under MIL. The MIL interface with the I/FDPE to run diagnostics is the same as the interface with the I/FDP. Floppy emulator diagnostics are not ROM resident but are loaded from the Prime.

5) Diagnostic Monitor. The Diagnostic Monitor can be requested at the local node during restart (see front panel interface). The Diagnostic Monitor will be loaded under MIL control.

D. Floppy Disk Port

MIL may load mainframe software directly from a local I/FDP. MIL interfaces with the floppy disk port IPL ROM (see section on I/FDP). MIL sends commands and receives data from the floppy as follows:

1) Run ROM resident diagnostics

   a) MIL activates floppy ROM diagnostics by sending a reset 1 to the I/FDP with a software block (see IPL ROM section) with a starting address equal to OF$IP$ROM:FDP_DIAG.

   b) MIL reads from BIC-Ø Diagnostic Termination packet (and displays information on front panel).

2) Run floppy loader

   a) MIL activates this by sending a reset 1 to the I/FDP with a software block with a starting address equal to OF$IP$ROM:FDP_LOAD.

   b) Get directory information. MIL reads a drive # and software level bytes for each ready drive from BIC Ø. When there are no more ready drives MIL reads a completion code (EQ$MIL:FD_NODRIVE) from BIC Ø.

      (Note that at this point MIL may discontinue the load process from the I/FDP).

c) Load software. MIL puts the drive # (Ø-3) to load software from in BIC Ø. MIL follows this with a one byte command code which specifies

   i load mainframe (EQ$MIL:LOAD_MF),
   ii load I/FDP (EQ$MIL:START_FDP), or
   iii load mainframe with diagnostics monitor (EQ$MIL:DIAG_MNTR)
   iv start floppy (EQ$IP$ROM:IPL_START).

d) i If the command code was EQ$MIL:LOAD_MF, MIL reads mainframe software in load block format from BIC Ø. Zeroes are encoded. (See load block interface.)

   ii If the command code was EQ$MIL:LOAD_FDP, the mainframe reads a load ACK or NAK. (EQ$MIL:LOAD_ACK or EQ$MIL:LOAD_NAK) from BIC Ø.

E. Floppy Disk Port Emulator

MIL may load mainframe software directly from a local I/FDPE. The interface to the floppy disk emulator is similar to the interface with a real floppy disk port. The differences are:

1) Finding the floppy disk emulator port. MIL does this by reading CMEM for the floppy emulator address (EQ$MCM:OF_FEAD). If the port at this address passes its ROM diagnostics and its card ID is either I/CTC or I/BYTE then MIL treats this port as an I/FDPE. There may be no more than one I/FDPE at a node.

2) The software blocks sent to the emulator must have a load address of OF$IP$ROM:EMBOOT_SPEED and one byte of data whose value is that of the CMEM node parameter EQ$MCM:OF_FEIS, in addition to the start address specified above. (MIL includes the data byte and load address in the load block for a real floppy port, even though they are not required.)

3) The floppy emulator requires some operator intervention to communicate with the host computer.

F. Network Ports

MIL may load mainframe software from an adjacent running node over a network link. This is done by communicating through a local network port which may be of either type, I/NP or I/GBNP. The local NP runs ROM network port (RNP) software: software which enables an NP to communicate with a running NP and load software to the IPL'ing mainframe.

The two types of RNP software (RNP or RGBNP) are resident in mainframe ROM and will be loaded into local NP's by the MIL routine described below (MDL may also call this routine. See section on Mainframe Downline Load Module):

MIL$SUBS:LOAD_RNP

Routine in MIL to load RNP software into any network port.

<u>On Entry</u>: Reg A = Port number of NP
Reg B = Comm Card ID
Port has been reset (reset 1) and is ready to load software.

<u>On Exit</u>:

If successful Reg A = Load ACK
RNP is loaded and started

If unsuccessful Reg A = Load NAK

This routine does the following:

1. Loads RNP software
2. Reads ACK/NAK from BIC
3. If load successful, starts RNP

The entry point to this routine is a vector located at OF$MROM:MIL$LOAD_RNP.

MIL interfaces with the running RNP through the BIC's to load software. The MIL-RNP interface is explained in the section on RNP's. The interface between the RNP and the adjacent NP is also found in that section.

Note that the data the mainframe receives from the BIC has been encoded and needs to be decoded as follows:

```
X'FFFF' --> X'ØØ'
X'FFFE' --> X'FF'
(See load block format.)
```

G. Front Panel Interface

1) Keyboard Entries

a) Entering Commands

MIL recognizes two commands from the front panel of the local mainframe.

i	Request Diagnostics Monitor	(DIAG) (ENTER)
ii	No Preferred Software	(LOAD) (ENTER)

Note that these commands may only be entered at the beginning of MIL (at the time when the display reads 'MF DIAG COMPLETE') if a power-up restart was done. The display will echo the commands.

b) Changing the Display

Hitting the (ENTER) key on the front panel will cause the display to change under the following circumstances.

i If a fatal mainframe error has occurred the display will show all asterisks. Hitting the (ENTER) key will allow the operator to see the error message.

ii When looking for software, the operator may circle through status messages for each floppy disk port (each FDP has a 32 character status message) by repeatedly hitting the (ENTER) key. Each time the (ENTER) key is hit the display changes to the next FDP status message or primary front panel display. Whenever the primary front panel display changes, the primary message is displayed.

c) Jumping to the Debugger

An NMI can be caused from the front panel (at any time) by turning the key to 'pgm', hitting (ENTER), turning the key to 'DIAG' and hitting (ENTER). This causes a jump to the mainframe debugger, if present. (Note that this is done by the master controller.)

2) Display

a) Primary Messages

Load status will be displayed on the front panel during MIL. The load status message is the primary message on the front panel. It is separated into two fields, (i) current status and, (ii) last error (or load block information while loading).

i) Current status will include message such as:

```
'MF DIAG COMPLETE'
'DIAGNOSTIC MONTR'
   (if diagnostics monitor was requested)
'NO PREFERED SFTW'
   (if no preferred software was entered)
'CMEM CHECKSUM'
'PORT TESTS##'
'RESTART TYPE RR'
   (where RR is the restart type)
'LOOKING SWVV.LL'
   (where VV is the revision, LL is the release)
'LOAD FDP FD##'
   (where ## is the port #)
'LOAD MF FD##'
'LOAD MF NP##'
'LOAD MF GB##'
'LOAD RNP NP##'
'FLOPPY DIAG FD##'
'MF LOAD COMPLETE'
'BAD KEY!!'
```

ii) Last error field

Until software is actually being loaded, the second field will contain a message indicating the last error condition encountered by MIL including the following:

```
'INVALID CMEM'
NP## LINKDOWN
GB## NO SOFTWR
NP## LOAD BAD
FD## NO RDY DR
FD## NO SOFTWR
FD## ERROR EE
  (where EE is an error code)
```

When software is actually being loaded, the loading address of the current frame, and the frame number are displayed instead of a last error field.

Note that the primary message is displayed unless the operator enters a key. If a secondary message is being displayed and the primary message changes, the new primary message is displayed.

b) Secondary Messages

Secondary messages are messages which give more information on status, and are displayed only when an operator enters a key from the keyboard. There are two types of secondary messages:

i) Error Message

When a fatal mainframe error occurs, the display will show all asterisks. When the (ENTER) key is depressed, the secondary message, a 32 character error message, will be displayed.

ii) Floppy Disk Port Status

There is a 32 character status message for each floppy disk port which has run diagnostics. The floppy disk port status messages are secondary messages. They are displayed one at a time. The operator sees the next message by depressing the (ENTER) key.

4.4 Mainframe ROM Resident Diagnostics

4.4.1 Introduction

The mainframe diagnostics module (MDAG) is the first ROM resident mainframe systems component to be executed after any D814 node restart. (See section on Mainframe IPL).

First, it examines data in Master Controller local memory and determines whether the restart is a reconfiguration restart. If so, it waits 10 seconds, allowing engine ROM diagnostics to run to completion and then jumps to MIL. If the restart is not a reconfiguration, it executes mainframe diagnostics and then jumps to MIL. Upon detecting any mainframe error, MDAG halts and displays an error message.

4.4.2 Diagnostic Routines

The tests in MDAG are organized into four functional groups.

A. Processor Tests

The processor test group comprises seven tests, which are designed to test the hardware on a processor card. All processors execute these tests simultaneously. The group includes a simple instruction set test, ROM checksum test, local storage test, base register test, status initialization test, uniqueness test and run-halt test.

B. Memory Tests

The memory tests test all RAM in the mainframe. Page Ø RAM is tested non-destructively. Both low memory and high bank RAM are tested. The locking function of the lockbytes is tested and lockbytes are left cleared. These tests are executed only by processor Ø.

C. Master Controller Tests

This group tests the Real Time Clock, various task queueing instructions, and level queue interrupts. The Real Time Clock Tests are executed only by processor Ø, and the other tests are executed sequentially by each processor.

D. NIC Loopback Test

When a node restart occurs, all NICs go into loopback mode. This test verifies the data paths to a NIC, takes it out of loopback and continues to the next INIC, repeating until all INICs are tested.

4.4.3 Front Panel Display

The self-scan of the 6050 displays messages while the diagnostics execute, indicating their progress. The following displays may appear:

A. Initially, the screen displays a line of "At" signs (@) for 2 seconds.

B. If it is a reconfiguration restart, the message "RECONFIGURATION RESTART" appears for 8 seconds.

C. During processor tests, the display reads:

```
MF DIAGNOSTICS:  # # # # # #
```

Each # represents a counter corresponding to a processor. Each processor increments its counter upon completion of each processor test. If diagnostics discover a failure, an asterisk is displayed to the left of the counter corresponding to the processor which discovered the failure. All processors are then halted. The numbers corresponding to each test are as follows:

0. Simple Instruction Set Test
1. ROM Checksum Test
2. Local Storage Test
3. Base Register Test
4. Processor Status Initialization Test
5. Processor Uniqueness Test
6. Processor Run-Halt Test

D. During memory tests, the display reads:

```
MF DIAGNOSTICS:  #  [RAM FAIL XXXX]
```

The # is the counter maintained by processor Ø, which continues to increment during the memory tests. A failure causes the message in brackets to appear, along with the address of the failure. Then all processors are halted. Random data briefly appears on the screen during the low RAM test. The numbers corresponding to each memory test are as follows:

7. Page 0 Memory Test
8. Low RAM Test
9. High Bank RAM Test
A. Lockbyte Test

E. During master controller tests, the display reads:

```
MF DIAGNOSTICS:  #  X  X  [MC FAILURE]
```

The # is the counter maintained by processor Ø, which continues to increment during the master controller tests. The X's represent two additional counters for the last two tests, which are executed by each processor. A failure causes the message in brackets to appear, and then all processors are halted. The numbers corresponding to each memory test are as follows:

```
B.   Real Time Clock Timing Test
C.   Real Time Clock Interrupt Test
D.   X  X  Fork, Dispatch and Suspend Test
D.   X  X-1  Interrupt Level and Enqueue Test
```

F. During the NIC loopback test, the display reads:

```
MF DIAGNOSTICS:  #  [NIC  n  FAILURE]
```

The # is again the counter maintained by processor Ø. A failure causes the message in brackets to appear and then all processors are halted. The n represents the number of the failed NIC. The number corresponding to this test is:

E. NIC Loopback Test

4.4.4 Interface to MIL

At successful completion of MDAG, only processor Ø is running. All other processors are halted such that when started, they will jump to MIL at Entry Point MIL$LOAD:START_PROC. Processor Ø jumps to MIL at Entry Point MIL$INIT:ENTRY. If any mainframe failure has been found, MDAG never jumps to MIL, but remains halted until another node restart occurs.

5. MAINFRAME MODULES

5.1 D814 Mainframe Operating System

The D814 Mainframe Operating system is divided into three submodule groups, each of which provides a related set of functions.

1. Mainframe Task Control
2. Mainframe Buffer Manager (MBM)
3. Mainframe Utilities (MUT)

Descriptions of these groups are found in Sections 5.1.1 through 5.1.3.

5.1.1 Mainframe Task Control

Task Control provides much of the most basic interface with the 6000 Master Controller (see 6000 Logic Design Specification). It provides the user with the ability to start and stop tasks, handle interrupts, etc., without worrying about the complex hardware control commands involved.

Before continuing we need to define some terms. A task is running when its code is executing on a processor. A task is suspended when it temporarily relinquishes control of the processor. The data space, registers, and stack of a suspended task are all saved to allow it to pick up where it left off when it is again allowed to run. A suspended task is said to resume when it is provided with a processor (not necessarily the one it had been using before it was suspended) and allowed to run. It should be remembered that, as long as a task is running with interrupts unmasked, it may be interrupted at any time and may resume on another processor. This means that a task may only use local storage when interrupts are masked and that, in general, it is impossible to make any assumptions about what processor any particular task is running on. (See 6000 Logic Design Specification for definition of local storage.)

5.1.1.1 Data Structures for MTC

This section describes the data structures used by MTC to interface with user code.

5.1.1.1.1 Data Spaces

MTC is responsible for providing a data space to each task when it is started up. Data Spaces are 32-byte blocks of memory which reside in a fixed region of RAM. While a task is running, the 6000 maps address 0 - '1F' onto the task's data space. Data spaces are used in the D814 operating system to hold the information needed by MTC about each task in the system. There are currently 64 data spaces. One data space is dedicated to each of the processors to be attached to tasks handling hardware service requests (called "hardware tasks"). These "hardware data spaces" are the data spaces at the top of the data space area. The rest of the data spaces are initially placed on a Free Data Space Queue and removed and allocated by MTC as needed for tasks not started by hardware service requests.

Data spaces have the following fields which are used by MTC:

OF$DS:LNK - One-byte field used to link the data space onto the Free Data Space Queue using the one-byte data space number (see 6000 Logic Design Spec.). This data space field may be used as scratch storage by user code only while interrupts are disabled.

OF$DS:SPH - Two-byte field used to store the saved stack pointer while the task associated with the data space is suspended. This field may be used as scratch by user code while interrupts are disabled.

OF$DS:BATCH_SP - Saved stack pointer used in batch task data spaces (see subsection on MTC$BATCH). This field must not be modified by batch tasks.

OF$DS:BATCH_TBL - Batch task table entry address used in batch task data spaces. Also used for scheduled task table entry address (see subsection on MTC$SCHD). This field must not be modified by batch or scheduled tasks.

With the exception of the restrictions mentioned above, the user task is free to use the data space as needed for storage.

5.1.1.1.2 User Stack

MTC also initializes a stack for the user task. For batch tasks (described later) the stack register is set to point to a permanent stack allocated to that task. For all other tasks the stack initially points to the top of data space.

5.1.1.1.3 Idle Cycle Counter

MTC maintains a three-byte idle cycle counter, OF$PGO:TCIDLC, for use by the Mainframe Statistics and Monitoring Module. This field is incremented by MTC once for every 10 M6800 idle micro-cycles executed. The idle cycle counter is interlocked by means of lockbyte OF$SYSLCK:TCILK to ensure that all three bytes are consistent.

5.1.1.1.4 Local Storage

MTC uses the local storage field OF$LS:MTC as a scratch idle cycle counter. This field may not be modified by user programs after system initialization. All other local storage may be used as needed for scratch as long as interrupts are disabled.

5.1.1.2 MTC Submodule Descriptions

This subsection describes the services provided to the user by each of the MTC submodules.

5.1.1.2.1 MTC$MAIN, Basic Hardware Interface Code

MTC$MAIN provides the lowest-level task control interface between D814 software and the 6000 Master Controller. Routines in MTC$MAIN are used by other MTC components as well as by higher-level MTC submodules such as MTC$BATCH and MTC$SCHD (described later).

MTC$MAIN performs the following functions:

1. Task startup for forked tasks (tasks not started by hardware service requests)
2. Task termination
3. Interrupt handling
4. Handling of forked task service requests and hardware service requests (see 6000 Logic Design Spec.)

<u>Task Startup</u>

The task startup function is provided by the entry points MTC$MAIN:FORK, MTC$MAIN:FRK0, and MTC$MAIN:FRK1. Each is called slightly differently but the exit conditions are the same.

<u>Entry Conditions</u>

```
MTC$MAIN:FORK - A-reg = Contents of A-reg for new task
                B-reg = Bit-7:  segment, Bit 0-2:  priority level of
                        task to be started
                X-reg = Start address for new task
                OF$DS:BFADR = Initial contents of both X-reg and
                              OF$DS:BFADR for new task
```

```
MTC$MAIN:FRK0 - B-reg = Initial contents of B-reg for new task
                OF$LS:AR = Bit-7:  segment, Bit 0-2:  priority level of
                           task to be started
                Interrupts disabled or masked
                The rest are the same as MTC$MAIN:FORK

MTC$MAIN:FRK1 - A-reg = Data space number for the new task
                B-reg = Bit-7:  segment, Bit 0-2:  priority level of
                        task to be started
                The rest are the same as MTC$MAIN:FORK
```

Exit Conditions

If fork successful, CC:Z=0, and A, B, and X-reg destroyed. If fork unsuccessful due to no Data Space, CC:Z=1, and entry parameters are all preserved. OF$LS:AR, OF$LS:DR, OF$LS:8, OF$LS:9, and two bytes on the stack are used. When the new task is started up, processor interrrupts will be enabled and unmasked.

Task Termination

The basic task termination function is provided by entry points MTC$MAIN:TERMH, MTC$MAIN:TERMQ, and MTC$MAIN:GETASK. The first two of these entries are invoked by the macros MAC$TSKMAC:TERMH and MAC$TSKMAC:TERMQ, respectively. This allows the MTC debug flag to control whether they are entered from a JMP or from a JSR instruction. MTC$MAIN:GETASK is invoked by a JMP.

Entry Conditions

TERMH entry - Called to terminate a hardware task, meaning a task initiated by an interrupt or system restart and to which a hardware data space is assigned. The stack pointer must point to the highest address in data space.

TERMQ entry - Called to terminate a forked (also referred to as "queued") task. The stack must be empty.

GETASK entry - Called to terminate a task whose data space is not to be returned to the Free Data Space Queue. This entry is used when a task wishes to suspend itself until some other task starts it running again using MTC$MAIN:FRK1. There are no restrictions on the stack for GETASK.

Exit Condition

The calling task no longer exists, and there is no return to the caller.

Interrupt Handling

The interrupt handling function is provided by entries MTC$MAIN:SWI and MTC$MAIN:IRQ.

MTC$MAIN:SWI is the D814 system software entry point for software interrupts. MTC checks if the two bytes following the SWI introduction contain the address of MTC$DELAY:ENTRY and, if so, vectors to that routine. If not, control is passed to the debugger software interrupt handler at X'FF80'.

MTC$MAIN:IRQ is the D814 system software entry point for IRQ interrupts. The 6000 Master Controller may interrupt one of the processors in order to allow a forked task or hardware service task to pre-empt a lower-priority task. (This is described in the 6000 Logic Design Spec.). The MTC$MAIN IRQ interrupt handler suspends the task currently running and then handles the highest priority service request pending.

Service Request Handling

MTC$MAIN service request handling is initiated by MTC whenever an interrupt occurs, whenever a task is terminated normally, and whenever MTC$IDLE detects a pending request. (It should be noted that even hardware service requests are frequently handled without generating IRQ interrupts in order to minimize overhead.) MTC$MAIN suspends the task currently running, decides what the source of the highest priority service request is, and vectors to the proper module for service.

5.1.1.2.2 MTC$BATCH, Batch Task Module

MTC$BATCH provides a user interface for the startup and termination of batch tasks. A batch task in a forked task which processes items taken one by one from a queue, called the "batch queue", associated with that task. The important difference between a batch task and one that is directly forked by the user in that there can never be more than one of any given batch task in existence (either running or suspended), while the same task may be forked, and will be started up, regardless of whether or not it is already running. So there can never be more than one data space assigned to one batch task, while any number of data spaces may be associated with identical forked tasks running concurrently.

Another difference between batch tasks and other forked tasks is that batch tasks are set up with a stack in a fixed area of RAM defined by the Batch Task Table (see below) rather than in data space.

Batch tasks are defined in the Batch Task Table template which is used by MSI$ to build the Batch Task Table at system initialization. Entries in the template have the following fields:

OF$BATCH:TEMADR - Entry point for batch task
OF$BATCH:TEMLVL - Priority level and memory segment for batch task (segment in high order bit)
OF$BATCH:TEMSTACKSIZE - Batch task stack size

The nth entry in the template is the entry for the batch task with module number n. (The batch module number is a unique number assigned in file EQ$BATCH.)

A batch task is started by calling routine MTC$BATCH:ENQ as follows:

Entry Conditions

* X-reg - Points to item to be enqueued to the batch task queue. The address contained in X must be the address of a four-byte queue element for use by the queue routines.

* A-reg - Contains module number

Exit Conditions

All registers are destroyed
CC:Z = 0

The desired entry is enqueued on the batch queue. The batch task will be forked by MTC$BATCH if necessary to ensure prompt processing of the enqueued item.

MTC$BATCH provides the following routines for use by batch tasks in accessing the batch queue:

MTC$BATCH:DEQTERM

Entry Conditions

* Batch task data space fields (described earlier) and stack must be the same as when started.

Exit Conditions

* If there is anything on this task's batch queue, the routine dequeues the item and returns to the caller with X register pointing to it. A and B registers are wiped out. Otherwise, the batch task terminates.

It should be noted MTC$BATCH:DEQTERM is synchronized with MTC$BATCH:ENQ and any attempt to dequeue items from the batch queue using other means will make it possible for a queue entry to be missed.

5.1.1.2.3 MTC$SCHD, Scheduled Task Submodule

A scheduled task is a forked task which is automatically forked periodically by the operating system. MTC$SCHD provides a user interface for the startup and termination of scheduled tasks.

Scheduled tasks are defined in the Scheduled Task Table template which is used by MSI$ to build the Scheduled Task Table at system initialization. Entries in the template have the following fields:

OF$SCHD:TEMPER -	Period (in 20 millisecond units) at which task will run
OF$SCHD:TEMADR -	Task entry point
OF$SCHD:TEMLVL -	Level (low order bits) and segment (high order bit) at which module will run

A scheduled task is terminated when the scheduled task calls MTC$SCHD:TERM as follows:

<u>Entry Conditions</u>

* The scheduled task information in data space must be unmodified and the stack pointer must be the same as when the scheduled task was started.

<u>Exit Conditions</u>

Task is terminated.

5.1.1.2.4 MTC$DELAY, Delay Submodule

MTC$DELAY allows the user task to suspend itself for any desired time period (expressed in milliseconds) and automatically resume when the period is finished. MTC$DELAY is invoked by a software interrupt where the two bytes following the SWI instruction contain the address of MTC$DELAY:ENTRY. The calling sequence is as follows:

<u>Entry Conditions</u>

* Called by SWI as noted above. Caller <u>must not</u> be a hardware task. X-register contains length of desired delay in milliseconds.

<u>Exit Conditions</u>

* Task is resumed at location three bytes after the invoking SWI instruction after the desired delay has expired. All registers are preserved.

5.1.1.2.5 MTC$RTC, Real-Time Clock Handler

MTC$RTC is the handler vectored to by MTC$MAIN whenever a real-time clock hardware service request is detected.

MTC$RTC maintains a delay queue (described in the D814 Detailed Design Specification) for tasks suspended by MTC$DELAY and, when a task's delay timer has expired, it resumes the task.

MTC$RTC has no software interface external to MTC.

5.1.1.2.6 MTC$IDLE, D814 Idle Code

MTC$IDLE contains the code executed by a processor when MTC$MAIN can find no work for it to do. MTC$IDLE contains a loop whose instructions are roughly representative in proportion of VMA cycles as well as in proportion of memory write cycles. This loop is used to maintain the idle cycle counter (described earlier). At the end of each pass through the loop, MTC$IDLE checks if there are any service requests pending and, if so, goes to the MTC$MAIN dispatch logic.

MTC$IDLE's only interface with system components external to MTC is the idle cycle counter.

5.1.2 Buffer Management Submodule Group (MBM)

Introduction

The Buffer Management Submodule Group is part of the D814 Mainframe operating system. The MBM contains utility routines for maintaining the mainframe's free buffer pool. The buffers in this pool are the dynamic memory units which tasks can obtain and return in real-time to meet such memory requirements as temporary data storage, input/output character buffering, and intertask communications message buffers.

General Description

The MBM has two main functions. The first function is to maintain the D814 mainframe free buffer pool and to keep the statistical information necessary for determining buffer utilization. The second function is to provide useful buffer utility features for the system in a central software module. Two buffering utilities are provided. The first is a general byte file utility and the second is a byte queue buffer utility.

1. Free Buffer Pool Management

   The D814 software system maintains a pool of fixed size free buffers so that tasks in the system may be able to dynamically obtain memory resources. This pool is created at system initialization time by the System Initialization Module and is maintained during system execution by the Free Buffer Management Submodule (FBMS). Tasks can obtain and return buffers from this pool by calling subroutines in the FBMS. The pool is kept as a queue so that a historical record of buffer use is available and so that background memory diagnostics which will test all of the buffer pool can be implemented. The buffer manager maintains a count of the total number of buffers in the free pool and a count of those presently allocated to software tasks. These numbers are used to calculate buffer utilization statistics.

   The buffer pool has two operating modes - normal and priority. When the number of free buffers in the pool is less than a specified threshold, the buffer pool goes into "priority" mode. In this mode, only "priority" get buffer requests are allowed to be successful. The purpose of the priority mode is to control buffer pool underrun. In priority mode, system software modules that need buffers but are low priority suspend operation until the buffer pool builds back up again to an acceptable level and the pool reenters normal mode. When the pool goes into priority mode, a flag is set so that a monitoring task can report the condition at some later time.

Fields Defined for System Buffers

0 - 3	4 - 5	6 - D	E - F
QUEUE LINKS	LBPTR	...	LNKPTR

QUEUE LINKS - 4 bytes for use by queue utility (see subsection on Main-
frame Utility Submodule Group)
LBPTR - Pointer to last buffer in list
LNKPTR - Pointer to next buffer in list

The following operations are available on the free buffer pool:

A.

Routine GBUF - Obtains one buffer from the free buffer pool

Entry Point - MBM$FBMS:GBUF_PRI - high priority entry

Entry Point - MBM$FBMS:GBUF - low priority entry

Entry Conditions

* None

Exit Conditions

A-register = destroyed
B-register = unchanged

* If buffer available:

X-reg = address of buffer
CC:Z = 0
CC:I = 0

* If buffer not available:

CC:Z = 1 and CC:I = 0
A,X-registers = destroyed
B-register = unchanged

B.

Routine RBUF - Return one buffer to the free buffer pool

Entry Point - MBM$FBMS:RBUF

Entry Conditions

* X-reg = Address of buffer

Exit Conditions

* CC:I = 0
* A,X registers = destroyed
* B-registers = unchanged

Entry Point - MBM$FBMS:RBUF_SP

Entry Conditions

* X-reg = Any address in returned buffer

Exit Conditions

* CC:I = 0

C.

Routine RLIST - Returns list of "n" buffers

Entry Point - MBM$FBMS:RLIST

Entry Conditions

* X-reg = Address of list of buffers to be returned
  LBPTR of first buffer = pointer to the last buffer in the list
  NBUFS of first buffer = count of "n" buffers in the list

Exit Conditions

* CC:I = 0
* All registers and data space locations OE$DS:BFADR and OR$DS:BFTBP are destroyed.

D.

Routine RCHAIN - Returns list of buffers

Entry Point - MBM$FBMS:RCHAIN

Entry Conditions

* X-reg = Address of returned list of buffers
  LNKPTR of the last buffer in the list must be null

Exit Conditions

* CC:I = 0
* All registers and data space locations OF$DS:BFADR and OF$DS:BFTMP are destroyed.

2. Byte File Buffer Utility

MBM$BFILE provides a utility submodule for creating, deleting, and maintaining a byte file buffer system. These byte file buffers are not multiprocessor interlocked so only one task may be using a byte file buffer at any one time.

Byte files are used for many crucial operating system functions and therefore use the priority buffer routine MBM$FBMS:GBUF_PRI to obtain buffers as needed.

The structure of a byte file is that it has a header buffer pointing to a list of buffers, each linked to the next with the last 2 bytes. The format of the header buffer is:

0 : 1 : 2 : 3	4 : 5	6	7	8	9	A	B : C : D	E : F

BYTES 0 - 3	- Reserved for linking files to lists
BYTES 4 & 5	- Pointer to last buffer in file
BYTE 6	- Total number of buffers making up file
BYTE 7	- Number of bytes allocated in file body
BYTE 8	- Address of highest written byte
BYTE 9	- Address of last byte written
BYTE A	- Address of last byte read
BYTES B -> D	- Not used
BYTES E & F	- Pointer to first buffer of file body

The file body is composed of a linked list of buffers where the first 14 bytes of each buffer are byte file data storage, and the last 2 bytes are a link pointer to the next buffer in the file body. The last link pointer in the file is zero.

The following functions are provided for manipulating byte file buffers:

A.

Routine CREATE - Creates a byte file

Entry Point - MBM$BFILE:CREATE

Entry Conditions

* None

Exit Conditions

* X-reg = File descriptor address
* CC:Z = CC:I = 0
* If no priority buffer is available, a system error occurs.

B.

Routine DELETE - Deletes a file

Entry Point - MBM$BFILE:DELETE

Entry Conditions

* X-reg = File descriptor address

Exit Conditions

* CC:I = 0

C.

Routine READ - Reads byte "n" from a given file

Entry Point - MBM$BFILE:READ

Entry Conditions

* B-reg = Byte address "n"
* X-reg = File header address

Exit Conditions

* A-reg = Contents of the n^{th} byte
* CC:V = "Out of range" error
* X-reg = unchanged
* Data space - OF$DS:BFADR points to byte file header. OF$DS:BFTMP destroyed.

D.

<u>Routine</u> SREAD - Reads sequentially "next" byte from a file

<u>Entry Point</u> - MBM$BFILE:SREAD

<u>Entry Conditions</u>

* X-reg = File header address

<u>Exit Conditions</u>

* A-reg = Contents of the "next" byte
* B-reg = Address of "next" byte in file
* CC:V = "Out of range" error
* X-reg and data space
* Same as for READ

E.

<u>Routine</u> WRITE - Writes into the nth byte of file

<u>Entry Point</u> - MBM$BFILE:WRITE

<u>Entry Conditions</u>

* A-reg = Data byte to be written to file
* B-reg = Byte address "n"
* X-reg = File header address

<u>Exit Conditions</u>

* CC:V = "Unable to write" message
* CC:I may be cleared to 0
* If no priority buffer is available, a system error occurs
* B-reg = Address of "next" byte in file
* X-reg and data space same as for READ.

F.

<u>Routine</u> SWRITE - Writes sequentially into "next" byte of file

<u>Entry Point</u> - MBM$BFILE:SWRITE

<u>Entry Conditions</u>

* A-reg = Data byte to be written to file
X-reg = File header address

Exit Conditions

* B-reg = Address of "next" byte
* CC:V = "Unable to write" message
* If no priority buffer is available, a system error occurs
* CC:I may be cleared to 0
* X-reg and data space same as for READ.

3. Byte Queue Buffer Utility

MBM$BQUE provides a utility submodule for creating, deleting, and maintaining byte queue data structures. The byte queues are multiprocessor interlocked so that one task may be putting bytes into a byte queue while another task may be removing bytes from the queue. Because of interlocking, 0 may not be stored in the byte queue. Byte queues have no maximum size but, since they only use low priority buffers, the total amount of memory dedicated to byte queues is limited by the size of the low priority buffer pool.

The first buffer, known as the queue descriptor, has the following format:

0 - 3	4 - 5	6	7 - 8	9 - 10	11 - 13	14 - 15
N/A	Last Buffer	# Buffers -1	Head Pointer	Tail Pointer	N/A	Link Pointer

BYTES 0 - 3	- Are reserved for use by the Queue Utility routines
BYTES 4 & 5	- Point to the last buffer in the list
BYTE 6	- Contains the number of buffers -1 in the list
BYTES 7 & 8	- Point to the next byte to be read
BYTES 9 & 10	- Point to the next byte to be written
BYTES 11 - 13	- Currently unused
BYTES 14 & 15	- Standard buffer link to the first data buffer

The next byte to be written (pointed to by tail pointer) always contains binary zeroes. When a byte is to be written into the byte queue, the next byte is cleared to zero and then the new data byte is written. This allows the "get" routine to check for an empty queue without having to disable interrupts and compare head and tail pointers. It simply gets the byte pointed to be the head pointer; if it is zero, the queue is empty.

The byte queue routines consist of four user called subroutines:

A.

Routine CREATE - Creates a byte queue

Entry Point - MBM$BQUE:CREATE

Entry Conditions

* None.

Exit Conditions

* If available:

  X-reg = Address of queue descriptor
  CC:Z = 0
  CC:I = 0

* If not available:

  CC:Z = 1
  CC:I = 0

B.

Routine DELETE - Deletes a byte queue

Entry Point - MBM$BQUE:DELETE

Entry Conditions

* X-reg = Queue descriptor address

Exit Conditions

* CC:I = 0
* All registers and data space location OF$DS:BFADR destroyed.

C.

Routine PUTBYT - Puts a byte into a queue

Entry Point - MBM$BQUE:PUTBYT

Entry Conditions

* B-reg = A byte of data
* X-reg = Queue descriptor address

Exit Conditions

* CC:Z = "Unable to enqueue" message
* CC:I may be cleared to 0
* X and B-register = unchanged
* A-register and data space locations OF$DS:BFADR and OF$DS:BFTMP are destroyed.

D.

Routine GETBYT - Gets a byte from a queue

Entry Point - MBM$BQUE:GETBYT

Entry Conditions

* X-reg = Queue descriptor address

Exit Conditions

* B-reg = Dequeued byte from queue
* CC:Z = "Empty queue" condition
* CC:I may be cleared to 0
* X-register = unchanged
* A-register and data space location OF$DS:BFADR destroyed.

5.1.3 Mainframe Utilities

The Mainframe Utilities (MUT) is a group of submodules providing services used by various mainframe modules. MUT includes these submodules:

MUT$BUF	- Buffer submodule
MUT$DELAY	- Delay submodule
MUT$MULT	- Multiplication submodule
MUT$DIV	- Division submodule
MUT$PCB	- Port Control Block submodule
MUT$QUE	- Queue manipulation submodule
MUT$AP	- Addressed packet submodule
MUT$SPD	- Data speed encode-decode submodule

The remainder of this subsection describes the individual submodule components of MUT.

5.1.3.1 MUT$BUF

MUT$BUF provides routines to wait for a buffer to be available. MUT$BUF is obsolete and should be deleted as soon as possible. It is presented here only for the sake of completeness. MUT$BUF has these entry points:

Entry - MUT$BUF:GBW

* Attempts to get a low priority buffer, waiting a specified period between retries

Entry Conditions

* X-reg = Length of time in milliseconds to wait between retries

Exit Conditions

* X-reg = Buffer address
* A, B register = Destroyed
* Data space = OF$DS:BFTMP destroyed
* CC:I=0

Entry - MUT$BUF:GBW10
MUT$BUF:GBW25
MUT$BUF:GBW50

These entries are identical to MUT$BUF:GBW except that a fixed delay period of 10, 25, or 50 milliseconds, respectively, is used.

5.1.3.2 MUT$DELAY

MUT$DELAY provides a convenient interface with MTC$DELAY, the MTC delay routine. It has these entry points:

Entry - MUT$DELAY:EXEC

Entry Conditions

* X-reg = Contains delay length in milliseconds

Exit Conditions

* Returns to user after task has been suspended for desired period
* A and B registers are preserved
* X-reg is destroyed

Entry - MUT$DELAY:DL10
MUT$DELAY:DL25
MUT$DELAY:DL50
MUT$DELAY:DL100
MUT$DELAY:DL1000

Identical to MUT$DELAY:EXEC except that delay is for fixed period of 10, 25, 50, 100, or 1000 milliseconds, respectively.

5.1.3.3 MUT$MULT

MUT$MULT multiplies two 8-bit unsigned numbers and returns a 16-bit result. The calling sequence is as follows:

Entry - MUT$MULT:ENTRY

Entry Conditions

* A and B registers each contain one multiplicand

Exit Conditions

* A, B registers contain the 16-bit result
* X-reg = Destroyed
* Two bytes of stack are used for scratch

5.1.3.4 MUT$DIV

MUT$DIV divides an 8-bit unsigned integer into a 16-bit unsigned integer to give an 8-bit remainder and a 16-bit result. The calling sequence is as follows:

Entry - MUT$DIV:ENTRY

Entry Conditions

* A, B = 16-bit dividend
* Stack = 8-bit divisor on top of stack (must not be 0)

Exit Conditions

* A-reg = Destroyed
* B-reg = Remainder
* X-reg = Quotient
* Stack = Destroys the divisor field but does not pull it off stack. Uses 5 scratch bytes on stack in addition to divisor field.

5.1.3.5 MUT$PCB

MUT$PCB contains utilities for accessing the D814 Port directory (see section on Subsystem Data Structures). These routines provide a convenient interface with the necessary synchronization for adding and deleting Port Control Blocks (PCB's) as well for locating the PCB for a port. The remainder of this subsection describes the entry points into MUT$PCB.

Entry - MUT$PCB:ADR

Returns PCB address (if any) for a given port ID

Entry Conditions

* B-reg = Port ID

Exit Conditions

* A-reg = Port type (see Subsystem Data Structures)
* B-reg = Port ID if port exists; otherwise, destroyed
* X-reg = PCB address if port exists; otherwise, destroyed
* CC:Z = Cleared if and only if the port exists
* Data space = OF$DS:BFTMP is destroyed

Entry - MUT$PCB:ADDPORT

*   Adds a PCB to the port directory, assigning the PCB a port address (ID)

Entry Conditions

*   X-reg = Points to PCB. The PCB and all necessary substructures must already be set up

Exit Conditions

*   X,A-registers = Destroyed
*   Data space = OF$DS:BFTMP destroyed
*   B-reg = Port ID if room in directory; else, 0
*   CC:Z = Clear if and only if there was room in the directory

Entry - MUT$PCB:DELETEPORT

*   Removes a port from the port directory

Entry Conditions

*   B-reg = Port ID

Exit Conditions

*   Port is removed from directory
*   A and X registers are destroyed
*   B-reg = Unchanged

5.1.3.6 MUT$QUE

MUT$QUE provides interlocked routines for enqueueing to and dequeueing from D814 mainframe queues. Each D814 mainframe queue must have a 6-byte queue descriptor of the following format:

OF$MISC:QTOP - Address of link field of first queue entry (oldest item on queue); zero if queue empty

OF$MISC:QBOT - Address of link field of last queue entry (newest item on queue); zero if queue empty

OF$MISC:QLOCK - Address of lock byte for the queue

Entries are linked onto the queue by a 2-byte link field. Each link field contains the address of the link field of the next entry on the queue or 0 if it is the end of the queue.

MUT$QUE has these entry points:

Entry - MUT$QUE:ENQ

Entry Conditions

* X-reg = Contains address of queue descriptor
* Data space = OF$DS:QUEADR (= OF$DS:BFTMP) contains address of link field of entry to be enqueued

Exit Conditions

* A, B, and X registers are destroyed
* CC:Z = Set if and only if queue was empty on entry
* Entry is enqueued on the proper queue

Entry - MUT$QUE:DEQ

Entry Conditions

* X-reg = Address of queue descriptor

Exit Conditions

* X-reg = Address of entry dequeued; unpredictable if queue empty on entry to routine
* A-reg = Destroyed
* B-reg = Set to 1 if queue empty after dequeue; set to 2 if queue not empty
* CC:Z = Set if and only if queue empty on entry to routine

5.1.3.7 MUT$AP

MUT$AP provides two routines to aid in handling Addressed Packets (see section on System Data Structures). These are the entry points:

Entry - MUT$AP:SEND

* Sends an Addressed Packet

Entry Conditions

* X-reg = Points to Addressed Packet byte file header. Packet must be set up with all necessary fields filled in

Exit Conditions

* X-reg = Unchanged
* A, B registers = Destroyed
* Data space = OF$DS:BFADR, OF$DS:BFTMP are destroyed
* There are no error conditions

Entry - MUT$AP:SDSWAP

* Swaps source and destination parameters, resetting delivery error indicator

Entry Conditions

* X-reg = Address of Addressed Packet byte file header

Exit Conditions

* A, B, and X registers = Unchanged
* Data space = OF$DS:BFTMP destroyed
* Addressed Packet = Contents of offsets OF$MAP:SRCND, OF$MAP:SRCPT, and OF$MAP:SRCMOD are interchanged with offsets OF$MAP:DSTND, OF$MAP:DSTPT, and OF$MAP:DSTMOD, respectively. The error indicator (high order bit of original OF$MAP:DSTND) is reset.

5.1.3.8 MUT$SPD

MUT$SPD provides routines for encoding and decoding 16-bit link and path speeds into a 1-byte number in a sort of floating point format. The encoded speed is composed of a four-bit exponent (high order nibble) and a four-bit mantissa (low order nibble).

The actual speed is computed as:

$S = (16\ 1/2 + B)2^A - 16$ (truncated if not an integer)

where

S = Actual speed
A = Exponent
B = Mantissa

The encoded speed exponent and mantissa are computed as:

$A = [\log_2 (S + 16)] - 4$ (truncated if not an integer)

$B = \frac{S + 16}{2^A} - 16$ (truncated if not an integer)

This encoding scheme results in accuracy better than ±6.3 percent for speeds greater than 15 and better than ±3.2 percent for greater than 1008.

Encoded speeds are continuous in that, if A and C are encoded speeds and A < B < C, then B is a valid encoded speed and the actual speed represented by B is less than that represented by C and greater than that represented by A.

MUT$SPD has the following entry points:

Entry - MUT$SPD:ENCODE

Entry Conditions

* A, B registers = Contain 16-bit speed

Exit Conditions

* A-reg = Speed in "floating point" format
* B-reg = Destroyed
* X-reg = Preserved

Entry - MUT$SPD:DECODE

Entry Conditions

* A-reg = Speed in "floating point" format

Exit Conditions

* A, B registers = Actual speed (if no overflow)
* X-reg = Preserved
* CC:C = Set if and only if overflow out of 16th bit occurs in decoding. If CC:C is set, then A, B contain X'FFFF'

5.1.3.9 MUT$BF

A routine which quickly copies a byte file. It creates a new byte file and copies an old byte file into it.

Entry - MUT$BF:COPY

Entry Conditions

OF$DS:BFADR = Pointer to old byte file

Exit Conditions

If copy successful:

```
CC:Z=0
X-Reg = Pointer to new byte file
A-Reg & B-Reg are destroyed
OF$DS:BFADR is destroyed
```

If copy unsuccessful:

```
CC:Z=1
A-Reg, B-Reg and X-Reg are destroyed
OF$DS:BFADR is destroyed
```

5.1.4 Mainframe Programming

This subsection is designed to help the new D814 mainframe programmer write code for the mainframe running under the operating system. It is assumed that the reader is familiar with the M6800 instruction set and has a basic understanding of the hardware and operating system environment.

1. Code and Data Areas in the D814 System

   Code and data areas in the D814 mainframe (in fact, in the entire system) are rigidly separated.

   Code may not be modified after system downline load. Code areas are periodically checksummed by the background diagnostic module, MDM, and any modified code would throw this checksum off.

   Data space (and local storage if interrupts are off) may normally be used for scratch data storage, with certain restrictions (see subsection on MTC). Page 0 and System Area (addresses X'80' to X'FE' and X'120' to X'1FF')may be used for permanently allocated storage. Large blocks of permanently allocated storage may be reserved in data-only submodules. Dynamic data storage is provided by system buffers and by byte files (see subsection on MBM).

2. Intertask Communication and Synchronization

   As in any multiprocessing system, communication and synchronization among asynchronous tasks is a particularly sticky problem. The most basic facility for task synchronization is the lock byte (see Subsystem Data Structures). Messages may be sent among tasks executing in the same mainframe using batch queues (see subsection on MTC$BATCH). Addressed packets may be used, although they involve more overhead.

3. Hardware Data Spaces

Special coding rules apply to tasks running in hardware data spaces (see 6000 Logic Design Spec). Interrupts may not be enabled while executing in a hardware data space because that would allow the data space to be assigned to another hardware service task.

4. Stack

MTC is normally responsible for setting up the stack for a task. It is the user's resonsibility to ensure that stack overflow does not occur. In particular, the user should remember that if interrupts are not masked seven bytes of stack must be available for saving the processor state on interrupt.

5. Hardware Interface

D814 system components exist to handle many hardware interfaces. Where such components exist they should not be circumvented. For example, all panel IO should go through module MPC and starting, stopping, changing of priority levels, etc., for tasks should be done through MTC. Some simple hardware functions, on the other hand, are done directly by executing Master Controller instructions. Among these are memory segment switching, manipulation of the 6000 Master Controller interrupt mask, and read-only operations such as reading processor status.

5.2 Mainframe Addressed Packet Control Module (MAP)

The Mainframe Addressed Packet Control Module is responsible for handling addressed packets (see section on System Data Structures) within the Mainframe. MAP has significant interfaces with these system components:

I/P (through the packet FIFO)

Mainframe Network Link Control Module (MNL)

Mainframe Path Management, Routing, and Congestion Control Module (MPMRCCM)

Senders and receivers of addressed packets within the local mainframe

5.2.1 Overview of MAP Addressed Packet Handling

All addressed packets except those sent between modules within the same I/P must pass through MAP for routing.

Addressed packets are received into MAP from three sources:

Addressed packets from remote nodes are received by MNL from the I/NP through the BIC data FIFO, along with user data (see subsection on MNL). MNL separates the addressed packets from data and sends them to MAP.

Addressed packets originating in modules within a local I/P are sent into the mainframe through the BIC packet FIFO. It should be noted that even modules within the I/NP send addressed packets destined to modules external to the I/NP through the BIC packet FIFO. MAP is responsible for all the physical I/O involved in handling the BIC packet FIFO. (See Bus Interface Chip Specification.)

Addressed packets originating within the Mainframe are enqueued directly to MAP.

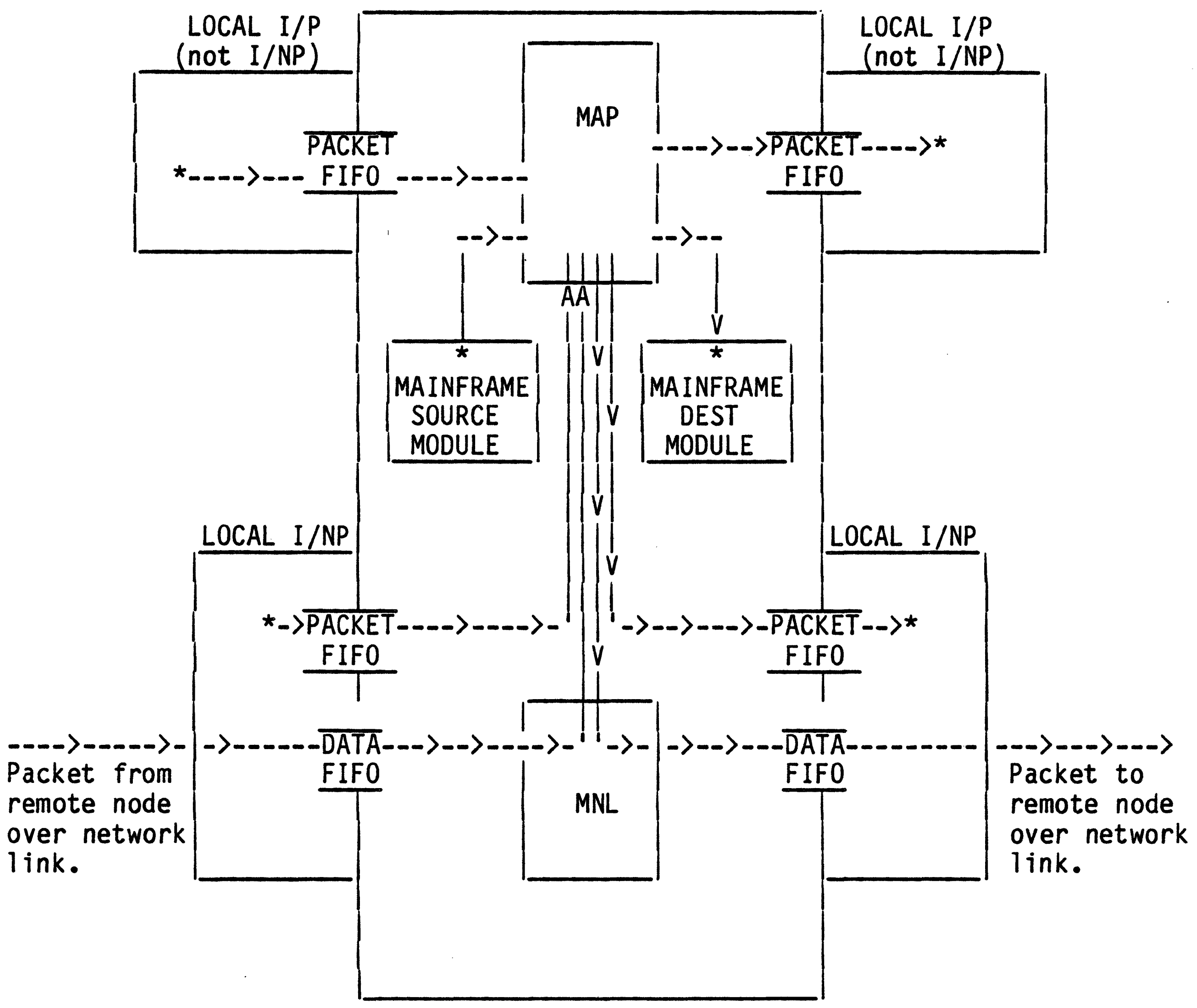
Addressed packets leave MAP in these three ways:

Addressed packets having a local mainframe module as destination are sent directly to the destination module.

Addressed packets having a local I/P as destination are sent there through the BIC packet FIFO. (This includes packets going to modules within the local I/NP.)

ADDRESSED PACKET HANDLING WITHIN A D814 NODE

MAINFRAME



*Source or destination of a packet.

Addressed packets having a module within a remote node as destination are passed to MNL for inclusion in the data sent over an I/NP BIC data FIFO. This data is transmitted by the I/NP over the network link. The included figure illustrates the above.

If an error is detected in an addressed packet, the source and destination address fields are interchanged, the delivery error bit is set, one byte error code is appended to the packet bytefile (increasing the packet byte count by one), and the packet is sent back to the source.

It should be noted that packets leave MAP in the order in which they are received into MAP.

5.2.2 MAP External Interfaces

This section describes all significant external interfaces with MAP.

5.2.2.1 MAP I/P Packet FIFO Interface

As already stated, MAP has complete responsibility for handling the mainframe side of the BIC inbound and outbound packet FIFOs during normal system operation. (The packet FIFO is also used by MSI during initialization, but that does not concern us here.)

The inbound and outbound BIC packet FIFOs are used as buffers rather than as true FIFOs. A packet FIFO may not be read by the reader until the sender has finished filling it and once reading has begun the sender may not write until the reader has emptied it.

This protocol is implemented by means of the sender's and reader's flag associated with each FIFO. The sender's flag is set by the sender (MAP if outbound FIFO or I/P software if inbound FIFO) to indicate that it may be read by the reader (I/P software if outbound FIFO or MAP if inbound FIFO) and the reader's flag is set by the reader to indicate that the FIFO has been emptied. The reader's flag must be cleared by the sender before or at the same time as the sender's flag is set, and the sender's flag is cleared by the reader before or at the same time as the reader's flag is set. When the sender's flag is set, the FIFO is said to be filled, even though there may be room for move data in the FIFO.

Packets being sent over a BIC FIFO are broken up into segments whose byte count is less than or equal to one less than the capacity of the FIFO.

Before writing a segment to a packet FIFO, the sender first writes the segment byte count. More than one segment may be placed in the packet FIFO when it is filled, although the reader is not required to read every segment in the FIFO when the FIFO is read. But the reader of a packet FIFO may not leave a segment partially read when it sets the reader's flag and the sender may not leave a segment partially written when it sets the sender's flag. It should be remembered that if the sender places more than one segment in the FIFO it is the sender's responsibility to check after the FIFO is read to see if the FIFO is empty and, if not, to set the sender's flag again.

MAP services level 4 service requests generated when the inbound packet FIFO sender's flag is set and when the outbound packet FIFO reader's flag is set. These requests are vectored to entry point MAP$PINT:ENTRY by MTC (see subsection on Mainframe Operating System).

The Packet Structures in I/P PCB's are used by MAP in transmitting and receiving packet data over the BIC. Rack substructure has both an outbound and an inbound addressed packet queue. The outbound addressed packet queue holds packets waiting to be sent to the I/P, the inbound addressed packet queue holds received packets waiting to be distributed.

5.2.2.2 MAP MNL Interface

This subsection describes the interface between MAP and the Mainframe Network Link Control Module (MNL).

As may be seen from the diagram in the Overview subsection any addressed packet going to or from a remote node via the local node passes through MNL between MAP and the I/NP. This interface is implemented through two queues.

When MNL gets an addressed packet in the data stream from the I/NP it recognizes it as such and enqueues it on the MAP batch queue using utility MUT$AP:SEND (see subsection on Mainframe Utility Module). This activates MAP at entry MAP$ROUTE:ENTRY. The packet is then routed as described later in this subsection.

When MAP gets an addressed packet whose destination node is not the local node it enqueues it to the Remote Addressed Packet Queue associated with the proper link (see subsection on MPMRCCM interface). It uses MNL utility entry MNL$UTIL:Q2RAPQ to do this. MNL then either includes it in the link data stream through the BIC data FIFO (see subsection on MNL) or, if the link goes down before it can be transmitted, re-routes it through MAP.

5.2.2.3 MAP MPMRCCM Interface

MAP calls MPMRCCM entry point MRM$ROUTE:PACKET to determine the link over which to route a packet destined for a remote node. MRM$ROUTE:PACKET is described in the subsection on MPMRCCM.

5.2.2.4 MAP User Interface

This subsection describes the MAP interface with senders and receivers of addressed packets within the local mainframe.

A mainframe module wishing to send an addressed packet enqueues the packet to the MAP batch queue using subroutine MUT$AP:SEND.

Addressed packets may be delivered to a mainframe module in two ways, depending on the kind of module:

1) If the module is a batch task, the packet is enqueued to the module's batch queue, causing the task to start up, if necessary, at its entry point and process the packet (see MTC subsection for description of batch tasks). The entry point and the batch queue address are found by consulting the Module Dispatch Table (see subsystem Data Structures).

2) If the module is not a batch task, the module is forked at the entry point contained in the Module Dispatch Table with the X register containing the address of the bytefile.

Whether or not the module is a batch task is determined by consulting the Module Dispatch Table.

5.3 Mainframe Statistics and Monitoring, and Reporting, Module

The Mainframe Statistics and Monitoring Module is responsible for maintaining statistical information for the node as a whole (but not for the individual ports), providing that information on request to other local or remote modules, and routing system reports of various danger conditions to the designated system report node and port.

5.3.1 Functional Description

The MSM is responsible for maintaining the following statistics:

1. Processor loading, the percentage of non-idle processor time.

2. Buffer utilization, the average percentage of buffer storage in use.

3. Apparent throughput, the maximum physically constrained channel capacity of all the terminal ports plus the transfer ports. (Apparent throughput changes as terminals establish and disconnect active paths.)

4. Statistical throughput, the average combined rate for characters coming into the mainframe from terminal ports and leaving the mainframe for remote nodes.

The MSM performs the following functions:

1. The MSM runs periodically as a scheduled task every Pm seconds (Pm defined by equate EQ$MSM:PM) to update statistical accumulations for the above statistics. Averages of the Pm-second samplings are kept for a configuration-defined time period called an "averaging period". Pm is currently set at 6 and must evenly divide 360 to guarantee that there will be an even number of sampling periods in an averaging period. The scheduled task entry point is MSM$MONITR:ENTRY.

2. The MSM monitors some of the above statistical accumulations on its scheduled run. If certain configuration determined thresholds are exceeded, it sends a system report(s) (see below). There are thresholds associated with processor loading and buffer utilization.

3. The MSM maintains weighted time-averages for each of the statistics it maintains.

   The weighted time-average (WTA) of a quantity at the end of an averaging period N is defined as WTA(N) = (1/2) (WTA (N-1) +S) where WTA (N-1) is the weighted time-average for the previous averaging period and S is the average value of the quantity as sampled at Pm-second intervals during the last averaging period. W(0) is taken to be 0.

These weighted time-averages are updated periodically at the expiration of each averaging period.

4. The MSM also handles addressed packets requesting statistical data. It responds to such requests by sending the requestor an addressed packet containing all its weighted time-averages extrapolated to reflect as closely as possible the current system state.

5. The MSM routes system reports received from external modules, as well as from within MSM, to the designated system report node and port.

5.3.2 Message Interface

MSM interfaces with the modules to which it supplies information by means of addressed packets. The addressed packets and their contents are described here.

5.3.2.1 Statistics Request Addressed Packet

The Statistics Request Addressed packet is sent to MSM (module number EQ$MDT:MSM_STATIS) by a module desiring mainframe statistics. The entry point for the MSM addressed packet batch task is MSM$AP:ENTRY. The MSM reverses the source and destination fields and sends the packet back to the sender with these fields filled in:

OF$MSM:AP_LOADING - Processor loading percentge (1 byte)

OF$MSM:AP_BUFFUTIL - Buffer utilization percentage (1 byte)

OF$MSM:AP_APTHRU - Apparent throughput characters per second (2 bytes)

OF$MSM:AP_STATTHRU - Statistical throughput characters per second (2 bytes)

OF$MSM:AP_MFCAP - Mainframe combined processor capacity in K MPU cycles per millisecond. This number is computed at initialization and does not change.

All but the last of these fields is computed from the WTA at the end of the last averaging period and extrapolated to the present as:

(1/2KMAX) (WTA*(2KMAX-K)+S*K) where

WTA is the weighted time-average for the previous averaging periods.

S is the average as sampled so far during the current averaging period.

K is the number of Pm-second samplings which have occurred so far during the current averaging period.

KMAX is the number of samples in an averaging period.

It can be seen that statistical information provided soon after or soon before the expiration of an averaging period (assuming the averaging period is much longer than Pm) will be close to the weighted time-average computed at the end of the averaging period.

5.3.2.2 System Report Addressed Packet

As already stated, MSM is responsible for routing system reports. MSM also originates system reports when any of its thresholds are exceeded. A system report is an addressed packet with these fields:

OF$SYSRPT:CODE - Code for the event being reported. MSM sends reports for processor loading threshold exceeded (code = EQ$SYSRPT:XPROCLOAD) and for buffer utilization threshold exceeded (code = EQ$SYSRPT:XBUFFUTIL).

OF$SYSRPT:P1 - First report parameter (some system components may use more than one parameter). When sending a processor loading or buffer utilization report, MSM places the offending percentage here.

The system report addressed packet is sent to module number EQ$MDT:MSM_SYSRPT where it is handled by the system report router at entry point MSM$SYSRPT:ENTRY. MSM then routes the packet to module EQ$IP$MDT:REPORT_SPLQ at the designated system report node and port (see subsection on Configuration Parameters).

5.3.3 Collection of Raw Statistics

The raw data used in computing MSM's statistics is collected, as already stated, every Pm seconds. This subsection describes the module interfaces involved in the collection of raw statistics.

Processor Loading Statistic

MTC maintains a count of tens of processor idle cycles executed (OF$PGO:TCIDLC) which is read and reset by MSM at each scheduled run (see MTC). MSI, the system initialization module, computes the number of micro-cycles per millisecond available from each processor, storing it in OF$SYS:MSM_CPS and the number of processors, storing it in OF$PGO:SY_NOPS. MSM in its first scheduled run multiplies these numbers and stores the total microcycles per millisecond back in OF$SYS:MSM_CPS. From OF$PGO:TCIDLC and OF$SYS:MSM_CRS MSM computes the processor loading statistic, the percentage of non-idle processor cycles compared to total processor cycles available.

Buffer Utilization Statistic

This is computed from field OF$PGO:BFMAX, the total number of system buffers (computed by MSI), and field OF$PGO:BFCNT, the count of currently available buffers (maintained by MBM). OF$PGO:BFCNT is interlocked through lockbyte OF$SYSLCK:BFPOOL.

Apparent Throughput Statistic

Raw data for the apparent throughput calculation comes from the MNL-maintained field OF$PCB:XMT_APTHRUPUT in the transmit data substructure of each I/NP PCB. This field is interlocked through the STATIS lockbyte associated with the PCB (see Subsystem Data Structures).

Statistical Throughput Statistic

MNL maintains accumulated receive and transmit character counts for each link in fields OF$PCB:RCV_CHRCNT and OF$PCB:XMT_CHRCNT in the receive and transmit data substructures, respectively, of each I/NP PCB. These fields are read and reset by MSM at each scheduled run and are used to update the statistical throughput accumulation. They are also interlocked through the STATIS lockbyte.

5.3.4 Configuration Parameters

MSM uses various configuration node parameters defined by the user through the ICTP and read into mainframe memory by MSI at system initialization. These paramters are:

OF$PGO:SY_BUTH - Buffer utilization threshold. When buffer utilization percentage (averaged over one Pm-second period) exceeds this number, a system report is sent.

OF$PGO:SY_PLTH - Processor loading threshold. When processor loading percentage (again averaged over one Pm-second period) exceeds this number, a system report is sent.

OF$PGO:SY_AVTC - Averaging Time Constant. This one-byte field controls the length of the averaging period. If AVTC is 0, then the averaging period is Pm seconds. Otherwise, the averaging period is five minutes times the contents of AVTC.

OF$PGO:SY_RPTN - System report node. System report addressed packets are sent to this node by the MSM system report router, MSM$SYSRPT.

OF$PGO:SY_RPTP - System report port. System report addressed packets are sent to this port at the above node.

5.4 Mainframe Panel Control (MPC) Module

5.4.1 Introduction

The Mainframe Panel Control Module interfaces the operator at the front panel of the mainframe with the local node (Mainframe, BICs and IPs).

Unlike an I/CTP, the Mainframe Panel has a very limited user interface in hardware which is hardly enough to provide any extensive man-machine dialog without being much too cumbersome. Therefore, the Mainframe Panel is delegated to perform just simple functions under normal operating conditions.

The Mainframe Panel Control module controls the 32-character Self-Scan, mode indicator LEDs, and the 18-key keyboard through which Panel commands may be entered. It also triggers each processor's status to be displayed in turn by 12 processor LED's. The Panel mode is dictated first by the position of the keylock switch, and then the most recent mode-selecting Panel command.

5.4.2 Panel Modes and Commands

There are three Panel modes which can be set from the keylock switch: monitor (MON) mode, program (PGM) mode, and diagnostics (DIAG) mode.

There is also an internal mode called control (CTRL) mode which can be entered from the PGM mode by the use of a Panel command. Only in this CTRL mode is it allowed to perform a potentially risky operation. Therefore, entry into the CTRL mode is protected by a password.

The MON mode is a subset of the PGM mode while the PGM mode itself is a subset of the CTRL mode.

DIAG mode is the same as monitor mode provided no keystrokes are hit prior to turning to DIAG mode, otherwise an NMI is generated on the next keystroke, which is not serviced by the Mainframe Panel Control Module.

Panel commands are presented in detail in Section 5.1 of the D814 Product Functional Specification.

They can be classified into the following three categories.

1. Initialization - Load Commands

    1. BOOT command is used to optionally reboot the entire network to a new configuration.

    2. RSET - resets a port that is currently active by clearing paths and calls used by the port and resetting it causing ROM start-up diagnostics to run.

3. DIAG - causes a diagnostic program to be loaded and started in a port. Diagnostic runs to completion or until a RSET command is executed for the port.

4. LOAD - load and starts system software in a port previously reset.

All Initialization-Load commands are valid in PGM and CTRL modes.

2. Examination-Modification Commands

These commands are used to interrogate the status of the Mainframe or IP's at the local node.

The following four commands belong to this category:

1. DUMP - Dumps a memory block (Mainframe or IP)
2. EXAM - Reads an IP's BIC status registers
3. MEM - Reads a memory location allowing for the modification of its content (Mainframe or IP)
4. STEP - Steps through messages.

The DUMP, STEP and EXAM commands are valid in all modes, but the MEM command is allowed only in the CTRL mode since it will be used in unusual circumstances such as a partial system failure as a preliminary step to a full diagnostics procedure.

3. Auxiliary Commands

The purpose of these commands is to help an operator use the above commands easily, and also to permit him to get into a different Panel mode under which a different set of Panel commands are allowed.

The following four commands belong to this category:

1. HELP - Displays the local node number and available commands in the current mode
2. MON - Enters the MON mode
3. PGM - Enters the PGM mode
4. CTRL - Enters the CTRL mode

The first three commands, HELP, MON, and PGM commands, are valid in all modes, while the CTRL command is allowed only in the PGM mode. Furthermore, in order to activate the PGM or CTRL mode, the keylock switch must be set at the PGM position.

The mode command selecting the same mode which is currently in effect is allowed, but it does nothing.

5.4.3 Functional Submodule Description

There are five functional submodules in the MPC module as described below:

1. Initialization

Called whenever there is a restarting of the local node; it initializes the data structures for the Mainframe Panel Control module.

It also displays an initial system message

"NODE xx CONF n : CODEX 6050 INP"

on the Self-Scan, puts the Panel in the initial MON mode, and unlocks the keyboard.

Entry Point - MPC$INIT:START

Entry Conditions

* None

Exit Conditions

* A-reg = Destroyed
* B-reg = Destroyed
* X-reg = Destroyed

2. Scanning of LED Lights for Processor States

Running as a scheduled task every two seconds (Entry Point - MPC$SCAN:START), this submodule selects the next processor number so that the processor state is displayed in the 12 LED indicators at the left side of the Self-Scan display area by the Mainframe's Master Controller.

3. Interrupt Handler

A key stroke on the keyboard is entered into the Mainframe Master Controller as a Level 1 hardware interrupt which in turn invokes this interrupt handling hardware task.

When started, the task first locks the keyboard, and then examines the key value entered. If it is an "ENTER" or "STEP" signaling the completion of a command line, the task forks the Command Interpreter task and terminates itself. If it is a "CLEAR", the command line entered up to that point is flushed and the Panel is reinitialized. Otherwise, the key value is saved as part of the command line being assembled, the keyboard is unlocked, and then the task is terminated.

While building a command line, it also echoes a correct function key or character for each key pushed by referring to a command table.

When a syntax error is committed for a command by entering a key, a "BAD COMMAND" error message is displayed on the Self-Scan and the Panel is reinitialized.

Entry Point - MPC$INTR:HANDLER

Entry Conditions

```
*    CC:I = 1
*    SR   = 1LLVVVVV

          where LL = Keylock switch position
             VVVVV = Key value
          (Done by the Master Controller.)
```

Exit Conditions

```
*    CC:I = 0
     The task is terminated.
```

4. Command Interpreter

As the main processing body for Panel commands, this queued task is forked by the MPC Interrupt Handler when a command line is completed by an "ENTER" key.

It checks the legality of the command, and if it is legal, dispatches to the command-processing routines for either:

1. HELP message generation
2. Reading of IP's BIC Status Registers
3. Memory Dump/Read/Write
4. Mode Management
5. Boot Interface
6. MDM AP Interface
7. Message Display

If a command is found illegal, an appropriate error message is displayed on the Self-Scan and the Panel is reinitialized.

At the end of the command processing, the keyboard is unlocked to allow next command to be entered, and the task is terminated.

5. Panel Message Handling

When a message is received from the MAP router at entry point MPC$MES:ENTRY, it is added to the queue of waiting messages. Duplicate messages are tallied and flushed. The Panel is notified that a message is queued. Messages (including the number of duplicates) are removed from the queue and displayed at the Destination by a STEP command.

5.4.4 External Interfaces

The auxiliary commands and the EXAM command have no external interfaces. The interfaces for the other commands are referenced below.

RSET, LOAD AND DIAG

The actions for these routines are carried out by MDM. Therefore, MPC$ sends an addressed packet to MDM passing the port number and default parameters. See section on MDM for addressed packet format.

BOOT

The actual processing of the BOOT command is done by the Mainframe System Boot (MSB) module. Therefore, the MPC module only needs to pass a configuration number to the MSB$MPC:BOOT subroutine (refer to Section 5.5.2.1).

MEM and DUMP

MEM and DUMP commands send a 'nondestructive dump request' addressed packet to the specified IP or mainframe. The MEM command also sends a 'patch' addressed packet when the operator elects to modify an IP or mainframe memory location. The nondestructive dump and patch addressed packet format is described in IPOS software IP Upload and Memory Modification Utility (Section 6.1.11). The entry point used for response addressed packets is MPC$CMES:ENTRY.

MPC uses the same addressed packet interface for both mainframe and IP memory reference commands. Routing is handled by MAP. MPC merely copies the port number, memory address and appropriate command code into an addressed packet in all cases.

5.5 Mainframe System Boot (MSB) Module

5.5.1 Introduction

A system boot is the procedure of synchronizing the network and subsequently reloading software and/or reconfiguring it in an orderly fashion.

The boot facility is needed when an operator wants to load new sofware and/or change the configuration in a network, and also when an incompatibility in configuration is found in the network. In the latter case, the incompatibility should be resolved without requiring any operator intervention in order to support the dynamic nature of the D814 network.

The power-up and NP restart are not processed by the MSB module. Instead, they are handled by the Master Controller which causes the Mainframe System Initialization (MSI) module to start up the local node with a special configuration such that the node is disconnected from the network.

The MSB module must handle the following three functions:

1. Boot Request Recognition
2. Boot Synchronization and Arbitration
3. Node Restart

The network may assume any topology including the one consisting of several disconnected subnetworks which is caused by failures in the network or by operator commands.

For a connected network, all nodes in it must be running under the same software and the same configuration indicated by the active software level and active configuration number respectively.

5.5.2 Boot Requests

In order to start a boot, a boot request must be made and entered into the MSB module. A boot request may come from an operator through an I/CTP or Mainframe Panel, or it may be generated automatically when a link comes up or goes down.

5.5.2.1 Operator Boot Commands

The boot commands from an I/CTP and a Mainframe Panel are described in Sections 5.2.2 and 5.1, respectively, of the D814 Product Functional Specification. Their program interfaces are presented below in detail.

1. Boot Command from a Local I/CTP

When a boot command is entered at a local I/CTP, the I/CTP module must send an Addressed Packet of the following format to the local node's MSB$MAIN:START batch task whose module number EQ$BATCH:MSB is to be written in the packet header as the destination module.

Addressed Packet Format

Following the standard Addressed Packet header, the Command Code of EQ$MSB:CC_OP is written in the first byte of the text block. Then the boot request parameters follow.

The second byte contains the boot source code which, in this case, indicates an operator-initiated boot.

The third byte contains the configuration number for the current boot request at bits 3 - 0, and forced/optional flag bit for reconfiguration (Fc) at bit 7 (set if forced). The configuration number must be verified to be legal, i.e., to be within the range of 1 - 6, by the I/CTP.

The fourth byte has the Software Revision number for the current boot request at bits 6 - 0, and forced/optional flag bit for software reloading (Fs) at bit 7 (set if forced). The Software Release Level number is in the fifth byte.

The sixth and seventh bytes contain the node number and port number, respectively, of an I/FDP for the source of new software if a software reloading is needed.

The text block of the Addressed Packet is depicted below:

EQ$MSB:CC_OP				
EQ$MSB:SOURCE_OP				
Fc				Conf. Number
Fs	Software Rev. Number			
Software Release Number				
Software Source Node Number				
System Disk Port Number				

The boot request from the local I/CTP is accepted if it is not of lower priority than any other boot request currently being processed by the MSB module at the local node. Otherwise, the Addressed Packet (with the local node number appended) is returned to the I/CTP to indicate the rejection of the command.

The priority rule used to arbitrate multiple boot requests in a network is specified later in Section 5.5.3.2.

2. Boot Command from the Local Mainframe Panel

The Mainframe Panel Control (MPC) module must call the following entry point if a legal boot command is entered through the Mainframe Panel.

Entry Point - MSB$MPC:BOOT

Entry Conditions

* A-reg = Configuration number

Exit Conditions

* A-reg = Destroyed
* B-reg = Destroyed
* X-reg = Destroyed

The boot request is accepted if it is not of lower priority than any other boot request currently being processed at the local node. In this case, the control is returned to the caller after a short time which is needed only to generate a boot request Addressed Packet. Otherwise, this routine displays a "BOOT n REJECTED" message on the Self-Scan.

For the priority rule, see Section 5.5.3.2.

5.5.2.2 Automatic Boot Request from the Local Node

There are two entry points provided in the MSB module to be called from the Mainframe Network Link (MNL) module. The first is called when a local link comes up, and the second is called when a local link goes down. These are listed below:

Entry Point - MSB$MNL:LINKUP

Entry Conditions

* A-reg = Remote configuration number
* B-reg = I/NP port number

Exit Conditions

* A-reg = Destroyed
* B-reg = Destroyed
* X-reg = Destroyed

The function of this routine depends whether any boot process is already being served by the local MSB module.

When a boot process is not in progress:

If the remote configuration is the same as the local configuration, the routine just marks the link to be up and returns.

If the configurations are different, the routine updates the link status, generates an automatic boot request A.P. (for internal use in MSB module), and then returns.

When a boot process is already in progress:

The routine updates the current boot process to include the link and returns.

Entry Point - MSB$MNL:LINKDOWN

Entry Conditions

* B-reg = I/NP port number

Exit Conditions

* A-reg = Destroyed
* B-reg = Destroyed
* X-reg = Destroyed

When a boot process is not in progress, the routine just marks the link to be down and returns. Otherwise, it updates the current boot process to exclude the link and then returns.

5.5.2.3 Boot Request from a Remote Node

Any of the boot requests described so far may have been initiated by the MSB module at a remote node and propagated to the local node's MSB module as a part of the synchronization process as described in the following section.

5.5.3 Boot Synchronization and Arbitration

Once a boot procedure is started, it has to be propagated over the entire network to make the MSB modules at all the nodes synchronized before any one can initiate a node restart.

When multiple boot requests must be handled by an MSB module, they have to be arbitrated so that the MSB modules at all nodes in the network agree on the final boot request when the synchronization is complete.

These are described below in more detail.

5.5.3.1 Boot Synchronization

The synchronization procedure used by the MSB module is a variation of the Resynch Procedure developed at Codex (see "Resynch Procedures and a Fail-Safe Network Protocol", Steve Finn, ICC Proceedings, 1979). The detailed MSB algorithm is presented in Appendix G of the D814 Product Functional Specification.

The synchronization is started when the MSB module at a node accepts a boot request and broadcasts resynch messages to the MSB modules at all neighboring nodes.

When the MSB module at a node receives such a resynch message, it updates the local information regarding the boot synchronization and further propagates it to the MSB modules at neighboring nodes in the form of resynch messages.

If the boot information kept at the local node indicates that the MSB modules at all of the nodes have been resynched, the synchronization is complete as far as the local node's MSB module is concerned. The resynch procedure quarantees in this case that the MSB modules at the neighboring nodes will also be synchronized immediately after receiving the last resynch message from the local node's MSB module if they are not already synchronized.

When the synchronization is complete over the network, the MSB modules at all the nodes in the connected network will have the same final boot request, i.e., the boot request with the same software level and the same configuration number.

If the synchronization was started by an operator, the final boot request is the winning operator-initiated boot request. If it was automatically started, the final boot request is the optional reconfiguration with the majority configuration of the network. See the next section for the definition of "optional" boot request.

5.5.3.2 Arbitration of Multiple Boot Requests

When multiple boot requests are entered into a network within a short time span, contentions will arise for the MSB modules at some nodes between the boot request updated so far at the local node and a newly arrived boot request. In such cases the following priority rule is applied to resolve the contention.

1. A "forced" boot wins over an "optional boot."

   "Forced" boot means that a node restart will be scheduled at a node even if the node already has the same software level and the same configuration number active as the boot request demands. With an "optional" boot, such a case will not result in a node restart.

2. An operator-initiated boot wins over an automatic boot.

3. For an operator-initiated boot:

   a) The higher level software along with the associated source node/port wins. If they have the same software level, the lower-numbered source node/port wins.

   b) The lower-numbered configuration wins.

5.5.4 Node Restart

When the MSB module at a node completes the synchronization (and possibly arbitration) step, it schedules a node restart under the following conditions:

1. with the software level from the source node/port as indicated by the final boot request,

   a) if the software reloading is forced,

   b) or if it is different from the active software level;

2. and with the configuration number of the final boot request,

   a) if the reconfiguration is forced,

   b) or if it is different from the active configuration number

When a node restart is indeed to be done, the scheduling process is started which depends on what kind of boot originated the network synchronization.

If it was due to an automatic boot request, the MSB module sends a message to all local I/CTP's and schedules a node restart to be triggered in one minute. The message to the I/CTP's will inform operators that node restarts will be triggered in one minute over the entire connected network for the selected majority configuration number. It will also provide the operators with the information about the source of the boot.

The delay period is facilitated to provide any operator the opportunity of overriding the pending network restart with an operator-initiated boot request. At the end of the delay period, the node restart is cancelled if another boot has gone into effect in the meantime.

If the completed network synchronization was due to an operator-initiated boot request, a node restart is triggered after allowing a one-second settling period.

5.5.5 Examples of Boot Process

Since the distributed MSB algorithm is rather complex, the following two examples, one for an operator-initiated boot and the other for an automatic boot, are provided to help readers understand the boot process. In order to depict the process easily, a very simple network was chosen where there are only 3 nodes and 2 links. There are also assumptions made regarding timing. However, they do not interfere with following the essence of the algorithm.

For the exact algorithm, Appendix G of the D814 Product Functional Specification should be consulted.

The following is the list of state variables and messages used in the MSB algorithm.

M	= Node Mode
R	= Resynch Count
B	= Boot Request Table
L	= Link Table (Link #1 [, Link #2])
N	= Node Table (Node #1, Node #2, Node #3)
CONF	= Configuration Counter Vector (conf C1, conf C2) This is actually a 4-element vector; but for simplicity, only 2 elements are used in the automatic boot process description.
RM1	= Resynch Message of Type 1 (R, boot source)
RM2	= Resynch Message of Type 2 (origin node, origin B, (list of not yet resynched, but connected neighboring nodes))

1. An Operator-Initiated Boot Process

```
+--------------------------------------------------------------------------+
|                                                                          |
|                      L12                      L23                        |
|          N1 ----------+---------- N2 ----------+---------- N3            |
|                                                                          |
+=======================+========================+=========================+
|   M = 0               |   M = 0                |   M = 0                 |
|   R = 0               |   R = 0                |   R = 0                 |
|   B                   |   B                    |   B                     |
|   L = (2)             |   L = (2, 2)           |   L = (2)               |
+-----------------------+------------------------+-------------------------+
|   B' input            |                        |                         |
|   (B' > B)            |                        |                         |
+-----------------------+------------------------+-------------------------+
|   M <- 1              |                        |                         |
|   R <-1               |                        |                         |
|   B <- B'             |                        |                         |
|   N <- (1, 0, 0)      |                        |                         |
|                       |                        |                         |
|                 RM1 (1, OP)                    |                         |
|   *-------------------+--->                    |                         |
|                       |   M <- 1               |                         |
|                       |   R <- 1               |                         |
|                       |   L <- (2, 1)          |                         |
|                       |   N <- (0, 1, 0)       |                         |
|                       |                        |                         |
|                 RM1 (1, OP)                    |                         |
|   <-------------------+---*                    |                         |
|   L <- (2)            |                  RM1 (1, OP)                     |
|   N <- (2, 1, 0)      |   *--------------------+--->                     |
|                       |                        |   M <- 1                |
|   ---*                |                        |   R <- 1                |
|   |                   |                        |   L <- (2)              |
|   |                   |                        |   N <- (0, 0, 1)        |
|   |                   |                        |                         |
|   |                   |                  RM1 (1, OP)                     |
|   |                   |   <--------------------+---*                     |
|   |                   |   L <- (2, 2)          |   N <- (0, 1, 2)        |
|   |                   |   N <- (1, 2, 1)       |                         |
|   |                   |                        |                         |
|   |         RM2 (N2, B, (N1, N3))              |                         |
|   |  <----------------+---*                    |                         |
|   |  N <- (2, 2, 1)   |                        |                         |
|   |                   |   ---*                 |                         |
|   |                   |   |                    |                         |
|   V                   |   V                    |                         |
|   |                   |   |          RM2 (N3, B, (N2))                  |
|   |                   |   |   <----------------+---*                    |
|   |                   |   |   N <- (1, 2, 2)   |                        |
|   |                   |   |                    |                        |
|   |                   |   |          RM2 (N2, B, (N1, N3))              |
|   |                   |   ---------------------+--->                    |
|   |                   |                        |   N <- (1, 2, 2)       |
|   |         RM2 (N3, B, ())                    |                        |
|   |   <---------------+---*                    |                        |
|   |   N <- (2, 2, 2)  |                        |                        |
|   |                   |                        |                        |
|   |        RM2 (N1, B', (N2))                  |                        |
|   --------------------+--->                    |                        |
|                       |   B <- B'              |                        |
|                       |   N <- (2, 2, 2)       |                        |
|   R <- 0              |                        |                        |
|                       |             RM2 (N1, B', ())                     |
|                       |   *--------------------+--->                    |
|                       |   M <- 2               |   B <- B'              |
|                       |   R <- 0               |   N <- (2, 2, 2)       |
|                       |                        |                        |
|                       |                        |   M <- 2               |
|                       |                        |   R <- 0               |
|-------------------------------------------------------------------------|
|                                                                         |
|   Synchronization is complete.                                          |
|   Delay 1 second before triggering node restart.                        |
|-------------------------------------------------------------------------|
```

2. An Automatic Boot Process

```
===========================================================================
|                                                                         |
|                     L12                      L23                        |
|         N1 ----------+---------- N2 ----------+---------- N3            |
|      (conf C1)               (conf C2)                (conf C2)         |
|                                                                         |
===========================================================================
|                       |                        |                        |
|  M = 0                |  M = 0                 |  M = 0                 |
|  R = 0                |  R = 0                 |  R = 0                 |
|  B = B1 = (AUTO, C1)  |  B = B2 = (AUTO, C2)   |  B = B3 = (AUTO, C2)   |
|  L = (0)              |  L = (0, 2)            |  L = (2)               |
|                       |                        |                        |
---------------------------------------------------------------------------
|                       |                        |                        |
|  L12 comes up.        |  L12 comes up.         |                        |
|                       |                        |                        |
---------------------------------------------------------------------------
|                       |                        |                        |
|  M <- 1               |  M <- 1                |                        |
|  R <- 1               |  R <- 1                |                        |
|  L <- (1)             |  L <- (1, 1)           |                        |
|  CONF <- (0, 0)       |  CONF <- (0, 0)        |                        |
|  N <- (1, 0, 0)       |  N <- (1, 0, 0)        |                        |
|                       |                        |                        |
|  ---*                 |                        |                        |
|  |                    |                        |                        |
|  |              RM1 (1, AUTO)                  |                        |
|  |   <----------------+---*                    |                        |
|  |   L <- (2)         |                        |                        |
|  |   CONF <- (1, 0)   |                  RM1 (1, AUTO)                  |
|  |   N <- (2, 1, 0)   |  *---------------------+--->                    |
|  |                    |                        |  M <- 1                |
|  |                    |                        |  R <- 1                |
|  |              RM1 (1, AUTO)                  |  L <- (2)              |
|  ---------------------+--->                    |  CONF <- (0, 0)        |
|                       |  L <- (2, 1)           |  N <- (0, 0, 1)        |
|  ---*                 |                        |                        |
|  |                    |                  RM1 (1, AUTO)                  |
|  |                    |  <---------------------+---*                    |
|  |                    |  L <- (2, 2)           |  CONF <- (0, 1)        |
|  |                    |  CONF <- (0, 1)        |  N <- (0, 1, 2)        |
|  |                    |  N <- (1, 2, 1)        |                        |
|  |                    |                        |                        |
|  |         RM2 (N2, B2, (N1, N3))              |                        |
|  |   <----------------+---*                    |                        |
|  |   CONF <- (1, 1)   |                        |                        |
|  |   N <- (2, 2, 1)   |  ---*                  |                        |
|  |                    |  |                     |                        |
|  |                    |  |          RM2 (N3, B3, (N2))                  |
|  |                    |  |   <-----------------+---*                    |
|  V                    |  V                     |                        |
|   |                   |   |  CONF <- (0, 2)    |                        |
|   |                   |   |  N <- (1, 2, 2)    |                        |
|   |                   |   |                    |                        |
|   |                   |   |                    |                        |
|   |                   |   |         RM2 (N2, B2, (N1, N3))              |
|   |                   |   ---------------------+--->                    |
|   |                   |                        |   CONF <- (0, 2)       |
|   |        RM2 (N3, B3, ())                    |   N <- (1, 2, 2)       |
|   |  <---------------+---*                    |                        |
|   |  CONF <- (1, 2)   |                        |                        |
|   |  N <- (2, 2, 2)   |                        |                        |
|   |                   |                        |                        |
|   |                   |                        |                        |
|   |        RM2 (N1, B1, (N2))                  |                        |
|   --------------------+--->                    |                        |
|                       |   CONF <- (1, 2)       |                        |
|                       |   N <- (2, 2, 2)       |                        |
|   M <- 2              |                        |                        |
|   R <- 0              |             RM2 (N1, B1, ())                     |
|                       |   *--------------------+--->                    |
|   B <- (AUTO, C2)     |   M <- 2               |   CONF <- (1, 2)       |
|                       |   R <- 0               |   B <- (2, 2, 2)       |
|                       |                        |                        |
|                       |   B <- (AUTO, C2)      |   M <- 2               |
|                       |                        |   R <- 0               |
|                       |                        |                        |
|                       |                        |   B <- (AUTO, C2)      |
|                       |                        |                        |
|-------------------------------------------------------------------------|
|                                                                         |
|   Synchronization is complete.                                          |
|   Send a message to local I/CTP's.                                      |
|   Delay 1 minute.                                                       |
|                                                                         |
|-------------------------------------------------------------------------|
|                       |                        |                        |
|   Trigger             |   No                   |   No                   |
|   node restart        |   node restart.        |   node restart.        |
|   with conf C2.       |                        |                        |
|                       |                        |                        |
|-------------------------------------------------------------------------|
```

5.6 Mainframe Path Management, Routing, and Congestion Control Module (MPMRCCM)

5.6.1 Overview

The MPMRCCM has complete responsibility for control of data paths used by customer data and by addressed packets. The MPMRCCM creates, deletes, routes, and reroutes paths for customer data. It also provides the Mainframe Addressed Packet Module with the route to be used by each addressed packet. The MPMRCCM has significant interfaces with the I/P Call Management Module (I/P CMM), the I/P Protocol Module, the Mainframe Network Link Module (MNL), the Mainframe Addressed Packet Module (MAP), and the Intelligent Network Port (I/NP).

The MPMRCCM is divided into three functional submodule groups. These are:

Mainframe Path Manager (MPM) - The MPM handles establishment and deletion of paths for user data. Paths are established in response to messages from the I/P CMM (initial path establishment) and messages from the Mainframe Congestion Controller submodule group (rerouting of existing paths). The first sort of path is referred to as a "primary path" while the second sort of path is referred to as a "secondary path". Paths are deleted in response to link failure messages from the Network Link Control Module as well as in response to messages from the I/P CMM. A path may also be deleted after successful rerouting. But it should be noted that a primary path being rerouted is never deleted until a secondary path has been created to replace it.

Mainframe Routing Manager (MRM) - The MRM is responsible for computing and providing to other system components routing information which is mutually consistent throughout the connected network. The system components using this routing information are the Mainframe Addressed Packet Module and the MPM. Raw data for routing table computation is gathered in a network-wide resynch including all MRM's in the connected network (see Appendix to the D814 Product Functional Specification). Resynchs are done as requested by MCC (see below) and MNL.

Mainframe Congestion Controller (MCC) - The MCC is responsible for controlling user data traffic congestion by selecting paths as candidates for rerouting. MPM does the actual rerouting when so instructed by MCC. The information used in the path selection algorithm is gathered from the MNL and from the MRM.

5.6.2 External Interfaces

This section describes all interfaces between MPMRCCM components and other modules.

5.6.2.1 Interface with MNL

MNL must communicate with the MPMRCCM whenever a path is created or deleted. In addition MNL gathers link information which is used by all three components of MPMRCCM. MNL communicates with MPMRCCM by the following means:

Port Control Block (PCB) - Each Network Port, each Terminal Port, and each path through an intermediate node has a PCB. (See Mainframe Subsystem Data Structures.) All three MPMRCCM groups read data maintained by MNL in the PCB's. MPM writes data in the PCB Path Substructures of TP (Terminal Port), VP (Virtual Terminal Port), and XP (Transfer Port -- associated with a path at an intermediate node) PCB's. The Path Substructure of these PCB's is set up by MPM in the initial stages of path creation. When a path enters the Active path state (see path state machine diagram), the PCB is passed to MNL. In general MPM may not modify a PCB whose path state is Active and MNL may not modify a PCB in any other path state.

Messages - In addition, various messages flow between the MNL and the MPMRCCM. These messages are:

LINKFAIL - Informs MPM that a link has failed. The message is sent by calling routine MPM$UTIL:LINKFAIL with the ID of the NP in B register.

MPMLINKFAILACK - Acknowledges the above. It is sent by calling routine MNL$RECVRY:MPMFAILACK.

LUP - Informs MRM that a link has been brought up. The message is sent by calling routine MRM$UPDATE:LINKUP with the NP ID in the A register.

LDOWN - Informs MRM that a link has gone down. This message is sent by calling routine MRM$UPDATE:LINKDOWN with the ID of the failed NP in A register.

MRMLINKFAILACK - Acknowledges the above. It is sent by calling routine MNL$RECVRY:MRMFAILACK.

ADDSLOT - Sent by MPM to tell MNL that a path has become active and that data for it should be handled by MNL. The message is sent by calling routine MNL$UTIL:ADDSLOT.

KILLSLOT - Sent by MPM to tell MNL to end active data traffic so that the path may be deleted. The message is sent by calling routine MNL$UTIL:KILLSLOTSWITCH or MNL$UTIL:KILLSLOTFAIL.

SLOTKILLED - Sent by MNL to MPM to signal the termination of active data transmission over a path. It is sent by placing a message byte file directly on the MPM batch queue. (See MNL for message format.)

5.6.2.2 Interface with I/P CMM

The I/P CMM communicates with the MPMRCCM on the following occasions:

- Initial establishment of each primary or secondary path between a source and destination
- Termination of any path when no secondary path has been created to replace it
- Initiation of a call
- Termination of a call

Among the subgroups of MPMRCCM, only the MPM communicates with the I/P CMM. All such communication is by means of addressed packets. The message parameters are described in detail in the Appendix to the D814 Product Functional Specification. The messages are summarized here:

ACTCALL - Sent by I/P CMM to tell MPM to activate a call.

ACTCALLACK - Sent by MPM to tell I/P CMM that the call has been activated, meaning the ITP's PCB has been initialized for the call.

CALLCLRD - Sent by MPM to tell I/P CMM that a call has been ended or that the call specified in an ACTCALL could not be activated.

ESTXMTPATH - Sent by I/P CMM to tell MPM to establish a path for an active call. Establishment of each path of a call is started by an ESTXMTPATH received by the source MPM.

XMTPATHACT - Sent by MPM to I/P CMM at the source of a path to tell I/P CMM that the path has been activated.

RCVPATHACT - Sent by MPM to I/P CMM at the destination of a path to tell I/P CMM that the path has been activated.

XMTPATHERR - Sent by MPM to I/P CMM at the source of a path to tell I/P CMM that the path has failed or could not be established.

CLRCALL - Sent by I/P CMM to MPM to start terminating a call.

5.6.2.3 Interface with I/P Protocol Module

MPMRCCM at the destination node of an active path informs the I/P Protocol Module if and when the path is deleted. It does this by inserting an ICSKILLFAIL ICS sequence into the data stream through the outbound BIC data FIFO. It should be noted that MPMRCCM (actually the MPM subgroup) only does this after MNL has informed MPMRCCM that the path is no longer active. This way it is impossible for both MNL and MPMRCCM to be using the same FIFO concurrently.

5.6.2.4 Interface with MAP

MAP consults MPMRCCM whenever it must route an addressed packet for a remote node. The MRM subroutine MRM$ROUTE:PACKET is called with the remote node ID as argument. The subroutine consults MRM's routing table and returns the proper network port to use or an error indication if none exists. The calling sequence for MRM$ROUTE is as follows:

```
    On Entry:

    *    A-reg = Remote node to route to

    On Exit:

    *    X-reg = Destroyed
    *    A-reg = Adjacent node packet is to be sent to
    *    B-reg = I/NP to be used
    *    CC:Z  = Set if and only if node is unreachable
    *    CC:C  = Set if and only if route cannot be returned due to routing
                 table update in progress
    *    Data Space = Routing buffer (OF$MRM:DS_ROUTEBUF) is wiped out
```

5.6.2.5 Interface with I/NP

The I/NP sends MPMRCCM a STATISTICS addressed packet every 30 seconds. This packet provides MCC with all the information it needs to compute the capacity of the link. (See section on the I/NP for the packet format.)

5.6.2.6 Operational Overview of Submodule Groups

This section presents an operational overview of each of the submodule groups making up the MPMRCCM. Where the detailed algorithm is included in the Appendix to the D814 Product Functional Specification, reference will be made to it.

5.6.2.7 MPM Operational Overview

As noted above, the MPM handles creation and deletion of paths for user data. MPM is a batch task with entry point MPM$MAIN:ENTRY. Messages placed on MPM's batch queue may be control frames or addressed packets, or they may be placed directly on the queue by a MPM utility provided for that purpose. The various messages which flow between MPM and other MPMRCCM components as well as external modules are described in the MPM Design Specification.

The operation of MPM may best be understood by considering the major states associated with a path. These states are the Call State and the Path State.

5.6.2.7.1 Call State

MPM maintains a call state at each terminal port and virtual terminal port. There are three allowed call states:

IDLE - This port is not involved in a call and is available for initiation of a call. The call state is initially IDLE. Whenever it is reset to IDLE from some other state, MPM notifies I/P CMM by means of the CALLCLRD message.

ACTIVE - This port is involved in a call. If paths both ways are not currently in existence, they may be established. The call state becomes active when an ACTCALL is received from I/P CMM and positively acknowledged by MPM with an ACTCALLACK.

DISCONNECTING (DISC) - A call involving this port is in the process of disconnection. When both paths are deleted, it will inform I/P CMM and enter the IDLE state. The call state becomes DISCONNECTING whenever a CLRCALL is received from I/P CMM.

The next subsection includes two diagrams showing the process of establishment and deletion of a typical call.

5.6.2.7.2 Path State

As a transmit path is built or deleted, it passes through a succession of path states at each node along the path. The following is a rough description of the path states.

NOPATH - No path currently exists.

EST (Established) - All resources needed for the path have been allocated at this node and an attempt has been made to continue the path to the next downstream node.

INACT (Inactive) - This node has been notified that the path has been established all the way to the destination node. An attempt has been made to so inform the next upstream node (if one exists).

ACT (Active) - Data flowing on this path is included by MNL in the link traffic. The path state may become ACT only after the path state at each node in the path has gone successively from NOPATH to EST and from EST to INACT.

KILLED - Creation of this path cannot proceed beyond the Established State. A path becomes KILLED when a path error occurs when the state is Established. MPM must then wait for a MARK message before it can reset the path state to NOPATH and free up its resources. This ensures that no path can be created and deleted without at least one MPM message traversing the entire path, from start to finish, in each direction. (Note "finish" here is taken to mean the last connected link in the path to cover the case of a path with a link failure.)

Every path has a path state at each node along the path except the destination node. At the destination there is no path state, although the path may be considered to have the state "path" if it exists and "nopath" if there is no path to the node.

Path states are stored in the PCB Path Substructure (see section on Subsystem Data Structures). Each XP PCB contains the path state for the path (if any) with which it is associated. Each ITP or VP PCB contains the path state for the primary path (if any) and the secondary path (if any) originating at the associated port.

It is helpful to consider MPM as a path state machine. The following figure describes the states a path may have at any node and the causes of transitions between states. These causes are either messages received from neighboring MPM's or external modules or conditions which hold at the local node. For example, if the path state in a path substructure of a PCB in INACT, it will become ACT if an ACTPATH message is received. It will also become ACT automatically, with no external cause, if the local node is determined to be the destination node and if the path is a primary path.

PATH STATE MACHINE FOR MPM

CAUSE OF STATE TRANSITION	OLD STATE				
	NOPATH	EST	INACT	ACT	KILLED
ESTXMTPATH msg from CMM (if at source node of path)	EST				
ESTPATH msg from neighboring MPM (if not at source)	EST				EST**
Request Reroute from MCC or REROUTE from neighboring MPM (if source of sec path)	EST				
LINKFAIL (incoming link) *		KILLED	NOPATH	NOPATH	
CLRCALL from CMM		KILLED	NOPATH	NOPATH	
LINKFAIL (outgoing link) *		NOPATH	NOPATH		NOPATH
MARK (no error) received from MPM		INACT			NOPATH
MARK (error) received from MPM		NOPATH			NOPATH
Local node found to be dest. of path		INACT			
ACTPATH from downstream MPM			ACT		
This node found to be dest. of primary path			ACT		
USKILL received from MPM or LINKFAIL from MNL			NOPATH		
DSKILL received from MPM *			NOPATH	NOPATH	
SLOTKILLED from MNL				NOPATH	

The entry in the table for any given state and cause of state transition is the resultant path state.

* Transition from ACT to NOPATH state occurs after KILLSLOT is sent to MNL and SLOTKILLED response has been received.

** Fixed path only.

The following figures show the process of path and call creation, path and call deletion, and path failure in some typical common situations. It should be remembered that in real life things may not be this simple. For example, a link may fail when a reroute is in progress or a call may be disconnected before it is active. To understand what happens in such cases, the Appendix to the D814 Product Functional Specification should be consulted.

CREATION OF PRIMARY PATHS FOR A CALL THROUGH ONE INTERMEDIATE NODE

Path states are in parentheses.
Call states are in brackets.

```
   Source       Source          Int Node           Dest           Dest
    CMM          MPM              MPM               MPM            CMM
 ---------------------------------------------------------------------------
          |                |                |               |
 ACTCALL  +--->[ACT]       |                |               |
   <------+ ACTCALLACK     |                |               |
          |                |                |               |
                       - - - CALL IS NOW ACTIVE - - -
          |                |                |               |
 ESTXMTPATH--->(EST)       |                |               |
          | ESTPATH     ---+--->(EST)       |               |
          |                | ESTPATH     ---+--->(EST)      |
          |                |                |    (INACT) *  |
          |                |    (INACT) <---+--- MARK       |
          |    (INACT) <---+--- MARK        |    (ACT) *    |
          |                |                | RCVPATHACT ---+--->
          |                |    (ACT)   <---+--- ACTPATH    |
          |    (ACT)   <---+--- ACTPATH     |               |
   <------+--- XMTPATHACT  |                |               |
          |                |                |               |
               - - - XMT PATH FROM CALLER IS NOW ACTIVE - - -
          |                |                |               |
          |                |                |    (EST)  <---+--- ESTXMTPATH
          |                |    (EST)   <---+--- ESTPATH    |
          |    (EST)   <---+--- ESTPATH     |               |
          |    (INACT) *   |                |               |
          |    MARK     ---+--->(INACT)     |               |
          |    (ACT) *     |    MARK     ---+--->(INACT)    |
   <------+--- RCVPATHACT  |                |               |
          | ACTPATH     ---+--->(ACT)       |               |
          |                |    ACTPATH  ---+--->(ACT)      |
          |                |                | XMTPATHACT ---+--->
          |                |                |               |
                   - - - BOTH PATHS NOW OPERATIONAL - - -
          |                |                |               |
```

*   Path state for destination port PCB goes from EST to INACT and to ACT automatically, without waiting for any further messages.

DELETION OF PRIMARY PATHS FOR A CALL THROUGH ONE INTERMEDIATE NODE

Both path states are initially Active.
Transmit path states for source-destination path are in parentheses.
Transmit path states for destination-source path are in brackets ([]).
Call states are in number signs (#).

```
   Source          Source           Int Node           Dest            Dest
    CMM             MPM               MPM              MPM         CMM/Protocol
--------------------------------------------------------------------------------
           |                   |                  |                 |
CLRCALL ---+---> #DISC#        |                  |                 |
           |     (NOPATH)      |                  |                 |
           | ICSKILLFAIL *     |---> (NOPATH)     |                 |
           |                   | ICSKILLFAIL * ---+---> (NOPATH)    |
           |                   |                  | ICSKILLFAIL     +--->
           | USKILL         ---+--->              |                 |
           |                   |     USKILL    ---+---> [NOPATH]    |
           |                   |     [NOPATH] <---+ ICSKILLFAIL *   |
           |      [NOPATH]  <--+--- ICSKILLFAIL * |     #DISC#      |
       <---+--- ICSKILLFAIL    |                  | CALLCLRD     ---+--->
           |     #IDLE#        |                  |     #IDLE#      |
       <---+--- CALLCLRD       |                  |                 |
           |                   |                  |                 |
                     - - - CALL IS NOW TERMINATED - - -
```

* This ICSKILLFAIL is actually sent by MNL after receiving a KILLSLOT from the local MPM. It is received by the neighboring MNL which then informs MPM in that node by sending it a SLOTKILLED message. These details have been eliminated in the interest of simplicity.

FAILURE OF A PRIMARY PATH FROM SOURCE TO DESTINATION WITH NO INTERMEDIATE NODE

Path states are initially Active.
Path states are in parentheses.
Only one path of the call is shown.

```
                         Source Destination

  I/P       MPM              MNL             MNL              MPM       I/P
 ________________________________________________________________________
        |               |             |              |               |
        | (NOPATH)  <---+--- LINKFAIL | LINKFAIL ---+---> (NOPATH)   |
   <---+ XMTPATHFAIL    |             |              | ICSKILLFAIL   |--->
        |               |             |              |               |
                        - - - PATH IS DELETED - - -
                 - - - CMM MAY NOW TRY TO RE-ESTABLISH - - -
```

LINK FAILURE ON A PRIMARY PATH THROUGH TWO INTERMEDIATE NODES

Path state is initially Active.
Path states are in parentheses.

It is assumed that the failed link is between the intermediate nodes.

The first diagram shows what happens between the failed link and the path destination:

```
       Intermediate Node                   Destination Node
     MPM            MNL              MNL             MPM         I/P
---------------------------------------------------------------------
              |                |               |                |
          <---+--- LINKFAIL    |               |                |
  KILLSLOT ---+--->            |               |                |
 (NOPATH) <---+--- SLOTKILLED  |               |                |
              | ICSKILLFAIL ---+--->           |                |
              |                | SLOTKILLED ---+---> (NOPATH)   |
              |                |               | ICSKILLFAIL ---+--->
```

The next diagram shows what happens between the failed link and the source.

```
             Source Node                     Intermediate Node
 I/P        MPM             MNL             MPM             MNL
---------------------------------------------------------------------
      |               |                |              |
      |               |                |          <---+--- LINKFAIL
      |           <---+----------------+--- USKILL    |
      | KILLSLOTFAIL  +--->            |              |
      |  (NOPATH) <---+--- SLOTKILLED  |              |
  <---+--- XMTPATHERR | ICSKILLFAIL ---+--------------+--->
      |               |                | (NOPATH) <---+--- SLOTKILLED
      |               |                |              |
                   - - - PATH IS NOW DELETED - - -
```

5.6.2.7.3 MRM Operational Overview

The algorithm used by the MRM is completely explained in Appendix A to the Product Functional Specification and need not be described further here.

5.6.2.7.4 MCC Operational Overview

The MCC, as noted earlier, is responsible for rerouting of already-established paths for any reason other than a link failure in the path. The MCC monitors the state of congestion of each outgoing link and, as long as there exist congested links, triggers a congestion resynch by MRM every small time interval (of the order of 10 seconds). If there are no congested links at the node, it still triggers congestion resynchs, but at a much longer time interval (of the order of 2 minutes). The reason for triggering resynchs when there are no congested links is that a better path with sufficient available bandwidth for an already-established call can open up due to a change in network traffic patterns. For example, suppose a link between node A and node B becomes congested, causing a call from A to B to be rerouted through a longer path. Once the reroute is done, the previously congested link is no longer congested. Assume that later on traffic between A and B decreases so that there is now sufficient bandwidth to accommodate the call without re-introducing link congestion. At this point there may be no congested links in the network, but the call can still be profitably rerouted.

At the completion of any resynch, whether initiated by MCC or not, MCC at each node begins a series of reroute attempts. It examines all paths through the node and classifies each path according to its potential for succcessful rerouting. It then goes through a subset of the potentially reroutable paths and, for each path, tells MPM to attempt a reroute and waits for a completion message from MPM. If no completion message is received, the wait terminates when the next resynch completes.

Perhaps the most important consideration in the design of the MCC is system stability. The algorithm tries to inhibit oscillations and system overloading due to repeated unsuccessful reroutes of the same path.

The MCC is a batch task with entry point MCC$EXEC:ENTRY. Messages are placed on MCC's batch queue in two ways:

1. System components which reside in the local mainframe call subroutines supplied by MCC to place messages in addressed packet format directly on MCC's batch queue.

2. The I/NP sends addressed packet messages to MCC which are routed to the MCC batch queue.

It should be noted that MCC receives no messages from non-local modules.

This section will describe the interface between the MCC and the other two MPMRCCM submodule groups in some detail, since this interface is fairly complex and is not described elsewhere. Next the basic algorithm used by MCC in choosing paths as candidates for rerouting will be described.

5.6.2.7.4.1 MCC Port Control Block Interface

Port control blocks (described in detail in Section XYZ) provide a data interface among MCC, MNL, MRM, and MPM. This interface is complicated by the fact that a PCB may be dynamically allocated (currently only transfer port (XP) PCB's are allowed to be dynamic). Dynamic PCB creation and deletion is done by MPM via operating system utility subroutines MUT$PCB:ADDPORT and MUT$PCB:DELETE_DYNXP.

MCC uses PCB's to gather two sorts of information:

1. Network link throughput data from which link excess capacity may be computed. This information is taken from the PCB associated with the links I/NP.

2. Path throughput data for use in rerouter decision-making. This information is taken from the PCB associated with the path being considered. This PCB may be an Intelligent Terminal Port (I/TP), Virtual Port (VP), or Transfer Port (XP) PCB.

The following PCB fields in I/NP PCB's are used by MCC for Link statistics:

Link Speed (XMT_LNKSPD) - This is the capacity of the link in bytes/sec, initialized by MNL at link startup to a configuration-set parameter and updated by MCC from information received in the STATISTICS message. It is not a long-term average, but merely the Link speed as computed directly from the TOTAL_BYTES field in the previous STATISTICS message. It is a two-byte field interlocked through the PCB's STATIS lockbyte.

Link Traffic (XMT_TRAFFIC) - This is a long-term average of the outgoing user data rate through an I/NP in bytes/sec. It is also a two-byte field interlocked through the STATIS lockbyte.

Overhead (XMT_OHEAD) - This is a long-term average of the overhead bytes per second transmitted by the I/NP. It is updated by MCC from the STATISTICS message. It is read by both both MRM and MCC to compute excess capacity. It too is a two-byte field interlocked through the STATIS lockbyte.

Link Inactive Traffic (XMT_INACT_TRAFFIC) - This is the total throughput in bytes/sec of all paths established but not activated through the link. It is a two-byte field interlocked through the STATIS lockbyte and updated by the Mainframe Path Manager (MPM) module using the MCC utilities (MCC$UTIL). It is reset by MNL whenever a link comes up.

The following PCB fields in I/TP, UP, and XP PCB's are used by MCC to gather information from MNL and MPM about an active path.

Path State (PATH_STATE and PATH_PSTATE) - PCB field PATH_PSTATE in I/TP and VP PCB's and PCB field PATH_STATE in XP PCB's are read by MCC to decide if the PCB is associated with an active primary path.

Destination node (PATH_DSTND) - Destination node of path with which this PCB is associated.

Estimated Statistical Speed (SLOT_ESSPD) - This is the estimated long-term speed of the path. It is read by MCC from the PCB slot data substructure to decide if a path can be rerouted without introducing new network congestion.

Path Length (PATH_HOPS and PATH_PHOPS) - Field PATH_PHOPS in I/TP and VP PCB's and field PATH_HOPS in XP PCB's express the length of the path from the local node to its destination. It is used by MCC to determine if a secondary path is shorter than the primary path.

Transmit Network Port (PATH_PXNP and PATH_XNP) - Field PATH_PXNP in I/TP and VP PCB's and field PATH_XNP in XP PCB's are used to determine the I/NP in the local node used for the outgoing path. It is used by MCC to determine if the link used for the primary path is congested.

The above fields, unlike those in the I/NP PCB, may be located in a dynamically allocated PCB. To ensure that the PCB is not deleted while the above fields are being read, the Path Data Substructure lockbyte OF$SYSLCK:PATH must be locked while accessing them.

5.6.2.7.4.2 MCC Routing Table Interface

MCC calls MRM$ROUTE_SEC to get routing information used in determining if an attempt should be made to reroute a path.

5.6.2.7.4.3 MCC Message Interfaces

The following Addressed Packet messages are received by MCC:

Resynch Completed - Sent by MPM at the completion of a resynch. This message means that network congestion and/or topological information has just been updated. It causes MCC to initiate the checking of all PCB's for possible rerouting.

Reroute Attempted - Sent by MPM to indicate that the attempt to reroute a path is complete.

Statistics - Sent every 30 seconds by the I/NP to provide MCC with the raw information needed to compute capacity of that link.

The following message is sent by MCC:

Request Reroute - Sent to MPM. Implies that the primary path associated with the PCB ID contained in the message is a candidate for rerouting and requests MPM to being rerouting. MCC commits itself to send no other such messages and to make no change in the PCB's reroute state until a Reroute Complete message is received from MPM.

Start Congestion Resynch - Sent to MRM to cause MRM to initiate a congestion resynch.

5.6.2.7.4.4 MCC Algorithm

Initiation of Resynch

As was seen in the Operational Overview Section, MCC must decide when to initiate a congestion resynch and must, at the end of a resynch (whether or not it was a congestion resynch), initiate a series of reroute attempts. This section describes how these two functions are performed.

Initiation of Resynchs

At the start of each series of reroute attempts, MCC sets a timer which at expiration will cause MCC to initiate a new congestion resynch. This timer is set to thirty seconds if there are any congested outgoing links at the local node and two minutes if not. This means that resynchs will occur roughly every thirty seconds as long as there are congested links in the network and that a resynch will almost always occur within approximately thirty seconds of the time any congested link becomes uncongested.

Doing a Series of Reroutes

After notification of completion of a resynch, whether initiated by MCC or not, MCC does a series of reroute attempts. Before starting any reroute attempts, MCC classifies all active paths. Paths are classified into one of four priority classes:

Priority 3 - A shorter path with sufficient excess capacity exists from this node to the path's destination.

Priority 2 - The outgoing link used by the path is congested, but there is a route longer than the current path with sufficient excess capacity from here to the destination of the path.

Priority 1 - The outgoing link of the path is congested, but the path is not priority 2 or 3.

Priority 0 - None of the above. Path is not a candidate for reroute.

At this point, MCC checks if a reroute is currently in progress from the prior series of reroute attempts If so, MCC waits for the reroute completed message from MPM before continuing with the current series of reroutes.

MCC then tries to reroute these paths, in priority order. Before an attempt is made, the path is checked to verify that its priority has not changed to a lower priority since the initial classification. If so, it is reclassified. Otherwise, MPM is sent a Reroute Request for the path and it is reclassified as priority 0. In order to avoid oscillation the paths are rerouted one at a time: MCC always waits for notification of completion of one attempt before requesting another.

5.6.2.7.4.5 Interface with MDL

The MPMRCCM maintains minimum depth spanning trees for use in broadcasting messages throughout the network. A minimum depth spanning tree rooted at node A is a set of links between adjacent nodes in the network such that:

1. Any node in the connected network may be reached by one and only one path using links from the spanning tree.

2. There is no path in the network from A to any other node in the network using fewer links that the one using the spanning tree.

Figure 1a shows a typical network and Figure 1b shows the same network with only those network links on a minimum depth spanning tree rooted at node 1.

```
6---------7---------10
|         |
|         |
|         |
2---------3         |
|         |         |
|         |         |
|         4         |
|         |         |
|         |         |
|         |         |
|         |         |
1---------5---------8----------9
```

Figure 1a

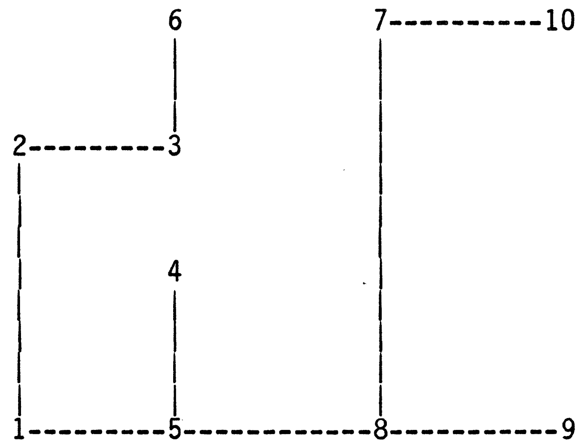


Figure 1b

In order to broadcast a message to all nodes in the network, the broadcasting node first sends it out over all links on the spanning tree which go outwards from itself. In other words, the broadcasting node first sends the message to each node adjacent to it. Then each node receiving the message from some adjacent node relays it to all other adjacent nodes on the spanning tree. In this way, barring any change in network topology, each node in the connected network receives one and only one copy of the message. If the network topology changes so that different spanning trees are used at different nodes, it is possible for many nodes either not to receive the message or to receive duplicates.

Spanning tree information for such a broadcast is obtained as follows:

Subroutine MRM$BROADCAST:ENTRY

On Entry:

A-reg =	Node from which messages are to be broadcast

On Exit:

CC:Z =	Set if and only if the spanning trees are unuseable because the network topology is in transition.
X-reg =	If CC:Z not set, points to a byte file containing a list of all adjacent nodes to which the message should be sent in order to reach the entire connected network.

Messages are broadcast on spanning trees by the Mainframe Downline Load Module (MDL).

ˌlainframe Configuration Manager Module (MCM)

The purpose of the Mainframe Configuration Management Module (MCM) is to .ow other modules in a D814 Network to access the Configuration informa- ,on located in the mainframe of the D814 node where the MCM resides. This ,nformation is stored online in the mainframe memory and offline in CMOS RAM's (CMEM).

This document describes the functions performed by the MCM as well as the command structure, the Addressed Packet Format, and the CMOS Rams (CMEM) used to store offline parameters.

The configuration information maintained at a D814 node can be broken down into two parts: node parameters and port parameters. Node parameters are those parameters common to the node. Port parameters are definitions of the terminal's and the port's properties.

The currently active port parameters are located at the port, and therefore, are maintained by the port modules. The node parameters are the only parameters that are maintained online by the MCM. Both node and port parameters are maintained offline by the MCM.

5.7.1 Hardware and Firmware

This section describes the hardware/firmware and I/O Communications interface required by the MCM.

1) Options Card

   MCM requires an options card on every D814 Mainframe. The MCM controls all system access to the CMOS RAM residing on the options card after system initialization. This CMEM stores off-line user configurations, and is expected to survive power and system failures. A battery is used to back up the low power CMOS RAM in case of power down.

2) Firmware

   Since the CMEM is not directly mapped in the 6800 address space the MCM must use the firmware to access it. Up to four configurations are mapped in the CMEM space depending on the maximum number of ports required per configuration as defined in the options PROM (see subsection on CMEM Map Table). Each port currently takes 24 bytes of dedicated CMEM space.

5.7.2 General Functional Description

The MCM performs two distinct functions. The first function is the maintenance of online node parameters. These parameters reside in the mainframe memory and specify information about the node as a whole, rather than information about any particular ports on the node. This function is performed by a batch task with entry point MCM$NODE:ENTRY.

The second function is the maintenance of offline (CMEM) configuration parameters. These include node and port information for all the different configurations. This function is performed by another batch task, with entry point MCM$CMEM:ENTRY.

Addressed Packets provide the only user interface with MCM. Each Addressed Packet contains a list of commands specifying operations to be performed on configuration data. If the Addressed Packet is sent to MCM$CMEM only the offline configuration parameters are modified. A node boot must occur before any changes made may take effect. If the Addressed Packet is sent to MCM$NODE the online node parameters are first modified, and the packet is then sent to MCM$CMEM to modify the corresponding offline parameters.

The online and offline parameter maintenance functions are described in the following subsections.

5.7.2.1 Online Node Parameter Maintenance (Submodule MCM$NODE)

Since the online node parameters are the only parameters accessible through this submodule, the configuration and port specified in packets received by this submodule must be 0. (See subsection on Addressed Packet format.) If they are not, an error code is placed in the packet error code and the packet is returned (via the MAP) to its source. The routine examines and executes each command in the sequence they appear in the packet.

If any errors occur, an appropriate error code is stored in the packet, the command in error is aborted, and the next command is processed.

If the command is a valid read, the field is obtained from the mainframe memory and stored in the packet for return to the source. If the command is a valid write, the field is updated.

If, at the end of the packet, any valid writes have been performed, the packet is rerouted (via the MAP) to the Offline Maintenance module of MCM for updating of the CMEM. Otherwise, the packet is returned (via the MAP) to its source.

5.7.2.2 Offline (CMEM) Parameter Maintenance (Submodule MCM$CMEM)

The configuration specified in packets received by this submodule may be either 0 or 1 through 4. If the configuration is zero, the online parameters have already been updated by the IP or Online Maintenance module and this module is to perform the corresponding offline update. The configuration and port are checked for an empty configuration, invalid configuration, or invalid port. If any of these errors occur, an appropriate error code is placed in the packet error code and the packet is sent back to its source. The routine then examines and executes each command in the sequence they appear in the packet.

If a command is detected that already has a non-zero error code other than "online change not allowed", the command is bypassed. This occurs if the online update detects an error. Also, if the configuration is zero, only valid writes contained in the packet are performed, since all reads were previously executed by the IP or Online Maintenance module and need not be duplicated.

If any errors occur, an appropriate error code is stored in the packet, the command in error is aborted, and the next command is processed.

If the command is a valid read, the field is obtained from offline memory and stored in the packet for return to the source. If the command is a valid write, the field is updated.

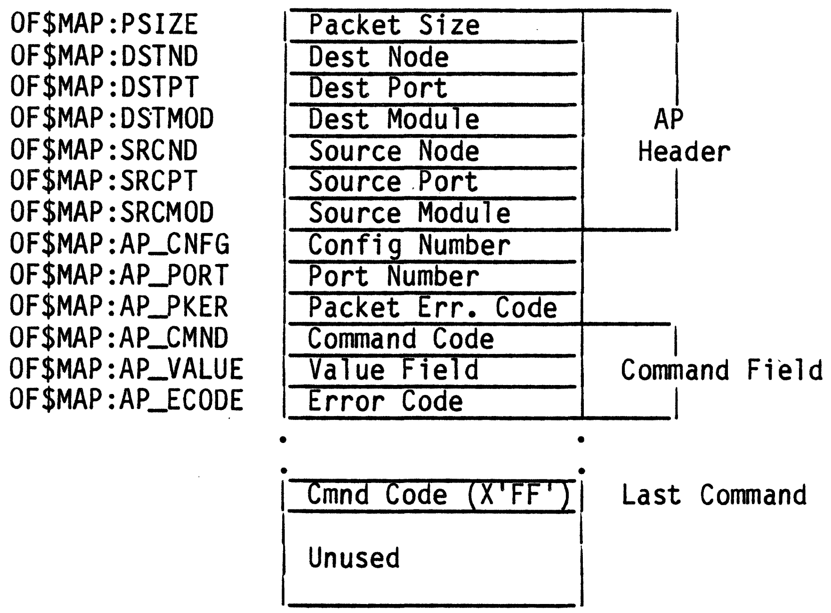
The command may also be a valid copy command, either configuration copy or port copy, in which case the corresponding operation is performed.

At the end of the packet, the packet is returned to the original source.

5.7.3 Addressed Packet Format

The configuration information is requested and returned in Addressed Packets (the same packet is used to return information).

One configuration and port can be referenced in a packet. Therefore, these fields occur only once in the packet. The Command Code, Value, and Error Code fields, i.e., the Command Field, can occur up to 81 times and are terminated by a special END command. Any information appearing in the packet after the END command is ignored, but is maintained intact. The fields in the packet are defined as follows:



The AP Header is described in the section on System Data Structures.

Config Number - 1 byte - configuration to be referenced:

0 =	current online configuration
1-4 =	offline configurations 1 through 4

Port Number - 1 byte - Port to be referenced:

0 =	node parameters
1 =	illegal
2-225 =	ports 2 through 255

Packet Error Code - 1 byte - This field contains zero when the packet is returned to the source if the configuration and port are valid, otherwise, none of the commands were processed and this field contains one of the following error codes:

EQ$MEM:EC_IVCNFG =	Invalid configuration
EQ$MCM:EC_IVPORT =	Invalid port
EQ$MCM:EC_NLCNFG =	Null configuration

Command Field - 3 bytes - Occuring up to 81 times, as follows:

Command Code - 1 byte - High order bit indicates reading or writing of field (0 = read, 1 = write). The remaining bits indicate a particular data item to be referenced (see Summary of Commands). For most, but not all, command codes the data item is a particular parameter in CMEM, either a port or a node parameter.

Value Field - 1 byte - If the Command Code indicates a read, when the packet is returned to its source this field will contain the value (right justified) retrieved from the referenced field. If the Command Code indicates a write, this field must contain the new value (right justified) to be placed in the referenced field.

Error Code - 1 byte - This field should be initialized to zero when the packet is created. It will contain zero when the packet is returned to the source if the operation was performed successfully, otherwise, it will contain an error code indicating why the operation could not be performed, as follows:

```
EQ$MCM:EC_ILLCMD = Illegal command
EQ$MCM:EC_INVCMD = Command invalid for config/port
EQ$MCM:EC_INVVAL = Invalid value
```

5.7.4 Offline Memory Format

The offline CMOS RAM (CMEM), as previously mentioned, contains node and port parameters. Up to 4 different configurations may be stored in this CMEM.

The next two subsections describe the formats for node and port parameters. For each field the following is given:

1) The name of the field
2) The size of the field
3) The command name to access the field
4) A description of the field

CMEM is divided into segments each consisting of 24 contiguous bytes. Parameters for each port use a single segment of CMEM.

5.7.4.1 Node Parameters

This information is maintained in the two segments which would normally be reserved for ports 0 and 1. Therefore, ports 0 and 1 cannot be defined in the configuration.

The following parameters are maintained in this entry:

CMEM Checksum - 2 bytes - EQ$MCM:OF_CHKSM - This field is maintained only in the first configuration. It contains the end-around carry checksum for CMEM. There is no command to read or write this field as it is maintained internal to MCM.

Currently Active Configuration - 1 byte - EQ$MCM:CC_ACNF - This field is maintained only in the first configuration. It is used on a reboot to determine what configuration was running when the system went down. Note: There is no command to read or write this field, as it should be known by all nodes and ports.

Report Node Number - 1 byte - EQ$MCM:CC_RPTN - This field contains the node to which all system report messages are to be sent.

Report Port Number - 1 byte - EQ$MCM:CC_RPTP - This field contains the port to which all system report messages are to be sent.

Routing Debug Flags - 1 byte - EQ$MCM:CC_RDBF - Bit flags to control the routing system trace option (see subsection on Mainframe Path Manager Module).

Buffer Utilization Threshold - 1 byte - EQ$MCM:CC_BUTH - Threshold value (in percent) of the number of buffers used versus the number of buffers allocated.

Processor Loading Threshold - 1 byte - EQ$MCM:CC_PLTH - Threshold mean value (in percent) of all processor loading.

Averaging Time Constant - 1 byte - EQ$MCM:CC_AVTC - Value of MSM's averaging time constant (see subsection on MSM).

Active Software Level Revision Number - 1 byte - EQ$MCM:CC_ASWLV_REV - Revision number of current software.

Active Software Level Release Number - 1 byte - EQ$MCM:CC_ASWLV_RELEASE - Release number of current software.

Active Software Source Node - 1 byte - EQ$MCM:CC_ASWSRCND - Node from which software is to be loaded.

Active Software Source Port - 1 byte - EQ$MCM:CC_ASWSRCPT - Port from which software is to be loaded.

5.7.4.2 Port Parameters

Port parameters are defined for each type of port supported by the D814 system. The first two fields always define the type of port and the subtype.

Generic Type - 4 bits - EQ$MCM:CC_GTYP - Defines the type of IP; i.e., I/NP, I/STP, I/ATP, etc.

Subtype - 4 bits - EQ$MCM:CC_STYP - Defines the subtype of the IP; i.e., spoofing I/STP, normal I/STP, etc.

The rest of the fields vary depending on the Generic Type, as follows:

Processor Loading Threshold - 7 bits - EQ$MCM:CC_PLIP - Threshold value (in percent) of processor loading danger level for the port.

Buffer Utilization Threshold - 7 bits - EQ$MCM:CC_BUIP - Threshold value (in percent) of buffer utilization danger level for the port.

Time Constant Factor - 7 bits - EQ$MCM:CC_TIME - A parameter controlling the statistics collection weighting.

5.7.4.2.1 Intelligent Network Port (I/NP)

Speed - 2 bytes - EQ$MCM:CC_SPDH & EQ$MCM:CC_SPDL - The line speed in binary. This field is broken into two 1-byte fields for updating and retrieval (SPDH being the high order byte and SPDL the low order byte).

Comm. Node - 2 bits - EQ$MCM:CC_MODE - Specifies whether the IP is operating in full duplex, local loopback, or remote loopback.

NRZI Mode - 1 bit - EQ$MCM:CC_NRZI - NRZ/NRZI coding specification.

User Density Threshold (USDN) - 8 bits (EQ$MCM:CC_USDN. Data rate (divided by 100) above which alarm is issued.

Non-Acknowledgement Timer (NOAK) - 7 bits (EQ$MCM:CC_NOAK). Time (in seconds) during which an I/NP will wait without receiving any ACKs from the remote (attached) I/NP before declaring itself dead.

5.7.4.2.2 Transfer Port (XP)

Recv. Adj. Node - 7 bits - EQ$MCM:CC_RADN - Contains the number of the node which will transmit to this port.

Recv. Adj. Port - 1 byte - EQ$MCM:CC_RADP - Contains the number of the port which will transmit to this port.

Xmit. Adj. Node - 7 bits - EQ$MCM:CC_XADN - Contains the number of the node to which this port will transmit.

Xmit. Adj. Port - 1 byte - EQ$MCM:CC_XADP - Contains the number of the port to which this port will transmit.

Xmit. Network Port - 1 byte - EQ$MCM:CC_XNTP - Contains the number of the port at this node used to transmit to the Xmit. Adj. Node/Port. If this field is zero, the 6050 will choose the NP to be used to transmit to the Xmit. Adj. Node/Port.

5.7.4.2.3 Intelligent Synchronous Terminal Port (I/STP)

Speed - 2 bytes - EQ$MCM:CC_SPDH & EQ$MCM:SPDL - (same as for I/NP).

Code Type - 1 byte - EQ$MCM:CC_CODE - Specifies the code type used by the terminal.

Data Bits - 2 bits - EQ$MCM:CC_DBTS - Specifies the number of bits contained in each character transmitted and received at the terminal (not including any parity bit). The valid values are:

0 = 5 bits
1 = 6 bits
2 = 7 bits
3 = 8 bits

Parity - 2 bits - EQ$MCM:CC_PRTY - Specifies whether even, odd, mark, or space parity is used by the terminal.

Delay - 1 byte - EQ$MCM:CC_DELY - Defines the amount of time to delay after receiving the first character of a block for the terminal before sending it.

Comm. Mode - 2 bits - EQ$MCM:CC_MODE - Same as for I/NP.

(OP) Mode - 1 byte - EQ$MCM:CC_OPMD - Defines special operating characteristics of the terminal port or modem.

Routing - 3 bits - EQ$MCM:CC_RTNG - Specifies whether dynamic, or fixed routing is to be used. Also specifies path priority.

Dest. Node - 7 bits - EQ$MCM:CC_DSTN - Defines the node that data from the terminal is destined for.

Dest. Port- 1 byte - EQ$MCM:CC_DSTP - Defines the port that data from the terminal is destined for.

Xmit. Adj. Node - 7 bits - EQ$MCM:CC_XADN - Same as for XP.

Xmit. Adj. Port - 1 byte - EQ$MCM:CC_XADP - Same as for XP.

Xmit. Path Priority - 3 bits - EQ$MCM:CC_XPTY - Specifies the network priority of this terminal (used to calculate the slot weight).

Xmit. Network Port - 1 byte - EQ$MCM:CC_XNTP - Same as for XP.

Security Level - 3 bits - EQ$MCM:CC_SECL - Priority of call for shortest path routing. 7 = high, 0 = low.

Call Type - 2 bits - EQ$MCM:CC_CALL - Method of call handling within the D814 network: Leased Line, Autodial, Dialup, contention.

RTS/CTS Delay - 1 byte - EQ$MCM:CC_CTSD - Delay for presenting CTS (in milliseconds) after detection of RTS.

Comp. Eff. Threshold - 1 byte - EQ$MCM:CC_CEIP - Threshold value (in percent) of the number of bits received from the user equipment to the number of bits sent over the network.

Error Density Threshold - 7 bits - EQ$MCM:CC_EDIP - Threshold value (in percent) of the number of characters received with bad parity to the total number of characters received.

5.7.4.2.4 Intelligent Asynchronous Terminal Port (I/ATP)

Code Type - 1 byte - EQ$MCM:CC_CODE - Same as for I/STP.

Speed - 2 bytes - EQ$MCM:CC_SPDH & EQ$MCM:SPDL - The line speed in binary, if external clocking is to be used. Otherwise, if internal clocking is to be used, the high order byte is set to X'FF' and the low order byte contains an encoded speed (X'00' to X'FF' being the 2651 on-chip baud rates). This field is broken into two 1-byte fields for updating and retrieval (SPDH being the high order byte and SPDL the low order byte).

Data Bits - 2 bits - EQ$MCM:CC_DBTS - Same as for I/STP.

Stop Bits - 2 bits - EQ$MCM:CC_STPB - Defines the number of stop bits for asynchronous transmission. The valid values are:

0 = 1 bit
1 = 1 bit
2 = 1.5 bits
3 = 2 bits

Parity - 2 bits - EQ$MCM:CC_PRTY - Same as for I/STP.

Auto Echo - 1 bit - EQ$MCM:CC_ECHO - Specifies whether chraracters received from the terminal are to be echoed back to the terminal.

Flyback - 1 byte - EQ$MCM:CC_FLYB - Defines the character to be searched for when transmitting to a mechanical printer. When this character is detected, PAD characters are transmitted for a defined period of time to allow the carriage to return. Typically, this field is set to the carriage return character. If this character is set to zero, no search occurs.

Garble Character - 1 byte - EQ$MCM:CC_GARB - Defines the hex character to be sent to the terminal when a character is received containing bad parity.

Comm. Mode - 2 bits - EQ$MCM:CC_MODE - Same as for I/NP.

(OP) Mode - 1 byte - EQ$MCM:CC_OPMD - Same as for I/STP.

Routing - 3 bits - EQ$MCM:CC_RTNG - Same as for I/STP.

Dest. Node - 7 bits - EQ$MCM:CC_DSTN - Same as for I/STP.

Dest. Port - 1 byte - EQ$MCM:CC_DSTP - Same as for I/STP.

Xmit. Adj. Node - 7 bits - EQ$MCM:CC_XADN - Same as for XP.

Xmit. Adj. Port - 1 byte - EQ$MCM:CC_XADP - Same as for XP.

Xmit. Path Priority - 3 bits - EQ$MCM:CC_XPTY - Same as for I/STP.

Xmit. Network Port - 1 byte - EQ$MCM:CC_XNTP - Same as for XP.

Error Density Threshold - 7 bits - EQ$MCM:CC_EDIP - Threshold value (in percent) of the number of characters received with bad parity, framing error, or overrun to the total number of characters received.

5.7.4.2.5 Intelligent Bit-Oriented-Protocol Port (I/BOP)

Speed - 2 bytes - EQ$MCM:CC_SPDH and EQ$MCM_CC_SPDL - Same as for I/NP.

Data Bits - 2 bits - EQ$MCM:CC_DBTS - Same as for I/STP.

Comm. Mode - 2 bits - EQ$MCM:CC_MODE - Same as for I/NP.

Routing - 3 bits - EQ$MCM:CC_RTNG - Same as for I/STP.

Dest. Node - 7 bits - EQ$MCM:CC_DSTN - Same as for I/STP.

Dest. Port - 1 byte - EQ$MCM:CC_DSTP - Same as for I/STP.

Xmt. Adj. Node - 7 bits - EQ$MCM:CC_XADN - Same as for XP.

Xmt. Adj. Port - 1 byte - EQ$MCM:CC_XADP - Same as for XP.

Xmt. Path Priority - 3 bits - EQ$MCM:CC_XPTY - Same as for I/STP.

Xmt. Network Port - 1 byte - EQ$MCM:CC_XNTP - Same as for I/NP.

Comp. Eff. Threshold - 1 byte - EQ$MCM:CC_CEIP - Same as for I/STP.

Error Density Threshold - 7 bits - EQ$MCM:CC_EDIP - Threshold value (in percent) of number of bad frames received versus total number of frames received.

Proc. Load. Threshold - 7 bits - EQ$MCM:CC_PLIP - Same as in port parameters.

Buff. Util. Threshold - 7 bits - EQ$MCM:CC_BUIP - Same as in port parameters.

Security Level - 3 bits - EQ$MCM:CC_SECL - Same as for I/STP.

Call Type - 2 bits - EQ$MCM:CC_CALL - Same as for I/STP.

RTS/CTS Delay - 1 byte - EQ$MCM:CC_CTSD - Same as for I/STP.

NRZ/NRZI Option - 1 bit - EQ$MCM:CC_NRZI - NRZ/NRZI coding.

Bad FCS Option - 1 bit - EQ$MCM:CC_BFCS - disposition of a frame received with bad FCS (abort or discard).

Address Field Ext. - 1 bit - EQ$MCM:CC_AEXT - Specifies whether address field of a frame may be extended.

Control Field Ext. - 1 bit - EQ$MCM:CC_CEXT - Specifies wehther control field of a frame may be extended.

Logical Control Field - 1 bit - EQ$MCM:CC_LCF - Specifies whether there is a logical control field in a frame.

Abort Ext. Idle - 1 bit - EQ$MCM:CC_AIDL - Specifies whether an abort is to be followed by an idle.

Two Flags Option - 1 bit - EQ$MCM:CC_2FLG - Specifies whether a flag can act as closing and opening flags at the same time.

5.7.4.2.6 Intelligent Control Terminal Port (I/CTP)

Speed - 2 bytes - EQ$MCM:CC_SPDH & EQ$MCM:SPDL - Same as for I/ATP.

Data Bits - 2 bits - EQ$MCM:CC_DBTS - Same as for I/STP.

Stop Bits - 2 bits - EQ$MCM:CC_STPB - Same as for I/ATP.

Parity - 2 bits - EQ$MCM:CC_PRTY - Same as for I/STP.

Auto Echo - 1 bit - EQ$MCM:CC_ECHO - Same as for I/ATP.

Flyback - 1 byte - EQ$MCM:CC_FLYB - Same as for I/ATP.

Garble Character - 1 byte - EQ$MCM:CC_GARB - Same as for I/ATP.

Comm. Mode - 2 bits - EQ$MCM:CC_MODE - Same as for I/NP.

5.7.4.2.7 Autospeed Definition Port (ADP)

Code Type - 1 byte - EQ$MCM:CC_CODE - Same as for I/STP.

Speed - 2 bytes - EQ$MCM:CC_SPDH & OF$CMEM:SPDL - Same as for I/ATP.

Data Bits - 2 bits - EQ$MCM:CC_DBTS - Same as for I/STP.

Stop Bits - 2 bits - EQ$MCM:CC_STPB - Same as for I/ATP.

Parity - 2 bits - EQ$MCM:CC_PRTY - Same as for I/STP.

Auto Echo - 1 bit - EQ$MCM:CC_ECHO - Same as for I/ATP.

Flyback - 1 byte - EQ$MCM:CC_FLYB - Same as for I/ATP.

Garble Character - 1 byte - EQ$MCM:CC_GARB - Same as for I/ATP.

Comm. Mode - 2 bits - EQ$MCM:CC_MODE - Same as for I/STP.

(OP) Mode - 1 byte - EQ$MCM:CC_OPMD - Same as for I/STP.

Recognition Character - 1 byte - EQ$MCM:CC_RCHR - Defines the autospeed character for this definition.

Substitution Character - 1 byte - EQ$MCM:CC_SCHR - Defines the character to be sent in place of the autospeed character.

5.7.4.3 Special CMEM Commands

Certain CMEM commands perform actions which are more complex than simply reading or writing one port or node parameter. These commands are listed here:

Copy Port - EQ$MCM:CC_CPYP - This command causes all the port parameters for the port to be copied to the port whose number is contained in the value field. The Copy Port command copies port parameters between port CMEM entries in the same configuration. It cannot be used to copy data between different configurations.

List VPs - EQ$MCM:CC_LSTV - This command causes a list of all VPs associated with the port (which must be a physical multi-threaded port) to be appended to the addressed packet. If the list cannot be appended without overflowing the maximum allowable size of an addressed packet (255 bytes) an error occurs.

The format of the list is such that the n'th byte in the list is the VP associated with thread number n. If n is not a valid thread (meaning there is no VP for that thread number) then an 0 byte is stored if n is less than the highest valid thread. The last byte in the list is the VP associated with the highest valid thread number.

The above commands ignore the high order (read/write) bit in the command code.

5.7.5 CMEM Definition

The following is a map of the CMEM entries for the D814 system. Note, all fields with the same name for different port types are assigned the same relative locations.

Node Parameters

CHKSM	CHKSM BUTH	ACMF PLTH	RPTN AVTC	RPTP REV	RDBF REVT	NODE	PORT

I/NP

GTYP STYP	NRZI	USDN	NOAK				
SPDH	SPDL						
	TIMC					PLIP	BUIP

XP

GTYP STYP	RADN	RADP	XADN	XADP	XNTP		

I/STP

GTYP STYP	DSTN	DSTP	XADN	XADP	XNTP	CALL SECL	RTNG XPTY
SPDH	SPDL	CODE	DELY	SYNC	RSYN IDLL	MODE DBTS	*1
CTSD		PPAD	THRD	CEIP	EDIP	TRNS	

I/MSTP, I/MATP

GTYP STYP							
	TIMC					PLIP	BUIP

VATP

GTYP STYP	DSTN	DSTP	XADN	XADP	XNTP	CALL SECL	RTNG XPTY
SPDH	SPDL	CODE	GARB	FLYB	FDLY	ECHO STPB MODE DBTS	*1
CTSD		PPAD	THRD	CEIP	EDIP	XONN	XOFF

I/BOP

GTYP STYP	NRZI DSTN	DSTP	BFCS XADN	XADP	XNTP	*2	RTNG XPTY
SPDH	SPDL	CLCK	AIDL		2FLG	MODE DBTS	
CTSP	TIMC			CEIP	EDIP	PLIP	BUIP

I/CTP

GTYP STYP	PSPDH	PSPDL	PSTPB PDBTS	PFLYB	PPRTY		
SPDH	SPDL		GARB	FLYB		ECHO STPB MODE DBTS	*1
	TIMC					PLIB	BUIP

ADP

GTYP STYP	RCHR	SCHR					NCR1 (*1)
SPDH	SPDL	CODE	GARB	FLYB	OPMD	CHR1 (*2)	CHR2 (*3)

*1 - Contains PRTY, ADCM, LOGG, ASP - for VATP contain DROP also.

*2 - For IBOP contains CALL, SECL.

5.7.6 CMEM Map Table

This table, which is built by MSI at system intialization from information stored in the Options ROM, describes where the four different configurations start in CMEM and how many port entries are available in each configuration. The address of this table is set up at OF$SYS:CMCMT. Each entry is three bytes long, containing: MP# - this is the number of highest port allowed for the configuration, and CMEM Base Address - this is the base address of the configuration. The table appears as follows:

MP#	Base Addr.	- Configuration #1
MP#	Base Addr.	- Configuration #2
MP#	Base Addr.	- Configuration #3
MP#	Base Addr.	- Configuration #4
0	End Addr.	- Ending Address of Config. 4 + 1

5.7.7 Options ROM Port Option Table

This table resides in the Options ROM and is used to validate a port generic type and subtype for a particular D814 sub-system. Each entry for a particular generic type is two bytes long, containing a bit for each subtype, indicating the validity of that subtype. If a bit is zero, the subtype is invalid for the system. The bit position for a particular subtype is determined from right to left by the subtype value, i.e., the LSB corresponds to subtype zero, the next higher bit to subtype 1, etc. For the generic port type to be allowed, the subtype 0 bit in the ports entry in this table must be set.

5.7.8 Summary of Commands

The following is a summary of the command codes passed to MCM in the addressed packet. The command names given are defined if EQ$MCM and are prefixed by EQ$MCM:CC_. The commands should be ORed with EQ$MCM:CC_WRITE to produce a write command.

	------Command valid for-------								
Command	NODE	I/NP	XP	ADP	I/STP	I/ATP	I/CTP	I/BOP	Field Referenced
RPTN	X								Report Node
RPTP	X								Report Port
EDTH	X								Error Density Threshold
CERT	X								Error Rate Threshold
RRTH	X								ReXmit. Rate Threshold
CETH	X								Compress. Eff. Threshold
BUTH	X								Buffer Util. Threshold
PLTH	X								Process. Loading Threshold
AVTC	X								Averaging Time Constant
REV	X								Current Software Revision Number
REL	X								Current Software Release Number
NODE	X								Current Software Source Node
PORT	X								Current Software Source Port
GTYP		X	X	X	X	X	X	X	Generic Type
STYP		X	X	X	X	X	X	X	Subtype
CODE				X	X	X			Code Type
SPDH		X		X	X	X	X	X	Speed (MS Byte)
SPDL		X		X	X	X	X	X	Speed (LS Byte)
DBTS				X	X	X	X	X	Data Bits
STPB				X		X	X		Stop Bits
PRTY				X	X	X	X		Parity
ECHO				X		X	X		Auto Echo
FLYB				X		X	X		Flyback
GARB				X		X	X		Garble Char.
MODE		X		X	X	X	X	X	Comm. Mode
OPMD				X	X	X			(OP) Mode
DELY					X				Delay
RTNG				X	X	X		X	Routing
DSTN					X	X		X	Dest. Node
DSTP					X	X		X	Dest. Port
XADN			X		X	X		X	Xmt. Adj. Node
XADP			X		X	X		X	Xmt. Adj. Port
XPTY					X	X		X	Xmt. Path Priority
XNTP			X		X	X		X	Xmt. Network Port
RADN			X						Rcv. Adj. Node
RADP			X						Rcv. Adj. Port
RCHR				X					Recognition Char.
SCHR				X					Substitution Char.
CPYC									Copy Config.
CPYP									Copy Port
EMPC									Empty Config.

(Continued on next page.)

	Command valid for								
Command	NODE	I/NP	XP	ADP	I/STP	I/ATP	I/CTP	I/BOP	Field Referenced
LSTV									Append List of VP's associated with a multi-threaded port (ordered by thread number)
CTSD				X	X	X		X	RTS/CTS Delay
SECL				X	X	X		X	Security Level
CALL				X	X	X		X	Call Type
TIMC		X		X	X	X	X	X	Time Constant Factor (Statistics)
USDN		X							User Data Rate Threshold (/100)
NOAK		X							No-Ack Timeout
NRZI		X						X	NRZ/NRZI Coding
BFCS								X	Bad FCS Option
AEXT								X	Address Field Extension Option
CEXT								X	Control Field Extension Option
LCF								X	Logical Control Field Option
AIDL								X	Abort followed by Idle
2FLG								X	2 Flags/1 Flag Option
CEIP					X	X		X	Compression Efficiency Threshold
EDIP					X	X		X	Error Density Threshold
PLIP		X			X	X	X	X	Processor Loading Threshold
BUIP		X			X	X	X	X	Buffer Utilization Threshold
PPAD									Physical Port Address (valid for all VP types)
THRD									Thread Number (valid for all VP types)

5.8 Mainframe Network Link Control Module (MNL)

5.8.1 Functional Specification

The processing of network link data is divided into two software modules, one residing in the mainframe and the other residing in the I/NP. The purpose of this section is to describe the functions of the Mainframe Network Link Control Module (MNL).

This section references the D814 Bus Interface Chip Functional Specification and the reader should be familiar with that document.

5.8.1.1 General Description

The MNL module is responsible for assembling data into 'frames' (or blocks) to be moved (via the I/NP) across a network link to another node in the network. It is also responsible for receiving such frames (via the I/NP) from adjacent nodes and distributing the data contained in these frames to the appropriate buffers, I/TPs and/or system modules in the local node. The function of the I/NP is to move the frames of data built by the MNL module error free across network links. The interface between MNL and the I/NP is the Bus Interface Chip (BIC), and in particular the BIC data FIFOs. In performing its functions, MNL handles all data passed between the mainframe and the I/NP via the BIC data FIFOs.

In addition to assembling frames for transmission to other nodes and distributing frames received from other nodes, the MNL module has responsibility for the local 'transmission' of data between I/TPs co-located at a node. This function will be referred to as the 'Local NP' function. The 'Local NP' will always exist at a node, and will be automatically configured at system initialization time. The port address X'01' is the reserved 'Local NP' address and no other port may be configured at that address.

MNL is also responsible for monitoring for certain system error conditions. These will be explained below.

5.8.1.2 Frame Types

There are three types of frames MNL sends to and receives from other nodes. These are Control Frames, Addressed Packet Frames and Data Frames. Control Frames contain system control information such as system boot information, routing control, path management control, and link set up and initialization information. Addressed Packet Frames contain system and/or user packet messages that are to be moved through the network via the 6050 dynamic packet system. Data Frames contain user data received from I/TPs that must be moved through the network. Control Frames are always given priority over Data and Addressed Packet Frames by MNL.

The 'Local NP' does not use a frame structure and therefore does not transmit either Control Frames or Data Frames. Furthermore, addressed packets are not sent via the 'Local NP'.

5.8.1.3 Internal Frame Structure

Information passed between the MNL module and I/NP is transmitted in frames through the BIC data FIFOs. These frames do not have the same format as the frames sent over the high speed network link. To distinguish between these two types of frames, we will call frames which are exchanged between the MNL module and the I/NP, Internal Frames. The formats of the three types of Internal Frames are defined below. The first byte of any frame exchanged between the mainframe and I/NP contains a type identifier which designates the frame that follows as either control, addressed packet or data.

5.8.1.3.1 Internal Control Frame Format

Internal Control Frames consist of multiple fields as follows:

1. Frame Type Identifier (FTI): A one byte field equal to X'CO' for control Frames.

2. Control Message Fields: A control message consists of three sub-fields as follows:

   a. Length/Terminator Subfield - A one byte subfield. If equal to X'01' this subfield terminates the control frame. If not equal to X'01', this subfield equals the total number of bytes in the current control message field (including the length byte).

   b. Control Message Body - A multiple byte field of data which is passed to the system module which processes the control message.

X'01'	control message "n" body	control length "n"	control message 1 body	control length 1	FTI X'CO'

>

Internal Control Frame Format

5.8.1.3.2 Internal Addressed Packet Frame Format

Internal Addressed Packet Frames consist of multiple fields as follows:

1. Frame Type Identifier (FTI): A one byte field equal to X'80' for Addressed Packet Frames.

2. Addressed Packet Fields: An addressed packet consists of two subfields as follows:

   a. Length/Terminator Subfields - A one byte subfield. If equal to X'01', this subfield terminates the addressed packet frame. If not equal to X'01', this subfield equals the total number of bytes in the current addressed packet field (including the length byte).

   b. Addressed Packet Body - A multiple byte field of data which is passed to the system router which processes the addressed packet when it arrives at the remote node.

X'01'	address packet "n" body	address packet "n" length	address packet 1 body	address packet 1 length	FTI X'80'	>

Internal Addressed Packet Frame Format

5.8.1.3.3 Internal Data Frame Format

The Internal Data Frame Format consists of multiple fields as follows:

1. Frame Type Identifier: A one byte field equal to X'40' for Data Frames.

2. Slot Fields: Multiple byte fields containing one or more Data Slot Subfields. The format of the Data Slot subfield is as follows:

   i) Address/Terminator Subfield - A one byte subfield. If equal to 'X'01', the Frame is terminated. If greater than X'01' but less than or equal to X'FF', then this field specifies the port address at the remote node to which the nibble Data Subfield is to be delivered. X'00' is not allowed.

ii) Nibble Data Subfield - A multiple byte subfield where each byte contains one or two 4 bit code segments, called nibbles. The format of the bytes is as follows:

a) X'ab' (a .ne. 0, b .ne. 0) - 2 data nibbles

b) X'0b' (b .ne. 0) - 1 data nibble

c) X'a0' (0<a<F) -in-channel control signal (ICS)

d) X'F0' - nibble data slot terminator

e) X'00' is not allowed

	X'01'	X'F0'	DATA	ADR	X'F0'	DATA	ADR	FTI X'40'	>

Internal Data Frame Format

5.8.1.3.4 BIC Data FIFO Coding

The BIC data FIFOs are implemented in hardware in such a way that reading an empty FIFO will return a value of X'00'. This scheme allows software to eliminate a status check of a FIFO before a read <u>if</u> X'00' is never stored in the FIFO. This feature is used by the MNL Module and I/NP to speed up processing and reduce I/O bus overhead (e.g., status reads).

In order to take advantage of this feature, data transferred through the FIFO must be coded such that X'00' never appears. The following coding scheme is used for passing data through the BIC data FIFOs:

Data Byte	FIFO Code
X'01 --> X'FE'	X'01' --> X'FE'
X'00'	X'FF',X'FE'
X'FF'	X'FF',X'FD'

This coding causes approximately a 1 percent increase in bus transfer overhead, but reduces software overhead by as much as 25 percent per byte transferred.

5.8.1.4 Port Control Block Interface

MNL uses the Port Control Block (PCB) both as a shared data interface with other Mainframe modules and as its own data area for data relevant to a particular port. The PCB and its various substructures is described in the section on Subsystem Data Structures. These modules interface with MNL through the PCB:

Mainframe Path Management, Routing, and Congestion Control Module (MPMRCCM).

Mainframe Addressed Packet Control Module (MAP).

Mainframe Statistics and Monitoring Module (MSM).

Mainframe Multithreaded Port Control Module (MMP).

The PCB interface with each of these modules is described in the section of this document devoted to that module and will not be described further here.

5.8.1.5 Detailed Functional Description

The MNL Module performs several distinct functions. These are listed below and are discussed in detail in the following subsections.

1. Initialization
2. Link Start Up
3. Internal Frame Transmission
4. Internal Frame Reception
5. 'Local NP' Processing
6. Failure Recovery
7. Miscellaneous Utility Functions

5.8.1.5.1 Initialization

The MNL initialization routine MNL$INIT:INP is called by the system initialization module (MSI) at system boot time. It is called once to initialize the 'Local NP' module and once for each I/NP configured in the system.

When the initialization routine is called, the address of the corresponding I/NP PCB is passed in the X register. At this time all data structures and lock byte areas have been allocated for the corresponding I/NP, software has been loaded to the I/NP, and the D814 mainframe multitasking operating system is running.

The initialization of the MNL Module for the I/NP consists of the following steps:

1. The I/NP's PCB status bits are initialized for link start up.

2. The Control Frame queue, Remote Addressed Packet queue, and Slot Add queue are set up and initialized.

3. Two data spaces are obtained from the free data space queue, one for the transmitter module and one for the receiver module.

4. Exit initialization.

The initialization of the MNL Module for the 'Local NP' consists of the following steps:

1. A Slot Add queue is set up and initialized for the 'Local NP'.

2. The 'Local NP' PCB is initialized to the start up state.

3. The 'Local NP' software module is started.

4. Exit initialization.

If any error occurs such that the NP cannot be initialized, MNL$INIT:INP returns to the caller with CC:Z=1. Otherwise it returns with CC:Z=0.

5.8.1.5.2 Link Start Up

When an I/NP starts up, it initializes its software and sends an 'I/NP Active' packet to the MNL module. Upon receiving this packet MNL initializes its transmitter and receiver modules and sends the 'Start Up Link' packet to the I/NP.

When the I/NP receives this packet, it reinitializes the link and tries to establish communications with the remote node. When communication is achieved, initialization parameters are exchanged between the local and remote nodes. The I/NP then sends the 'Link Up' packet to MNL with the following initialization parameters included:

1. Remote Node's Number
2. Remote I/NP's Port Address
3. Remote Node's Software Level
4. Remote Node's Configuration Number
5. Local I/NP Line Speed (BPS)
6. Round Trip Link Delay (milliseconds)

When the 'Link Up' packet arrives, MNL notifies the system Down-Line Load module of the new link. Then the remote node's software level is compared to the local node's software level. If they are different, MNL exits. Otherwise, the system boot module is notified of the new link's presence. Next the remote node's configuration level is compared to the local node's configuration level. If they are different, MNL exits. If they are the same, the link is brought up normally, the routing manager is informed of the new link and the node number of the remote node, and MNL exits.

5.8.1.5.3 Internal Frame Transmission

Internal frame transmission to the I/NP is handled by the MNL Transmitter Module. The basic function of the transmitter is to collect data to be transmitted to the remote node, form that data into internal frames, and transmit the internal frames to the I/NP through the BIC data FIFOs. In performing this basic task, the transmitter must also maintain its data structures, accept commands from its slot Add Queue, and monitor for 'KILL' signals.

5.8.1.5.3.1 MNL Transmitter Data Structures

The MNL transmitter is concerned primarily with four data structures, the Control Frame Queue, the Remote Addressed Packet Queue, the Slot Add Queue, and the Transmit Slot List.

Control Frame Queue:

The control frame queue is a queue data structure (see section on MUT, subsection on MUT$QUE) which contains control messages to be transmitted. Control messages are stored in Byte File format (see section on MBM, subsection on MBM$BFILE).

Remote Addressed Packet Queue:

The Remote Addressed packet Queue is a queue data structure which contains addressed packets to be transmitted. The addressed packets in the queue are in Byte File format.

Slot Add Queue:

The Slot Add Queue is a Byte Queue (see section on MBM, subsection on MBM$BQUE) which contains port addresses of slots to be added to the Transmit Slot List.

Transmit Slot List:

The Transmit Slot List is a linked list of slots which are to be transmitted over the high speed network link. The entries in the list are the slot substructures of PCB's whose ports MNL scans for data to be included in the link traffic. (See subsection on PCB in Subsystem Data Structures section.)

5.8.1.5.3.2 MNL Transmitter Functional Description

The MNL Transmitter Submodule sends internal frames to an I/NP through the outbound BIC data FIFO when the I/NP requests them. For each frame request, the transmitter will send a control frame if any control messages are queued. If the queue is empty, MNL will send alternating Addressed Packet and Data frames. If no addressed packets are queued, only data frames are sent.

If a data frame is to be sent, the Slot Add Queue is checked and any new slots to be added to the frame are linked in. Following this, each slot in the Transmit Slot List is serviced. If no data is to be sent for a slot, the slot is not included in the frame. If data is to be sent, up to approximately a slot weight worth of nibbles are sent. Each slot is monitored for 'KILL' signals as it is serviced. If a 'KILL' signal is detected, special 'KILL' flags are set. When all slots have been processed, the frame terminator is sent. If at the end of a frame 'KILL' flags were set, the transmitter rescans the Transmit Slot List, unlinks the slots for which 'KILL' signals were detected, and sends the appropriate messages to the Path Manager Module. The transmitter than waits for the next frame request from the I/NP.

5.8.1.5.4 Internal Frame Reception

The reception of internal frames from an I/NP is the job of the MNL Receiver Module. The I/NP delivers frames to the receiver one at a time in the order in which they are transmitted from the remote node.

5.8.1.5.4.1 MNL Receiver Data Structures

The receiver interacts primarily with three data structures, the control Frame Dispatch Table, the Port Directory, and XP Byte FIFOs.

Control Frame Dispatch Table:

The Control Frame Dispatch Table defines which system modules are to process control messages received over the high speed link. The entries in the table are indexed by the control frame destination code included in each control message.

Port Directory:

The Port Directory is a table of pointers to Port Control Blocks (PCB). Each port defined in the system must have a PCB. This table is indexed by port address.

XP Byte FIFO:

XP Byte FIFOs are Byte FIFO data structures (see Section 6.3) used for buffering path data at intermediate nodes on a path.

5.8.1.5.4.2 MNL Receiver Functional Description

The MNL Receiver Module receives frames in the 'Internal Frame' format from the I/NP through the inbound BIC data FIFO. Its main function is to distribute the data contained in the frames to the appropriate system modules and buffers.

When a Control Frame is received, the control frame distribution code in each control message is used to dispatch each control message to the proper module for processing.

When an Addressed Packet Frame is received, all addressed packets in the frame are sent to the MAP$ROUTE module for processing.

When a data frame is received, slot data must be distributed. Slot data is distributed differently for XP's and I/TP's. For an XP data slot the data is stored in the XP Byte queue. For an I/TP, the data is sent directly to the I/TP via the outbound BIC data FIFO. If the FIFO is full, the receiver waits for a short time (about 10 milliseconds) for it to go non-full. If it does not go non-full, the I/TP is declared dead, data to the port is discarded, and 'clear call' message is sent to the PMM.

5.8.1.5.5 'Local NP' Processing

The 'Local NP' Module is responsible for two basic functions. The first is to move data between co-located I/TPs so that local I/TP - I/TP communications is possible. The second is to flush data from XP byte queues on paths whose transmit I/NP link has failed. In performing each of these functions the 'Local NP' must appear functionally similar to a 'real' I/NP and respond to 'KILL' signals, 'add slot' commands, and send appropriate messages to the rest of the system as error or failure conditions may require. The 'Local NP' module does not process either control frames or address packets.

5.8.1.5.5.1 'Local NP' Data Structures

The 'Local NP' module has several data structures it utilizes in performing its functions. These include the Slot Add Queue, Transmit Slot List and Port Directory. (These data structures were described in the previous two subsections.)

5.8.1.5.5.2 'Local NP' Functional Description

The 'Local NP' services its Transmit Slot List periodically. The period is dynamically set by the module in response to the load requirements measured during the last service period. In no case will the period be more than 50 milliseconds between service intervals or less than 20 milliseconds.

At the beginning of each service period the Slot Add Queue is checked and any new slots to be added are linked onto the Transmit Slot List. Then each slot in this transit list is processed. If a slot is an I/TP to I/TP path, data is moved directly from FIFO to FIFO. If any outbound FIFO becomes full, the 'Local NP' sets a flag in the port's slot data structure and proceeds to the next slot in its transmit list. If the FIFO remains full for 4 scans, the I/TP is declared dead, and a 'clear call' message is sent to the MPM. If a full FIFOs worth of data or more is moved during the servicing of any slot, the 'Local NP' service period is reduced by 1 millisecond. If no slot has a full FIFOs worth of data to move, the service interval is increased by 1 millisecond. If a slot is an XP slot data is flushed from the XP byte queue until empty or 'KILL' signal is seen.

For all slots, if a 'KILL' signal is encountered, the slot is unlinked from the transmit list and the appropriate message is sent to the PMM. At the end of the transmit list processing, the 'Local NP' delays itself until the next service period is to begin.

5.8.1.5.6 Failure Recovery

There are two basic failure modes associated with an I/NP. The first is an I/NP failure and the second is a high speed network link failure.

The Mainframe Diagnostic Monitor Module (MDM) is responsible for detecting I/NP failures. When such a failure is observed the MNL failure module is called so that the appropriate shut down of the link is done. Once this is finished, the MDM takes over again to perform diagnostics on the IP to test its hardware viability. If no fault can be detected in the hardware (a software failure is assumed), MDM reloads the I/NP software and starts the MNL high-speed link recovery procedure.

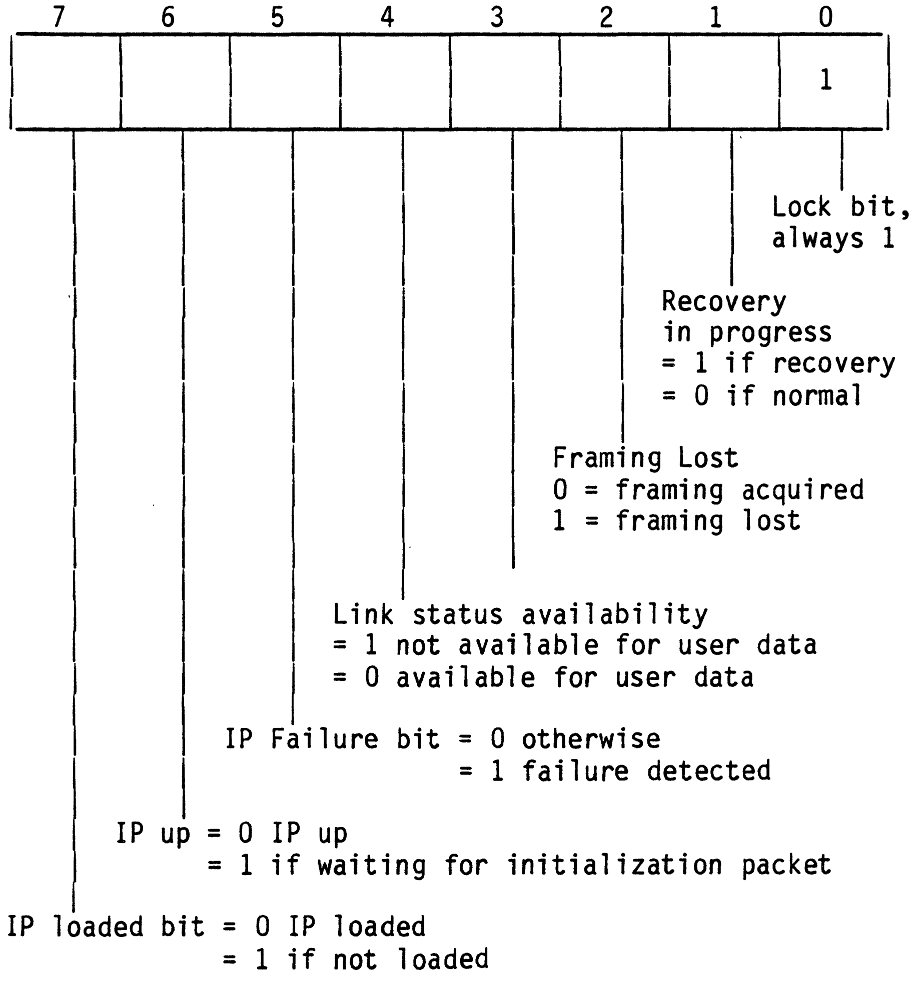
If the I/NP detects a link failure condition, it sends the 'framing lost' message directly to MNL which then starts the link recovery procedure.

5.8.1.5.6.1 Failure Interlocks

Before starting failure recovery for any port, all tasks which use the port data structures must be locked out of those data structures so that they do not interfere with the recovery. To accomplish this, two lock bytes are used to interlock between port failures and user tasks. The first lock byte is called the Status Lock Byte and the second lock byte is called the User Lock Byte.

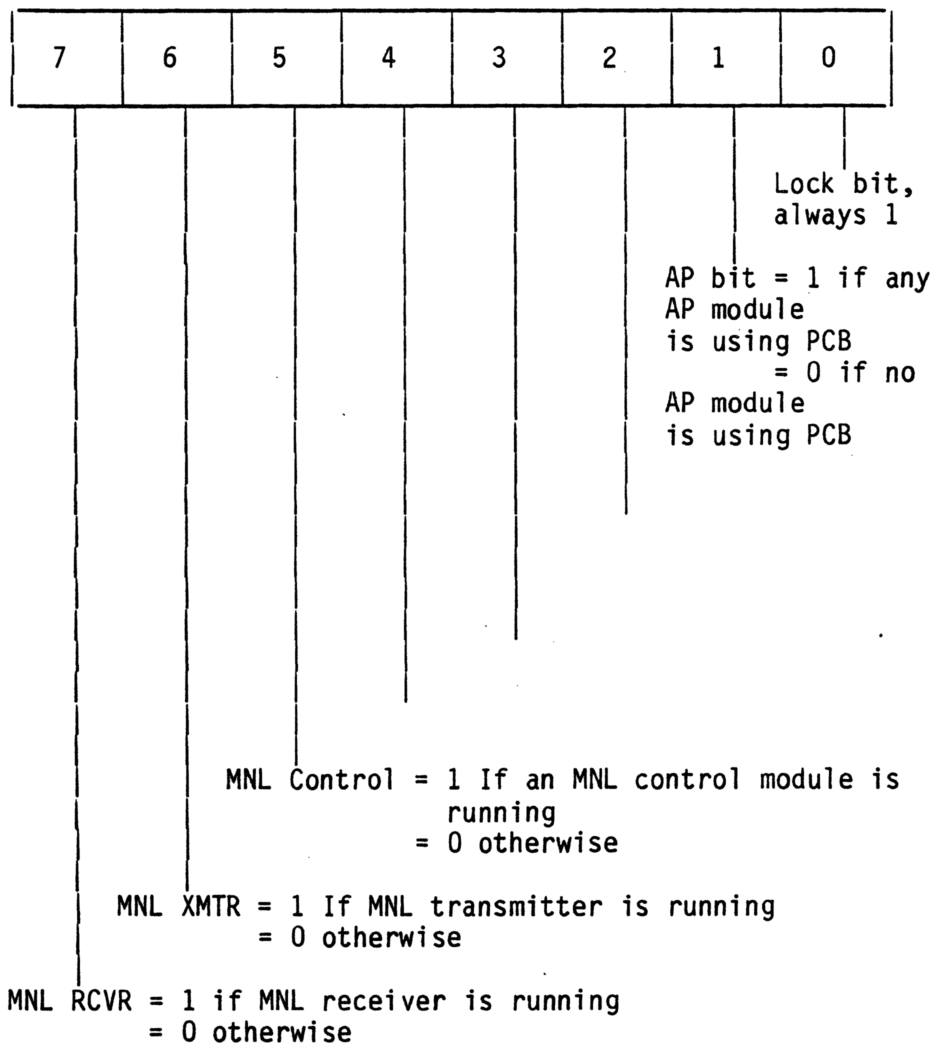
Status Lock Byte

This lock byte contains port status flags. If any fail status bits are set in this byte a user task may not access the related port data structures. The format of this byte is as follows:


7
6
5
4
3
2
1
0
1
Lock bit, always 1
Recovery in progress = 1 if recovery = 0 if normal
Framing Lost 0 = framing acquired 1 = framing lost
Link status availability = 1 not available for user data = 0 available for user data
IP Failure bit = 0 otherwise = 1 failure detected
IP up = 0 IP up = 1 if waiting for initialization packet
IP loaded bit = 0 IP loaded = 1 if not loaded

User Lock Byte

This lock byte contains flags which are set by various user tasks when they are accessing port data structures. Any user task must lock the status lock byte before setting a user flag. If the status of the port is inconsistant with using a desired port data structure, the user task may not use that data structure and must exit not setting a user flag and restoring the status lock byte. The format of the MNL user lock byte is as follows:


7
6
5
4
3
2
1
0
Lock bit, always 1
AP bit = 1 if any AP module is using PCB
= 0 if no AP module is using PCB
MNL Control = 1 If an MNL control module is running
= 0 otherwise
MNL XMTR = 1 If MNL transmitter is running
= 0 otherwise
MNL RCVR = 1 if MNL receiver is running
= 0 otherwise

5.8.1.5.6.2 MNL Hardware Failure Submodule

When this submodule is called, the MDM module has detected an I/NP failure and has set the IP fail status bit. This submodule is responsible for the orderly shut down of the link and notifying other system modules of the failure.

The link shut down procedure consists of the following steps:

1. Notification of the link failure is sent to the MRM, MPM, MSB, and MDL modules.
2. Wait for all users to stop using the I/NP's data structures.
3. Return all packets on the Remote Packet Queue and IP Outbound Packet Queue to the router.
4. Throw away all Control Frames in the Control Frame Queue.
5. Transfer all XP slots in the Transmit Slot List and in the Slot Add Queue to the 'Local NP' via its Slot Add Queue.
6. Set the transmitter and receiver initialization bits so that they will start up in the correct mode.
7. Set the 'Framing Lost' status bit in the port's status lock byte.
8. Return to the MDM module for IP testing.

5.8.1.5.6.3 Link Failure Recovery

This module may be called from the I/NP via a 'framing lost' message or from the I/NP after a new software load.

If the 'framing lost' status bit is not set then the link failure was detected by the I/NP. Set the 'recovery in progress' bit in the status lock byte and execute Steps 1 through 7 of the link shut down procedure described in the previous section. Then clear the 'recovery in progress' bit and send the 'Activate Link' message to the I/NP and terminate.

If the 'framing lost' status bit is already set, link shut down has already been done. Send the 'Activate Link' message to the I/NP and terminate.

5.8.1.5.7 Miscellaneous MNL Utility Functions

The MNL module provides several utility functions that are used by MNL and other system modules to perform miscellaneous functions. The entry points for these functions are described here:

Entry MNL$UTIL:Q2RAPQ

Queues an addressed packet to a remote addressed packet queue for transmission by MNL.

Entry conditions --

X register -- Addressed packed bytefile header address

B register -- Port number of the I/NP

Exit conditions --

X, A registers -- Destroyed

B register -- unchanged

CC:C -- Set if and only if I/NP is down

Data space -- OF$DS:BFADR, OF$DS:BFTMP destroyed

Entry MNL$UTIL:SENDCF --

Enqueues a control frame for transmission by MNL

Entry conditions --

X register -- Points to control frame bytefile header

Exit Conditions

All reegisters destroyed
Data space - may be destroyed
If the I/NP is up, the control frame is enqueued. If not, it is deleted.

Entry MNL$UTIL:DISTCF --

Distributes a control frame or a control-frame format message

Entry conditions --

X register -- Points to header buffer of message bytefile

<u>Exit Conditions</u>

Data space - destroyed
Control is passed to message handler (via jmp) with pointer to the message bytefile stored in OF$DS:BFADR. The message handler must terminate in a RTS.

Entry MNL$UTIL:KILLSLOTSWITCH --

Kills a slot and sends ICSKILLFAIL along path

Alternate entry MNL$UTIL:KILLSLOTFAIL --

Kills a slot and sends ICSKILLFAIL

Entry conditions (called by MPM) --

X register -- Points to PCB's path data substructure

Data space -- OF$MPM:DS_PCBPTR points to the port's PCB. OF$MPM:DS_PATHDSS points to path data substructure.

<u>Exit Conditions</u>

A register -- Destroyed
B register -- unchanged
X register -- unchanged

This routine sets the appropriate fail bits in the slot data substructure causing MNL to delete the slot.

Entry MNL$UTIL:ADDSLOT --

Adds a slot (called by MPM when a path becomes Active)

Entry conditions --

B register -- Port number of the port (XP, ITP, or VP) whose data is to be included in the link traffic

A register -- I/NP port number

<u>Exit Conditions</u>

All registers destroyed
Port number is queued on the appropriate slot add queue. EQ$PCB:LOCK_USERS_SLOT is set in source port's user lock byte.

Entry MNL$RECVRY:MRMFAILACK --

Receives acknowledgement to LDOWN message sent to MRM

Entry conditions --

B register -- I/NP address

Called by MRM when processing of LDOWN is complete

Exit Conditions

All registers destroyed
Data space - OF$DS:BFADR, OF$DS:MFTMP destroyed

Entry MNL$RECVRY:MPMFAILACK

Called by MPM to acknowledge receipt of the LINKFAIL message

Entry conditions --

B register -- contains I/NP address

Called after MPM has received and processed the LINKFAIL message from the I/NP in B register

Exit Conditions

All registers destroyed
MNL's link failure cleanup is complete and the link is ready to be brought back up

5.9 Mainframe Downline Load Module

Introduction

This module will service the following requests from the Mainframe Diagnostic Module (MDM):

1. Load a local port(s) with the specified software. (Note that I/NP, I/GBNP, I/FDP are all special cases. See Section 5.9.1.2.)

2. Load a specified software to a remote node through a specified local port.

3. Upload of mainframe software to a neighboring node, through a specified local port.

4. Upload of mainframe software to a remote node so that the remote node can pass it on to its neighbor that is not running.

5. Abort loading of a specified port(s).

MDL will inform MDM upon the completion of each load; and in case a certain port cannot be loaded due to some unrecoverable error, MDM will be informed and the request terminated. Otherwise MDL will retry indefinitely to fulfill the request.

Sources of each software will not be specified by MDM. However, the following rules will always apply:

1. There will be a list of software names; and if any of these are requested, MDL will first search for that software in a running port or mainframe.

2. All other software will come from the floppy only.

3. Updated software running in a port or mainframe will not be chosen as a source of software.

4. In choosing a source of software from a port, the ability of the port to upload software in terms of resources such as processor loading will be taken into consideration.

5.9.1 MDL Algorithm Main Features

Where possible, port software is loaded from an already-loaded local port. A node needing software not available locally locates a source of that software by a broadcast mechanism combined with a point-to-point addressed packet protocol.

The algorithm attempts to allocate bandwidth so that usually no more than one downline load is going through any one link in each direction.

In general, software should be loaded from a nearby source of the needed software although the algorithm is not optimal in this respect.

A more detailed description of the algorithm follows.

5.9.1.1 Bandwidth Availability

We say that outbound bandwidth is available to load a remote node B from local node A if the first link on the shortest path known to the Routing Manager from A to B is not currently on the shortest path known to the Routing Manager from A to any other node now being loaded.

Similarly, we say that inbound bandwidth is available in the above situation if the last link on the shortest path from A to B is not currently on the shortest path from B to any other node now being loaded.

Bandwidth availability is determined from the Inbound and Outbound Load Lists. The Inbound Load List lists all nodes from which the local node is loading along with the corresponding software types being loaded. The Outbound Load List lists all nodes to which the local node is loading along with the software types being loaded. When inbound or outbound bandwidth is to be allocated, the proper load list is scanned and it is determined if sufficient bandwidth exists, as explained above. If so, the new node and software type are entered. Bandwidth is de-allocated by simply deleting the proper entry from the load list.

This scheme does not guarantee at all times that no link will be used for more than one inbound and one outbound load to/from the local node, but it does make that infrequent.

5.9.1.2 Loading of Special Ports I/NP, I/GBNP, I/FDP

All local ports can be loaded in the same way by the mainframe first obtaining the software and then passing it to the port through the BIC.

For I/NP and I/GBNP since there is always another port of the same type at the other end of the link, the needed software can be obtained through that link. An I/FDP can be loaded from the floppy that may be mounted on one of its drives. In all three cases, MDL will attempt to load them by all possible means simultaneously and the method that initiates the loading first will be the chosen method.

5.9.1.3 Local Loading

This section and Remote Loading (Section 5.9.1.4) will discuss the algorithm used to locate the best source of software. Best in the sense that it will complete the load as quickly as possible and with minimal impact on ongoing network activities, if the network is already running.

For each request, MDL will first determine if the software requested must come from a floppy. If so, a local floppy is searched for. If none exists or software not on the floppy, then a remote floppy is searched for by broadcasting a request to MDL in other nodes. If the software can come from a running port, then a packet is sent to each eligible port in the local node requesting for possible upload. Responses are then collected over a period of time, and one is chosen by means of an indicator in the reply packet from the ports.

If no local ports are available or cannot upload due to some conditions such as no available resource to upload or software has been edited, then a remote source is searched for by means of a broadcast mechanism.

5.9.1.4 Remote Loading

In repsonse to a broadcast request for software from another node, MDL will do the following:

1. Determine if bandwidth is available; if not, then rebroadcast the request with the requesting node as the root address.

2. If bandwidth is available, then search for a source locally in the similar manner as described in Section 5.9.1.3. If software is not available, then rebroadcast as above. If software is available, then a message is returned to the requestor with the information as to the port address which has the software.

It is then left to the requesting MDL to communicate with the source port to obtain the software. The source MDL will not intervene other than receive messages from the requesting MDL at the beginning and the end of an upload. The purpose of this is to update the bandwidth load table.

The following describes how messages are passed between MDL in different nodes in locating software.

MDL State

MDL maintains a load state for each software type. This state may have the following values:

Not Interested (NI) - This node neither has nor wants the software.

Loaded - This node can supply the software to remote nodes.

Needy - This node needs and has broadcast a request for the software.

Waiting for Software (WFS) - This node has sent a Send Me Software message to a potential source node and is waiting to complete a downline load.

MDL Messages

The following messages are sent between MDL modules:

Broadcast Software Request (Software Type, Source Node) - Abbreviated BSR. Sent as control frame along the minimum depth spanning tree rooted at originating node. It is a general request for sources of the specified software to make their whereabouts known.

A BSR received over a link with available inbound bandwidth is retransmitted over all outgoing links with available outbound bandwidth of the minimum depth spanning tree rooted at the source node of the BSR. If there is no available inbound bandwidth over the source link, then the BSR is not broadcast. Similarly, the BSR is not sent over any outgoing link without available bandwidth. This bandwidth checking helps to minimize congestion while multiple downline loads are in progress.

I Have Software (Software Type, Source Node) - Abbreviated IHS. Addressed packet sent in response to a Broadcast Request indicating that Source Node has the software and sufficient bandwidth to send it to the receiving node.

Send Me Software (Software Type, Source Node, Highest Address, so far, Loaded) - Abbreviated SMS. Addressed packet requesting software. Used to pace addressed packets containing downline load data.

MDL State Machine

MDL may be thought of as a set of state machines, one for each software type. The MDL state machine is summarized in the diagram. The entry in the table corresponding to a given state and event is the action taken by MDL if the event occurs while in that state. State transitions are shown by placing the new state in parentheses.

MDL State Diagram

Event	Loaded	Needy	NI	WFS
BSR Received	Send IHS if bandwidth available	Broadcast on originating node's spanning tree	Broadcast on spanning tree	Broadcast on spanning tree
SMS Received	Allocate bandwidth if not already allocated. Send software if allocation successful.	Ignore	Ignore	Ignore
IHS Received	Ignore	Send SMS (WFS)	Ignore	Ignore
Software Received	Ignore	Ignore	Ignore	If last block, then (LOADED); Else send SMS if needed.
Timeout	De-allocate bandwidth for any node not heard from since last time out	Broadcast BR if bandwidth available for load	Ignore	Broadcast BSR (NEEDY)
Port load requested by MSI or MDM	Load software	Remember that this port needs software	Broadcast BSR (NEEDY)	Remember that this port needs software

5.9.2 External Interfaces

This section describes all the interfaces between MDL and the other modes in the network. In all cases except for the actual upload of port software to a remote node from a local port, the local MDL will communicate with the local port.

5.9.2.1 MDL - MDM Interface

Requests to load software is assumed to come from MDM. All requests are passed in a packet (see MDM specification for a detailed description of these packets).

Except for the loading of a remote port, through a local I/NP or I/GBNP, MDL will assume that all destination ports are not running and is ready to receive reset 1 (see IP ROM Section).

MDL will not check the compatibility of the software and the receiving port. When loading a local port, MDL will check for the port type to see if its one of the I/NP or I/GBNP to start a special loading sequence. Otherwise, no other check is done.

Unless an unrecoverable error occurs or a request is received to stop loading a port, MDL will try indefinitely to load that port with the requested software. For each request, MDL will inform MDM when the loading is complete, via an addressed packet.

5.9.2.2 MDL - MIL Interface

When loading a local I/NP or I/GBNP, MDL will attempt to do this by getting software from the port at the other end of the link, as well as the normal way. To do this, MIL$LOAD_DNP is called to load the port with sufficient software to communicate across the link. MDL will wait for a response from MIL before doing anything else with the port.

5.9.2.3 MDL - FDP Interface

MDL will interface with I/FDP to obtain software form one of the disks mounted or to give commands to the I/FDP to load itself. See I/FDP specification for a detailed description of the commands.

5.9.2.4 MDL - Running Port

MDL will call IP$UTIL:MEMUPD to request for possible software upload or for specific software. See the section on IP$UTIL:MEMUPD for more detail on the interface.

5.9.2.5 MDL - I/NP or I/GBNP

When uploading software to a neighboring node that is not running, MDL will have to communicate with a local I/NP or I/GBNP that is connected to it. The format and the sequence of the upload is as follows.

Data passed are split up into blocks and formatted as shown below. The local port will then reformat this information before passing it to the neighbor.

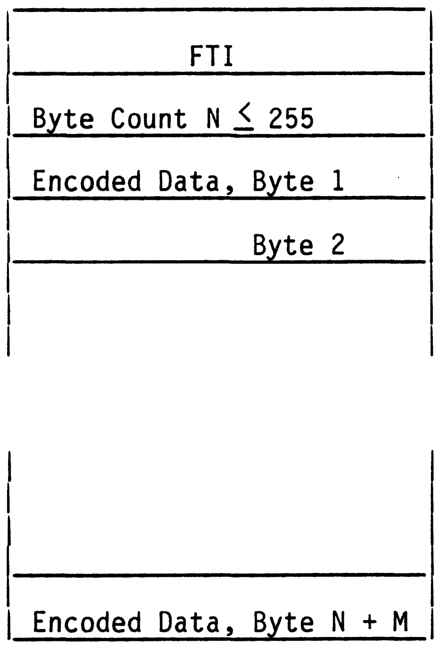
Each block will be of the same format, except that the first block will have a command byte (supplied by MDM) in the first byte. This command will indicate to the receiving port in the neighboring node as to the type of software passed.

Data passed are encoded as follows:

X'00' --> X'FFFF'

X'FF' --> X'FFFE'

Note that M = # of bytes encoded, and this is not included in the byte count.



5.9.2.6 MDL and Mainframe Updating Module

There is a requirement that Mainframe software that is already updated will not be passed to an IPLing mainframe. To accomplish this there will be a lockbyte which will indicate one of four states: (1) lockbyte not available; (2) lockbyte available, software already updated; (3) lockbyte available, software not yet updated; and (4) lockbyte available, uploading in progress.

5.9.2.7 MDL and Other Mainframe Modules

MRM$BROADCAST:ENTRY is called to get list of neighboring nodes for broadcasting purposes.

MRM$ROUTE:PACKET is called to send packets.

5.9.3 MDL Structure

This module consists of six submodules:

1. MDL$REQUEST
2. MDL$BROADCAST
3. MDL$LOCATOR
4. MDL$LOADER
5. MDL$XPORTS
6. MDL$TIMER
7. MDL$EXEC

The descritpion of each submodule and their functions are in Section 5.9.3.1 to 5.9.3.6. In designing this module, the following are the major goals:

1. Wherever possible, all ports requiring the same software are loaded at the same time.

2. Any request that requires software that is already in the progress of downline loading will be delayed until the loading is complete. This does not apply to software that must come from a floppy disk.

3. MDL controls buffer usage for downline loading. Buffer availability will be monitored, and request for software will not be generated if a threshold is exceeded. In all interfaces MDL dictates the flow of data into the mainframe.

5.9.3.1 MDL$REQUEST Submodule

All requests to MDL are assumed to come from the local MDM. All requests must be initially processed by this submodule. All messages returned to MDM come from this submodule, and any requests that are rejected are sent back to this submodule. The following are the major functions of this submodule:

1. Check validity of request, i.e., that the port address is legal and that it is configured.

2. For each new request, the outstanding requests are checked for any that include port addresses in the new request. Any downline loading on those ports are then terminated.

3. Requests that require software that need not come from a floppy disk and are in the process of downline loading will be delayed until the loading is complete.

4. For special ports, I/NP, I/GBNP and I/FDP, MDL$XPORT is called to initiate the alternate method of loading.

5. Make up packets and send to local MDM to inform the end of each request.

5.9.3.2 MDL$BROADCAST

This submodule is responsible for locating the software source that is nearest to the requestor. It accepts requests from the local MDL as well as remote MDL.

For remote nodes, MDL$LOCATOR is called to locate a local source. If the source exists, then the original requestor is informed. If none exists, then the request is rebroadcast and then deleted.

For local ports, MDL$LOCATOR is called. If the source exists, then MDL$LOADER is informed. If not, a request is broadcast and the whole procedure is repeated every 6 seconds until a source is found.

5.9.3.3 MDL$LOCATOR

This submodule deals with the locating of software at the local node. This includes communicating with local I/FDP and searching for software in one of the mounted floppy's.

For each request, this submodule performs the search once and reports on the success or failure and the whereabouts and type of source if found.

In locating a source of software from local running ports, this submodule will choose a source that is most able to upload software.

In all cases a response is sent back to the user whether a source is found or not. All requests are assumed to come from the local MDL$BROADCAST.

5.9.3.4 MDL$LOADER

This submodule handles the loading of local ports and neighboring nodes through a local port (I/NP or I/GBNP).

Functionally, this submodule consists of 5 ports:

1. Formatting of incoming data. This is different for a software from I/FDP and local running ports.

2. Loading of formatted software through BIC interface to download local port.

3. Loading of remote node through a local port with interface as described in Section 5.9.2.5.

4. Maintain flow of incoming software and monitor buffers availability. This includes repeating of request of software from source.

5. Read BIC status flags at the end of the load and report it to MDL$REQUEST.

5.9.3.5 MDL$XPORTS

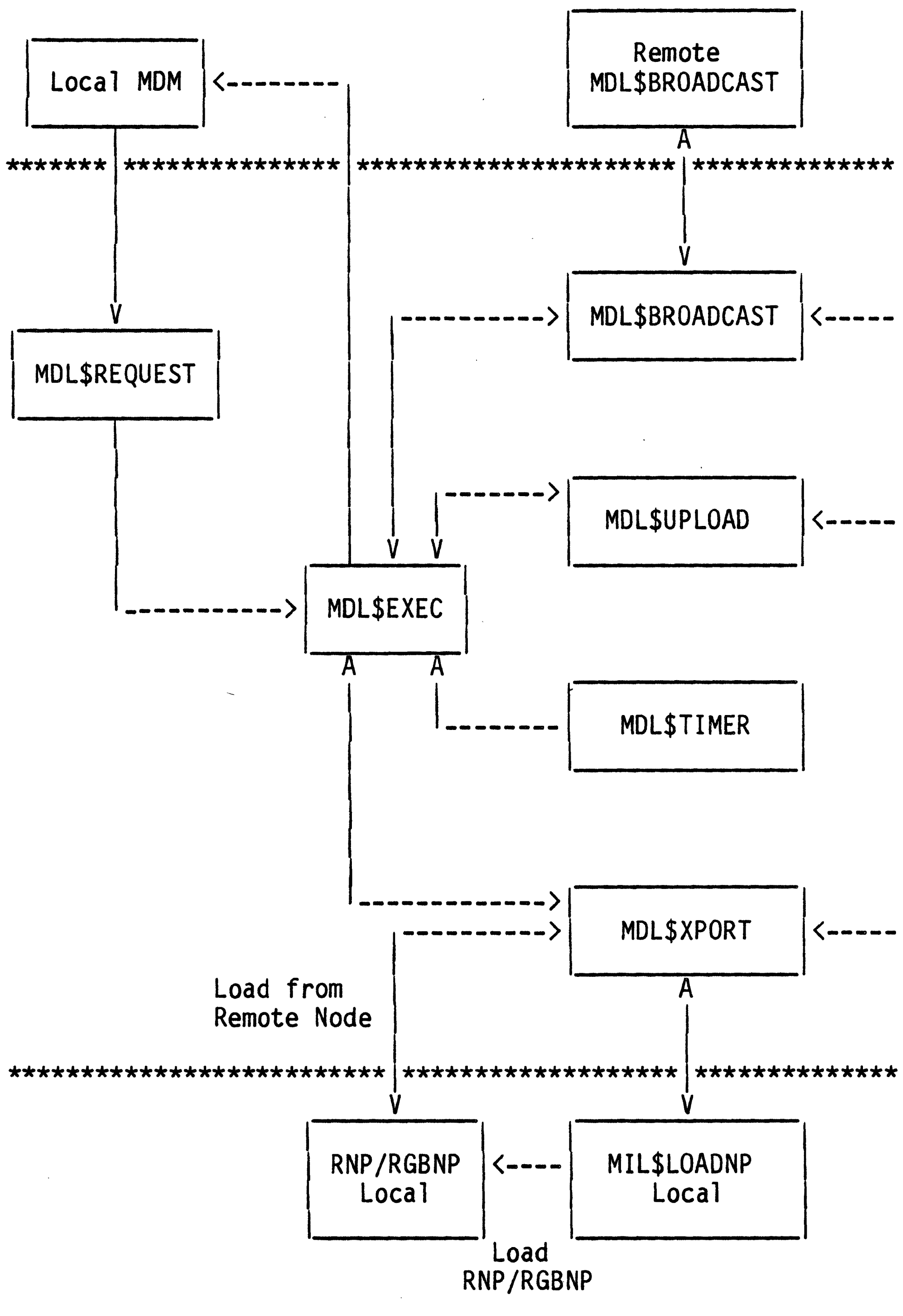
This submodule deals with the loading of I/NP, I/GBNP and I/FDP by their respective alternate methods. Requests to stop loading of the ports are also handled here, and there are two types: (1) stop if loading has not started, and (2) stop under all circumstances.

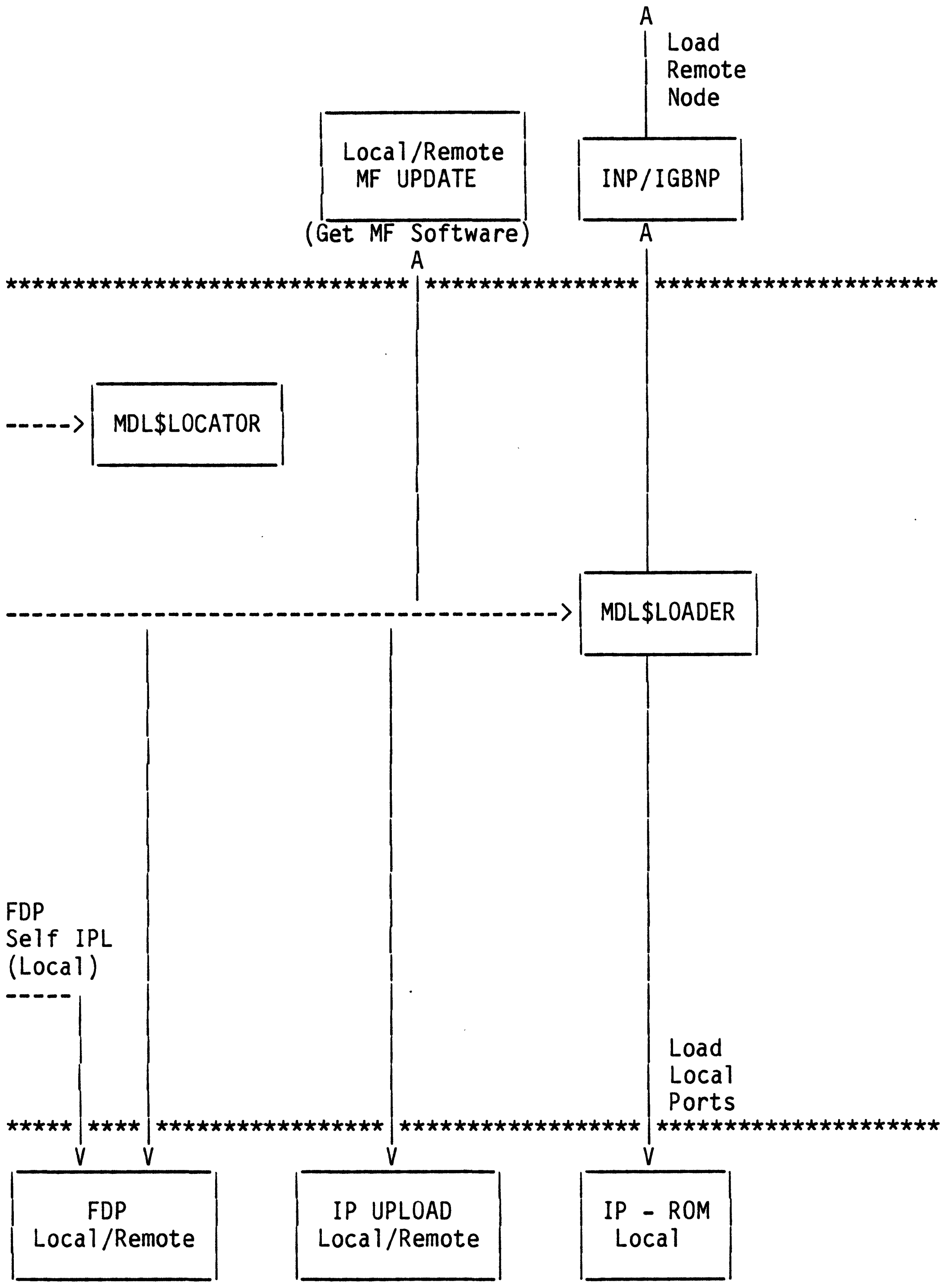
At the end of the load, a report is sent back to MDL$REQUEST.

Note that requests to stop loading may not be serviced immediately, since certain functions such as loading I/NP with dummy software have to be completed before interruptions.

5.9.3.6 MDL$TIMER

This is a schedule task that runs once every 2 seconds. Its only function is to send a packet to the other submodules to inform them of the elapse of 2 seconds.


Local MDM
Remote MDL$BROADCAST
MDL$BROADCAST
MDL$REQUEST
MDL$UPLOAD
MDL$EXEC
MDL$TIMER
MDL$XPORT
Load from Remote Node
RNP/RGBNP Local
MIL$LOADNP Local
Load RNP/RGBNP


Load Remote Node
Local/Remote MF UPDATE
(Get MF Software)
INP/IGBNP
MDL$LOCATOR
MDL$LOADER
FDP Self IPL (Local)
Load Local Ports
FDP Local/Remote
IP UPLOAD Local/Remote
IP - ROM Local

5.10 Mainframe Initialization Module

The Mainframe Initialization Module (MSI) is responsible for initializing the Mainframe hardware and operating system and performing various simple initial checks for system integrity.

5.10.1 MSI Entry Conditions

MSI is entered by all mainframe processors at MSI$RSTART:ENTRY from the ROM-resident mainframe IPL Module (MIL) after any system restart. If the restart was a software restart (initiated by MDL for a software inconsistency or by MSB for a configuration inconsistency), then configuration and software information is stored in Local Storage (the fields are described in the MSB subsection). All mainframe processors but one are halted immediately after entry into MSI, and one MSI from that point on runs only on that processor.

5.10.2 MSI RAM Initialization

MSI initializes various areas of RAM for use by the mainframe operating system and other mainframe software components. These are:

1. Lock byte area. Those lock bytes not allocated permanently either by MSI at runtime or in equate file OF$SYSLCK are allocated by MSI to a dynamic lock byte pool for use as needed during system operation. Lock bytes are described in the 6000 Logic Design Specification. Individual lock bytes are explained in the subsection describing the module and data structures using them.

2. Data spaces. MSI allocates the data spaces from RAM and initializes them as required by Mainframe Task Control. Data spaces are discussed in the subsection on MTC and in the 6000 Logic Design Specification.

3. Fixed RAM. Fixed blocks of RAM are allocated by MSI for tables and other fixed data structures used by various modules.

4. Dynamic buffers. The dynamic buffer pool used by MBM (see subsection on Mainframe Operating System) is initialized by MSI. MSI also sizes memory, verifies that data may be stored in the data area of memory, clears the data area to 0, and verifies that the lock byte are starts at address X'400'. If any error is found, the system halts with an appropriate message displayed on the front panel.

/

5.10.3 MSI Dynamic Routing System Initialization

The Mainframe Routing Manager (see subsection on Mainframe Path Management, Routing, and Congestion Control Module) provides two initialization subroutines that are called by MSI. These are MRM$INIT:NONDYNAMIC, called to initialilze of MRM's fixed data structures, and MRM$INIT:DYNAMIC, called to initialize all MRM's dynamic data structures. MRM$INIT:NONDYNAMIC uses an MSI fixed memory allocation routine (described later) and is therefore called before the free buffer pool is set up. MRM$INIT:DYNAMIC, on the other hand, uses the free buffer pool and is called after the free buffer pool has been initialized.

5.10.4 Node Configuration Parameter Initialization

MSI reads the node-related configuration parameters from CMEM into page 0 memory at the labels prefixed by OF$PGO:SY_. In addition, the active configuration number, either the previous active configuration stored in CMEM or the configuration passed by MSB in Local Storage, is both written to location OF$PGO:SY_ACNF and stored in the ACNF field in CMEM. MSI also sets up the CMEM map table from configuration information.

The CMEM node parameters and the map table are all described in the subsection on the Mainframe Configuration Module.

5.10.5 MSI System Boot Module Interface

MSI leaves complete configuration and software descriptive information in the system area fields prefixed by OF$SYS:MSB_RESTART_. These fields are discussed in the subsection on the Mainframe System Boot (MSB) Module.

MSI also calls MSB entries MSB$INIT:START and MSB$MAIN:START to initialize MSB (see subsection on MSB).

5.10.6 Mainframe Panel Control Module Initialization

MSI calls Mainframe Panel Control (MPC) entries MPC$INIT:START and MPC$INIT:SCAN to initialize MPC (see subsection on MPC).

5.10.7 MSI Scheduled and Batch Task Initialization

MSI constructs from templates the Scheduled Task table and the Batch Task table used by MTC for Scheduled and Batch Tasks, respectively. (See subsection on Mainframe Operating System.) These templates are defined in equate files EQ$SCHD and EQ$BATCH and reside in MSI$MAIN.

5.10.8 MSI Port Initialization

MSI provides these initialization functions:

1. Port Control Block (PCB) initialization - a PCB for each permanent (non-dynamic) configured port is allocated and initialized. This includes allocation and initialization of all associated data sub-structures (see PCB discussion in subsection on Mainframe Subsystem Data Structures).

2. Port loading - MSI calls Mainframe Downline Load (MDL) routine MDL$SUBS:LOADIP for each configured I/P. This initiates a load of each port. When and if the load completes a Load Complete addressed packet is sent to batch task MSI$INIT. This process is discussed more completely in the subsection on MDL.

3. Physical port initialization - MSI leaves the port BIC FIFO registers in their normal operational state. For further information about BIC FIFO we consult the subsections on the Mainframe Addressed Packet Module and the Mainframe Network Link Module.

5.10.9 MSI Machine Cycle Timing

MSI computes the number of M6800 cycles executed per millisecond by a single processor under approximately normal conditions. It saves this number in OF$PGO:MSM_CPS for use by the Mainframe Statistics and Monitoring (MSM) Module (see subsection on MSM).

5.10.10 Boot Complete System Report

When all the mainframe software has been successfully initialized a Boot Complete system report is sent. (System reports are described in the subsection on MSM.) This system report has these parameters:

1. Configuration
2. Software source node
3. Software revision
4. Software release
5. Software source port

The system report code is EQ$SYSRPT:BOOTCOMP.

5.10.11 MSI Subroutines

MSI submodule MSI$SUBS contains subroutines used both by MSI and by external mainframe modules. Those subroutines which may be used by external modules are listed here.

1. MSI$SUBS:AFMS

   This routine is called to allocate 255 or less bytes of free memory from the free memory list. It may only be called by initialization routines called by MSI$MAIN before the free buffer pool has been allocated.

   ```
   On Entry  B-reg = Number of bytes required

   On Return X-reg = Pointer to available memory
             A-reg = Destroyed (unless error)
             B-reg = Destroyed
   ```

   If the block cannot be allocated, the error code for "OUT OF MEMORY' is loaded into A-reg and control returned to the caller with CC:C = 1.

2. MSI$SUBS:AFML

   This routine is called to allocate more than 255 bytes of free memory from the free memory list. It uses MSI$SUBS:AFMS to do the actual allocation and may not be called after free buffer pool allocation.

   ```
   On Entry  A-reg = MSB of number of bytes required
             B-reg = LSB of number of bytes required

   On Return X-reg = Pointer to available memory
             A-reg = Destroyed (unless error)
             B-reg = Destroyed
   ```

   If the block cannot be allocated, the error code for "OUT OF MEMORY" is loaded into A-reg and control returned to the caller with CC:C = 1.

3. MSI$SUBS:ROR

   This routine is called to read one byte from the offline ROM residing on the options card (see subsection on Mainframe Configuration Module).

   ```
   On Entry  B-reg = Offset into ROM
   ```

```
On Return A-reg = Unchanged
          B-reg = Value from ROM
          X-reg = Unchanged
```

Note: This routine, at the present time, does not access the ROM. It reads the bytes from a table in RAM.

5.11 Multi-Threaded Port Control Module

The Mainframe Multi-threaded Port Control Module (MMT) provides the software interface between the mainframe software and the multi-threaded terminal ports. MMT multiplexes and demultiplexes the data streams for many virtual ports through the Bus Interface Chip (BIC) interface with a single multi-threaded I/TP.

MMT is designed to make the virtual port interface with external mainframe modules resemble as closely as possible the single-threaded interface so that virtual ports and single-threaded ports may be handled by common logic.

5.11.1 MMT Port Control Block Interface

Port Control Blocks (PCBs - see sections on Mainframe Subsystem Data Structures) are the main interface between MMT and other mainframe software modules. Each physical multi-threaded Terminal Port has a stripped-down PCB and each Virtual Port has a PCB which is functionally equivalent, from the point of view of other mainframe software modules, to the PCB of a single-threaded terminal port.

5.11.1.1 Multi-Threaded Terminal Port PCB

The PCB associated with the Multi-Threaded Terminal Port is needed for functions which involve the port as a whole rather than an individual virtual port residing in the physical port. Such functions are implemented by means of addressed packets sent to the physical port address. The Multi-Threaded Terminal Port PCB therefore has as its only substructure the Packet Data Substructure (see subsection on Subsystem Data Structures). It should be noted that the inbound and outbound addressed packet queues for the port are located in this PCB, not in the individual VP PCB's (described below).

5.11.1.2 Virtual Port (VP) PCB

As already noted, the VP PCB is, from the point of view of external mainframe modules, functionally similar to the PCB of a single-threaded I/TP.

Since the VP has no physical BIC FIFOs associated with it, inbound and outbound data byte queues take their place. These byte queues may be located using pointers in the PCB Slot Data Substructure. All data belonging in the user data stream (that is, all data which would otherwise flow through the data BIC) goes through the proper data byte queue between the MMT and any external mainframe module.

5.11.2 MMT BIC Interface

The Bus Interface Chip (BIC) data FIFOs are the interface between MMT and the Multi-Threaded Port software. The data format used is the Multi-Threaded High-Speed Data Interface described in Section 3 of the System Software Specification. Data for a particular thread in this format is preceded by a one-byte VP address and terminated by an ICS called the Multi-Threaded End-of-Slot (MTEOS).

The inbound and outbound data FIFO Reader's and Sender's Flags are used for control functions: The reader of a data FIFO clears the FIFO's Sender's Flag and sets the Reader's Flag to signal to the writer that it may begin sending data; the writer sets the Sender's Flag and clears the Reader's Flag to signal to the slot reader that it has completed its transmission of data slots.

MMT, running every 25 milliseconds, uses this mechanism to control the timing of inbound (port to mainframe) data transmission so that data is not sent inbound until MMT is active and ready to read data. The timing and control sequences involved in slot transmission in both directions is illustrated in the following figures.

Data Flow from Multi-Threaded Port to Mainframe
Over BIC Inbound Data FIFO

MAINFRAME	PORT
Sets Reader's Flag and Clears Sender's Flag when ready to read data	
	<Slot 1 .... Slot N
	Sets Sender's Flag, clears Reader's Flag when all data is in FIFO.

.
.
.
.

25 Millisecond delay

Sets Reader's Flag and
Clears Sender's Flag
when ready for next
batch of slots.

.
.
.

etc.

Data Flow from Mainframe to Multi-Threaded Port
Over BIC Outbound Data FIFO

MAINFRAME PORT

Slot N .... Slot 1 >

Sets Sender's Flag
Clears Reader's Flag

Sets Reader's Flag, clears Sender's Flag when done reading

.
.
.
.

25 Millisecond delay

Slot N .... Slot 1 >

.
.
.

etc.

5.11.3 Operational Overview of MMT

MMT is a low-priority task activated every 25 milliseconds at entry point MMT$MAIN:ENTRY. There it activates, if possible, one of at most three concurrent scanning tasks. The scanning task first reads the inbound BIC data FIFO for each physical multi-threaded port and fills all the received data byte queues associated with that port. Next it again goes through the list of physical multi-threaded ports and, at each port, multiplexes all the data stored in the transmit data byte queues of VP's associated with it into the outbound BIC data FIFO.

To aid in the above processing, all multi-threaded port VP PCBs are linked together by means of the link field in the PCB main data structure, and access to any single port by MMT tasks is interlocked through the USER lockbyte. The slot data substructures for VP PCBs associated with each physical port PCB are linked by means of the VPLINK fields in each slot data substructure and in the physical PCB. This linking together of PCBs is done at system initialization time in subroutine MMT$INIT.

The data format used for data sent between port software and MMT over a multiplexed BIC is the Multi-Threaded High-Speed Data Interface (MTHSDI) described in Section 3. This interface is designed so that MMT may handle data for any given VP transparently, scanning for no control characters other than the slot terminator. It should be noted that MTHSDI provides no "end of data" signal. As a result the MMT scanning tasks must assume that when an inbound BIC FIFO is empty, the port has sent all the data it desires to send. Because of this, the port must be able to keep up with the mainframe the vast majority of the time or data will back up in the port. Also, to maintain mainframe efficiency, the port should almost always be able to empty the outbound FIFO fast enough to prevent it from becoming full.

5.12 Mainframe Diagnostics Monitoring and Physical Port Control Module

5.12.1 Introduction

The MDM performs the following offline I/P management and failure monitoring functions:

1. Through an addressed packet user interface, any port may be brought online or taken offline and any desired software may be loaded to any offline port, any adjacent I/NP, or any adjacent 6000 Mainframe. (The term 'offline' is used in this section to refer to a port which is not available for its normal port functions such as sending and receiving addressed packets, establishing paths, etc.) The addressed packet user interface also provides various control and monitoring features to be used in conjunction with these functions.

2. Automatic Loading - MDM is responsible for supervising the Automatic Loading process, the means by which an adjacent node or a local port may be tested (if required) and supplied with the required system software of the active revision and release level without operator intervention.

3. Failure Monitoring - MDM considers an I/P to have failed if no packet is received from the port for a 12-second period. When this happens, the port is automatically taken offline by MDM and the I/CTP operator is notified through a system report. Depending on the sort of port failure, MDM may then attempt to bring it back online. MDM also runs various simple online mainframe tests and sends system reports when errors are found.

4. System Error Handling - MDM entry point MDM$SYSERR:CRASH is called whenever a fatal system error occurs. Upon entry, the A register contains the appropriate error code, as defined in the file EQ$SYSERR.

5. Mainframe Code Space Interface - MDM submodule MDM$UPLOAD provides the only interface by which system components external to the local mainframe may access the Mainframe RAM code space. The interface is through addressed packets, with a destination (module dispatch number) of EQ$MDT:MDM_UPLOAD or queued to batch task number EQ$BATCH:MDM_UPLOAD. The packet formats are identical to those of I/P module IP$UPLOAD and will not be described further here.

MDM's most important function is to provide a sort of gateway by which local and remote user interface modules may interface with ports not actively running system software. The physical BIC interface with such ports is handled either directly by MDM or by the Mainframe Downline Load Module (MDL) in response to commands from its local MDM.

The means by which this gateway is implemented is the MDM addressed packet user interface, mentioned in 1 above. The term 'user' in this section is taken to mean any module invoking this gateway function through the addressed packet user interface. Such modules include:

Mainframe System Initialization Module (MSI) - Submodule MSI$INIT uses MDM to initialize, test, and load a port when it is initially brought up.

I/CTP - The I/CTP, under operator control, may use any of the functions provided by the MDM addressed packet user interface.

Mainframe Panel Control Module (MPC) - Module MPC allows an operator, through the mainframe front panel, to perform a subset of the functions provided by the addressed packet user interface.

5.12.2 Detailed Specification of the Addressed Packet User Interface

The MDM user interface includes commands to be sent in addressed packets from the user to MDM and various addressed packet messages to be sent from MDM back to the user during the execution of the command. Packets are sent to MDM by queueing them to the batch task number "EQ$BATCH:MDM_AP". The entry point for this batch task is "MDM$COMMAND_ENTRY", and the dispatch number is "EQ$MDT:MDM_AP".

For every command executed by MDM, at least one 'MDM Report' is sent back to the user. The MDM report format is used both for diagnostics packets and for sending MDM System Reports. It will be discussed in a later subsection. When an MDM command terminates at a port, the user is always notified with an MDM Report, but if the user is not local, it cannot be guaranteed that the message will arrive at its destination (although it is extremely rare for this not to happen).

MDM command packets contain both fixed-length and variable-length fields. Offsets for fields are defined in file OF$MDM and are prefixed by OF$MDM:AP_. The first parameter is at offset OF$MDM:AP_CC in all command packets and contains the MDM command code for the particular command. Other fields depend on the command code.

The following commands are supported:

Run Local Port (command code EQ$MDM:CC_RLP).

Run Remote Node (command code EQ$MDM:CC_RRN).

Initiate Automatic Load (command code EQ$MDM:CC_IAL).

Port Failure (command code EQ$MDM:CC_PORTFAIL).

Check Ports for Diagnostics Packets (command code EQ$MDM:CC_CHECK_DIAGS).

The following subsections detail the packet formats required by each.

5.12.2.1 Run Local Port Command Packet Format

This command is used to perform all the basic MDM local port functions. It can be used to perform these functions on a list of ports or on a single port. The command packet contains the following fields:

Function (at offset OF$MDM:AP_FUNCTION) - This byte specifies the function(s) to be performed on the port(s) contained in the Port List field (described later). Each function corresponds to one bit in the byte and most combinations may be specified. Where more than one function is specified, they are performed in the order below:

Reset - If bit EQ$MDM:RESET is set, and the port is on-line, it is taken offline and certain 'cleanup' functions, described in the subsection on the local port interface, are performed. If it is in diagnostic monitoring mode, the monitoring is cancelled. If a port is a multi-threaded port, then the cleanup procedures are performed for each virtual port that is linked to the multi-threaded port.

ROM Diagnostics - If bit EQ$MDM:ROM_DIAG is set, several stages of diagnostics are performed on the port. If all stages are successful, the port can be loaded with software. If any stage fails, a system report will be sent with an error code that is unique to the stage that failed (see EQ$MDM:EC_), and the port is left in a state that requires a RESET function before MDM will do further processing. The stages of ROM Diagnostics are:

1. Basic diagnostics for the port hardware. This test is initiated by giving the port a "Reset Ø", and takes 8 seconds to complete. MDM will expect the "Port Bit" to be set in the port's Packet Bic Status register if the test was successful.

2. Parity diagnostics. This test is initiated by giving the port a "Reset 3". MDM expects the Packet Bic Status register to equal a value of Hex 84 if the test was successful. The test is followed by a Reset Ø, Reset 1, sequence to leave the port in a non-error state.

3. Basic Bic test. This test checks that the Bic is operational on a basic level, and can be used to load Software into the port.

Load - If bit EQ$MDM:LOAD is set, the port is loaded with the software in the software field, assuming the port is of loadable type, and in a loadable state.

Set Start Address - If bit EQ$MDM:SET_START_ADDRESS is set, then the address in the field START is taken to be the start address for the code currently loaded in all of the referenced ports. This function can be used to modify the start address from the file (if any) down-line loaded with the LOAD function.

Monitor - If bit EQ$MDM:MONITOR is set, then the port is started (assuming a start address has been specified or was passed in the load block), the parameter list is passed to it in a Diagnostic Packet, and the port is monitored every 30 seconds for diagnostic packets which are then passed to the user as will be described later. (Diagnostic Packets are addressed packets passed between port diagnostics and the mainframe, whether or not the mainframe is running D814 system software. They are described in the section on Subsystem Interfaces). The port is now considered to be in 'diagnostic monitoring mode'. The command will not terminate (meaning the port leaves diagnostic monitoring mode) until a Diagnostic Packet with type equal to "termination" is received, or until an MDM Reset command cancels it.

Set Online - If bit EQ$MDM:ONLINE is set, then MDM will execute the on-line procedures, described in the 'Local Port Interfaces' section, after the port has been successfully loaded with operating software.

(Note: The 'monitor' and 'set on-line' functions are inconsistent with each other. If both these functions are specified, only monitor is performed. If the reset function is specified, no other function except ROM diagnostics can be performed unless the load function is also specified.)

Software (at offset OF$MDM:AP_SOFTWARE) - 8-byte field identifying the file to be loaded, if the 'load' bit was set in the function field. The field must be in one of these formats:

1. If standard port operating software is to be loaded, it must contain a 0 in the first byte.

2. If a standard startup diagnostic is to be loaded, it must contain a X'FF' in the first byte.

3. If the file to be loaded is neither the standard startup diagnostic nor the standard operating software, the eight byte file name in ASCII must be specified.

Monitoring Flags (at offset OF$MDM:AP_MFLAGS) - This contains bits which specify various options for diagnostic monitoring if the monitor bit was set in the Function field. If bit EQ$MDM:MFLAGS_UPDATE is set, then update, as well as Termination, Diagnostic Packets are sent to the user; otherwise, only the Termination Packet is sent. The port is polled every 30 seconds to see if there are any packets to be sent. If bit EQ$MDM:MFLAGS_ASCII is set, then the ASCII portion of the Diagnostics Packet will be forwarded to the user. If the bit 'EQ$MDM:MFLAGS_BINARY' is set, then the binary portion will be forwarded to the user. The user can select either, both, or neither of these fields to be forwarded. (See Subsystem Intérfaces for a description of the Diagnostic Packet format.)

Start (at offset OF$MDM:AP_START) - This is the 2-byte start address to be set if the 'Set Start Address' function was specified. If 0, the code in the port will not be allowed to run until a non-zero start address is set.

Port List (at offset OF$MDM:AP_PORT_LIST) - This is a variable-length list of ports terminated either by the end of the packet or by a 0 delimitor. The functions specified in the FUNCTION field are performed on all the ports in this list. If more than one port is listed, the command is handled as if it were really a set of identical command packets each of which involved one and only one port. Therefore, if a completion message is sent to the user, for example, when the last function of a command is successfully completed, one such message would be sent for every port in the list. The port list can contain different types of ports, but if standard software is specified, ports requiring identical software will be grouped together into separate load request packets. Load request packets are sent to the Down Line Loader (MDL), where the load operation is actually performed, unless the port is a floppy disk or a floppy disk emulator port, and standard diagnostics software is requested. MDM handles the down load operation itself for those two cases.

Parameter List (optionally follows the Port List) - If diagnostic monitoring was specified in the FUNCTION field, the parameter list is sent to the port (preceded by the number of parameters) after the port's diagnostic software has been successfully loaded and sent a start code.

5.12.2.2 Run Remote Node Command Packet Format

This command is used to manipulate a remote adjacent node through an operational local I/NP. The command packet contains the following fields:

Function (at offset OF$MDM:AP_FUNCTION) - This byte specifies the function(s) to be performed on the adjacent node, in a manner similar to that of the FUNCTION parameter in the Run Local Port command. The following functions are supported:

Hard Boot - If bit EQ$MDM:HARDBOOT is set, then the remote adjacent INP is sent the NP boot sequence causing it to restart, and then accept any software it is sent from this node. Once a node is hard booted it will only accept software from the I/NP which received the NP boot sequence.

Load Mainframe - If bit EQ$MDM:LOAD_MF is sent, then the remote adjacent mainframe is to be loaded and started executing the software specified in the software field. To be sure that the remote mainframe will load this software, as opposed to software received over some other link adjacent to it, the Hard Boot function should also be specified.

This bit need only be specified if the user wants to specify the software to be loaded. Otherwise, MDM will respond to help messages sent by the remote node and will load whatever software it requests.

MDM will send an immediate acknowledge back to the origin of Run Remote Node command. The acknowledge is the original command, with the source and destination exchanged.

Software (at offset OF$MDM:AP_SOFTWARE) - 8-byte field identifying the name of the file to be loaded, if the 'load mainframe' bit is set in the Function field. The file is specified as follows:

1. If standard Mainframe operating software is to be loaded, it must contain X'0100' in the first two bytes.

2. If the standard Mainframe diagnostic package is to be loaded, it must contain X'01FF' in the first two bytes.

3. If the file to be loaded is neither the standard startup diagnostic nor the standard operating software, the eight byte file name in ASCII must be specified.

Port Number (at offset OF$MDM:AP_PORT_LIST) - The command must contain a single port number through which the operations will be performed. The port must be an up and running I/NP.

5.12.2.3 Initiate Automatic Load Command Packet Format

This command causes MDM to start the standard load sequence for local ports. If a port is taken offline for some reason, this command may be used to bring it back online. MDM will then run diagnostic if indicated, load the port, and set it online. All this will happen automatically without any further user intervention. When MDM processes this command, it immediately sends a response back to the user to indicate that the command has been successfully received. No message is sent by MDM to the user when loading is completed, although a system report is normally sent when any node or port comes up.

The diagnostics enable bit (EQ$MDM:IAP_DIAG) inside the function byte at offset OF$MDM:AP_FUNCTION causes MDM to generate a "Run Local Port" command packet and queue it to the Command Processor batch task. The packet contains:

1. The Reset Function
2. The ROM Diagnostics Function
3. The Load Function
4. The Monitor Function
5. The Codes for Standard Diagnostics Software
6. No Monitoring Flags

If an error is encountered, a systems report is sent. Otherwise, when a successful diagnostics termination packet is received by MDM, it generates another "Run Local Port" command packet. This time the packet contains:

1. The Load Function
2. The ON-LINE Function
3. The Codes for Standard Operating Software

If the diagnostics enable bit is not set in the "Initiate Automatic Load" command, then the MDM operation is identical to a "Run Local Port" command with the following contents:

1. The Reset Function
2. The ROM Diagnostics Function
3. The Load Function
4. The ON-LINE Function
5. The Codes for Standard Operating Software

Beginning at offset OF$MDM:AP_PORT_LIST, the command packet contains a list of the port addresses to be loaded.

5.12.2.4 Declare Port Failure

If an MDM user detects a port failure, this command can be used to inform MDM of the failure immediately rather than waiting for MDM to time 12 seconds of inactivity on the port before the failure is declared. The command will cause MDM to look at the port's BIC status for a parity error, and then send the appropriate system report. If the failure was due to the first parity encountered in the last 30 minutes, an "Initiate Automatic Load" command will be generated for the port entered in the command packet at the offset OF$MDM:AP_PORT_LIST. The automatic load command will have the diagnostic function selected.

5.12.3 MDM Interface With Mainframe Downline Load Module (MDL)

As already noted, MDM uses MDL to load software when required. The interface with MDL is described in detail in the subsection on that module. Only the important features of that interface will be summarized here.

The interface with MDL is through addressed packet format messages, queued to the MDL batch task number "EQ$BATCH:MDL_REQUEST". The command code in the packet contains one of the following:

Load Local Port - MDM uses this MDL command to locate and down load formatted load blocks to any port type except floppy disks or floppy emulators, that are loading standard diagnostics software. These load block types have to be down loaded from the MDM submodule 'DOWNLOAD'.

Load Remote Node - Causes MDL to initiate loading of a remote mainframe through a local operational I/NP. Any load already in progress is cancelled.

Load Remote Port - Causes MDL to initiate loading of a remote NP, through a local operational I/NP.

Abort Load - Cause MDL to terminate a load in progress.

MDL's responsibility is strictly to get a port loaded with the proper software. MDL does not start the port or time out if the port cannot be loaded. Further, MDL interfaces with no system component other than the local MDM.

MDL sends a message back to MDM whenever a load of a port listed in one of the Load Port commands above is completed, whether or not the load was successful. Possible reasons for a load failure are port failure and an invalid port address. A load cannot fail because the software could not be found, since MDL in such a case retries indefinitely.

MDL also informs MDM whenever an Abort Load command is complete.

5.12.4 MDM Reports and System Reports

MDM system reports as well as messages sent to users of the MDM addressed packet user interface share a common format and are known collectively as MDM reports. This format allows MDM to build an error message, for example, before deciding whether it is to go to a user or be sent as a system report. All system report fields (defined in file OF$RPT) are present in MDM Reports. The system report code field OF$RPT:CODE always contains the system report code EQ$SYSRPT:MDM, which allows the report to be sent as a system report with no reformatting. Fields in the data area of the report are defined by offsets in file OF$MDM as follows:

OF$MDM:REPORT_PORT - Port to which the report refers, 0 if Mainframe.

OF$MDM:REPORT_TYPE - Contains the type of report:

EQ$MDM:REPORT_DIAG_UPDATE - Report contains contents of Update Diagnostic Packet.

EQ$MDM:REPORT_DIAG_TERM - Report contains contents of Termination Diagnostic Packet.

EQ$MDM:REPORT_FAILURE - A test run by MDM itself has failed, for example the BIC loopback. Since the MDM report format is used in all MDM system reports, one of the errors discussed later under Failure Monitoring could for example be reported in this packet.

OF$MDM:REPORT_EC - Error code. The meaning of error codes depends on the report type. If the report is a Termination or Update message, then the error code is the Summary Status Byte from the Diagnostic Packet (a value of zero implies no error). Otherwise, it is defined in file EQ$MDM and prefixed by EQ$MDM:EC_.

OF$MDM:REPORT_BIN_LEN - This contains the number of bytes of binary data that follow this byte. The use of the binary data is defined by the diagnostic software.

String of ASCII Characters - Following the list of binary data, starts the list of ASCII characters. The length of the packet is used to locate the last character.

Undeliverable MDM system reports are returned to MDM at entry point MDM$SYSRPT:ENTRY by the addressed packet router. These are then rerouted to the Mainframe Panel Control Module for display on the front panel.

5.12.5 MDM Interfaces Used in Providing Software Over a Link to an Unloaded I/NP

This subsection describes the MDM interfaces involved in loading software to a remote adjacent node (referred to as the 'loading node') which is running the 'ROM NP', a special bootstrap module loaded through a call to the Mainframe IPL Module (MIL).

A loading node communicates with an adjacent network node using HELP messages, described in the sections dealing with I/NP and ROM NP link protocol sections. The HELP message is used both to bring the link up and to send parameters from the loading node to the node running system software. When a HELP message is first received by an I/NP, MNL is informed through the Link Active addressed packet that the link (which it had previously considered to be down) is now operational. This message contains all the HELP message's parameters including a possibly empty software specification field. Since the HELP is otherwise treated by the I/NP as an AMSTR or ASEC message (described in the subsection on I/NP software), the link is now up and data may be sent. MNL calls MDM subroutine MDM$LINK:UP as follows to pass it the HELP message parameters in its received link Active message:

Entry Conditions:

X-register points to the link Active message received by MNL, containing the HELP message parameters. Before calling this routine MNL must have determined from the message parameters that a HELP message rather than an ordinary AMSTR/ASEC message was received.

On Exit:

All registers and data space fields are destroyed. The message byte file has been deleted.

Function - The subroutine sets the 'IPL in Progress' bit in the I/NP PCB's status lockbyte, which is a signal to MNL that the link is in use by MDM. What MDM does depends on the HELP message's parameters:

If the 'type' parameters is 'diagnostic result', then it is meant only to supply information to the ICTP operator. A system report is sent with the message parameters in the MDM Report format discussed earlier, and nothing more happens.

If the software revision and release are specified and are different from the active software, a system report is also sent.

If the restart type is NP-boot, a system report is sent.

If a fatal error is specified as the error code, a system report is similarly sent.

If the 'type' parameter is 'load', the error code has its high order bit set (either no error or non-fatal error), the restart type is 'software restart' or 'power-up', and the software revision and release are consistent with the active software, then software is to be downline loaded without operator intervention. If a specific software module is requested, then that module is downline loaded to either the remote Mainframe or the remote I/NP, as indicated by the M and N bits in the HELP message parameter byte. If no specific module is requested but the mainframe is not loaded (this is indicated by the L bit in the Help Parameters field of the HELP message), then the Mainframe software is downline loaded. If the error code is a non-fatal error (higher than X'7F' but not X'FF'), then a system report is sent but the proper software is also loaded as above.

Once MDM has been informed that a link with a loading node has been brought up, no mainframe module other than MDM or MDL (Mainframe Downline Load Module) is allowed to access the I/NP's BIC data FIFO until MDM has been informed that the same link has gone down. This is guaranteed through the 'IPL in Progress' bit in the I/NP PCB. To implement the interlock, the following subroutine must be called whenever MNL receives a link Failure message from an I/NP or whenever an I/NP is declared to be down:

Entry Point:

MDM$LINKDOWN

Calling Conditions:

A-Register - Contains address of the I/NP

On Exit:

All registers and data space fields are destroyed and the 'IPL in Progress' bit in the I/NP's status lockbyte is cleared.

Function - The subroutine does nothing if the 'IPL in Progress' bit is not set. Otherwise MDM terminates any load in progress using the I/NP. This is done by sending MDL an 'abort' command and waiting for the response. Before returning, the IPL in Progress bit is cleared.

It should be noted that this routine and MDM$LINKUP are the only routines which modify the IPL in Progress bit for an I/NP.

5.12.6 MDM Local Port Interface

This subsection describes the interfaces needed for MDM's local port functions.

5.12.6.1 Port Control Block (PCB) - Offline Port Cleanup and MDM Local Storage

This subsection describes those MDM local port interfaces implemented through the PCB of the IP involved. The PCB of a local IP is used by MDM for local storage of MDM data relating to the port and to implement required synchronization for the 'cleanup' functions which must be performed by various mainframe modules when a port is to be brought off-line by MDM. The remainder of this subsection discusses the above functions.

Each I/P PCB contains in its MAIN data substructure a field, OF$PCB:MAIN_DDSS, which contains a possibly zero pointer to a Diagnostic Data Substructure (DDSS). The DDSS is a single buffer which is allocated and delocated as needed by MDM. It is not referenced by external modules and is not allocated for a port which is up and running D814 port system software. The DDSS is used by MDM for storage of data associated with an off-line I/P.

Each I/P also has a STATUS lockbyte which is used for synchronization with other Mainframe modules when bringing a port on-line or off-line. An on-line port is assumed to be running system software and may perform all of its normal system functions, while an off-line port may only interface with the rest of the network through MDM or MDL. The STATUS lockbyte has a DOWN bit which is set by MDM whenever a port is taken offline. Also in the STATUS lockbyte are 'busy' bits for each system module for which cleanup must be done when an I/P is taken offline. The mask for a module's busy bit is EQ$PCB:LOCK_STATUS:xxx_BUSY where 'xxx' is the module name. A module sets its busy bit whenever it is using the port BIC FIFO's (this may be for an indefinite period) and MDM may not do physical IO to the port while any busy bits are set. On the other hand, no module may set a busy bit while its DOWN bit is set by MDM. Each module having a busy bit provides to MDM a cleanup subroutine which may be called after the DOWN bit has been set to do whatever cleanup the port must do and turn off the busy bit. A module's cleanup subroutine has entry point xxx$CLEANUP:ENTRY where 'xxx' is again the module name. Therefore, to take a port off-line, MDM must do the following:

1. Set the port's DOWN bit.

2. For each busy bit which is set, call the proper module's cleanup subroutine. There are busy bits and cleanup subroutines for modules MNL, MAP, MMT, and MPM.

3. Wait for all busy bits to clear.

Once this has been done, MDM may do whatever it wants with the port.

MDM will put a port on-line after it receives a successful load response from MDL, and the ONLINE function was specified. The steps are:

1. The down bit in the port's Status lockbyte is reset.

2. The down bit in the port's Packet lockbyte is reset.

3. The pointer to the diagnostics substructure in the port's PCB is zeroed, and the buffer is returned to the operating system.

4. A start code (Hex 55) is sent to the port's BIC Ø FIFO.

5. When the first Wakeup packet is received from the port, the active configuration is put in Byte 9 of the Packet, and the packet is queued to the Packet Transmitter. The Senders flags are set in the BIC status registers to enable the port to read the packet.

The two-bit TIMER field in the PCB Packet lockbyte is used in port failure monitoring. It contains the number of 6-second monitoring periods since an addressed packet was last received from a port. The addressed packet router clears it whenever it receives a packet from the port. Every six seconds a scheduled MDM task increments the TIMER field and declares the port to be dead if it is incremented to 2. Since a port is required to send a packet at least every ten seconds, an operational port cannot be declared dead. When MDM declares a port dead, it is taken offline and a system report sent.

5.12.7 Failure Monitoring

MDM periodically runs various port and system diagnostic tests. When an error is found an attempt is made to report it to the network operator through a system report. If the system report packet is returned, it is rerouted to the Mainframe front panel.

The following diagnostic procedures are run by MDM:

Port Failure Monitoring - Any physical port from which no addressed packet is received for roughly twelve seconds is considered to have failed. When MDM discovers such a timeout, it takes the port offline. After the port is taken off-line, MDM does a reset-3 to the port data BIC and reads data from the inbound data FIFO. This data, which must be less than 128 bytes, is placed in a MDM Report packet and sent as a system report with either a 'port time-out' error code (if there was no parity error) or a 'RAM parity failure' error. If the error was a parity error and if the port had not failed in the previous 30 minutes, an automatic load of the port is then initiated.

RAM Test - A memory test is run continuously at priority 0, the lowest Mainframe priority level. This test consists of a free buffer memory test and a code space checksum test. If a buffer is found to contain a bad memory location, a system report containing the bad address is sent. If the code checksum at location OF$SYS:MIL_CHECKSUM (a 2-byte checksum computed by adding bytes with wrap-around carry) does not match the checksum computed by MDM an error also occurs.

Lockbyte Test - Every three seconds all the lockbytes except the debugger lockbyte are tested. If any lockbyte is found to be clear for more than 1000 microcycles, an error occurs and a system report containing the lockbyte address is sent.

Link Monitoring - Every 30 minutes, starting 30 minutes after system startup, an MDM system report is sent for every I/NP whose link is down. A link is considered down if the Framing Lost bit is set in the I/NP's STATUS lockbyte.

CMEM Test - Every 30 minutes Configuration Memory is checksummed in the same manner as code space. If the checksum does not match the checksum stored in the first two bytes of CMEM, an error system report is sent. CMEM is read using routines provided by the Mainframe Configuration Manager Module (MCM).

5.12.8 System Errors

Fatal errors in any Mainframe system module are handled by calling MDM routine MDM$SYSERR:CRASH with the appropriate error code, defined in file EQ$SYSERR, in the calling A register. MDM$SYSERR:CRASH first checks if the Debugger Port is plugged in. (The debugger port is always in slot 0, which is an otherwise invalid port address.) If the Debugger Port is there, the debugger is invoked. Otherwise a 6000 software restart is done and the error code is passed to the IPL ROM as described in the Firmware Section on the Mainframe IPL Module.

5.13 Mainframe Subsystem Data Structures

This subsection describes all mainframe data structures not attached specifically to any individual mainframe module and, therefore, not described elsewhere.

5.13.1 Port Directory

Addresses X'200' to X'3FF' contain the mainframe Port Directory. The address in RAM of the Port Control Block (PCB) for port number _i_ will be found at offset 2*i into the Port Directory if such a PCB exists. If there is no port _i_, then the entry for it will be X'0000'.

The Port Directory is interlocked through lockbyte OF$SYSLCK:PORTDIR and may be accessed through mainframe utility MUT$PCB.

5.13.2 Port Control Blocks

A Port Control Block (PCB) is maintained in the mainframe for each user data source or destination. These include all physical IP's as well as virtual ports (associated with an individual data stream through a multiplexed I/TP) and XP's (associated with the data stream for a user path at an intermediate node).

Fixed PCB's are created by the Mainframe System Initialization Module (MSI) and dynamic PCB's are created by the Mainframe Path Management, Routing, and Congestion Control Module (MPMRCCM). The modules having read/write access to PCB's are MPMRCCM, the Mainframe Addressed Packet Module (MAP), the Mainframe Network Link Module (MNL), and the Mainframe Statistics and Monitoring Module (MSM). In addition, various mainframe modules read static information from the PCB's.

The structure of a PCB depends on the type of port with which it is associated. Each PCB has a top level main data structure containing some fields common to all PCB's as well as some fields dependent on the type of port. The PCB has various data substructures depending on the type of port. In general, each substructure contains data used primarily by one specific module or submodule. The structure of the PCB is described in the following subsections.

5.13.2.1 Main Data Structure (All Offsets are Prefixed by OF$PCB:MAIN_)

Fields common to all port types:

TYPE	- Port type (high order nibble) and subtype (low order nibble)
PADR	- Port address
LOCKS	- Pointer to port's lockbyte area. Each port has specific lock bytes associated with it for synchronization between different tasks accessing the PCB.

Fields only found in I/NP PCB's:

XMT	- Pointer to MNL Transmit Data Substructure
RCV	- Pointer to MNL Receiver Data Substructure
NPLINK	- Link to next I/NP PCB. 0 if last PCB in linked list. It should be noted that this linked list is unchanging once normal system operation has begun.

Fields only found in I/TP, XP, and VP PCB's:

SLOT	- Pointer to Slot Data Substructure (used by MNL$XMT)
PATH	- Pointer to Path Data Substructure (used by MPMRCCM)

Fields common to all PCB's except LNP, VP and XP:

PACKET	- Pointer to Packet Data Substructure (used by MAP)
DDSS	- Pointer to Diagnostic Data Substructure used by Mainframe Diagnostic and Physical Port Control Module (MDM) for scratch storage. This field is set to 0 by MDM when the substructure is unallocated.
MDM_LOAD	- MDM's load control flags (described in subsection on MDM).
PARITY_COUNT	- Counter maintained by MDM of port memory parity errors in a 30-minute time period.

5.13.2.2 MNL Receiver Data Substructure
(All offsets are Prefixed by OF$PCB:RCV_):

DS	- Data space number for MNL$RCV task for this link
RNODE	- Remote node at other end of link
FILEINUSE	- Address of byte file being received
STACK	- Stack for MNL$RCV

5.13.2.3 MNL Transmitter Data Substructure
(All Offsets are Prefixed by OF$PCB:XMT_):

DS	- Data space number for MNL$XMT task for this link. 0 if link down.
SAQ	- Pointer to slot add queue
CFQ	- Pointer to control frame queue descriptor block (Used only to shorten code.)
CFQDB	- Control frame queue descriptor block
RAPQ	- Pointer to remote addressed packet packet queue descriptor block
RAPQDB	- Remote addressed packet queue descriptor block
RNODE	- Number of node at remote end of this link
RPORT	- Number of I/NP at remote end of this link
LNKDLY	- Link delay in hundredths of a second
LNKSPD	- Nominal link speed
APPTHRUPUT	- Apparent link throughput in characters per second (for MPM)
FILEINUSE	- Address of byte file being transmitted. 0 if none.
XMTLST	- Pointer to linked list of slot substructures of ports included in this link's active data traffic.
RCONFIG	- Remote node's configuration level
RSWLVL	- Remote node's software revision/release level

KS

TRAFFIC	- Long term outbound link traffic (see subsection on MPMRCCM)
STACK	- Stack for MNL$XMT

5.13.2.4 Packet Data Substructure, Not Present in I/NP, VP, or XP PCB's (All Offsets are Prefixed by OF$PCB:PACKET_):

45

XMTLNK	- Link used by MAP to queue this substructure to the MAP Addressed Packet Transmitter queue
RCVLNK	- Link used by MAP to queue this substructure to the MAP Addressed Packet Receiver queue
OPQTOP, OPQBOT, and OPQLOCK	- Pointers to start of, end of, and lock byte associated with the outbound packet queue descriptor block for packets sent to this port.
RCVPTR	- Pointer to packet currently being received
IPADR	- Port address (address of associated physical port in VP PCB's)

5.13.2.5 Path Data Substructure, Present in I/TP, XP and VP PCB's (All Offsets are Prefixed by OF$PCB:PATH_):

Fields used by I/TP and VP PCB's:

CSTATE	- Call state
RNODE	- Remote node for call
RPORT	- Remote port for call
PSTATE	- Primary transmit path state
SSTATE	- Secondary transmit path state
SPD	- Path's source port speed
PRIO	- Path's priority level
PXAN	- Primary transmit path adjacent node
PXAP	- Primary transmit path adjacent port (number of XP or I/TP at next node on path)

PXNP	-	Primary transmit path I/NP
PRAN	-	Primary receive path adjacent node
PRAP	-	Primary receive path adjacent port (number of XP or I/TP in previous node or path)
PRNP	-	Primary receive path I/NP
SXAN	-	Secondary transmit path adjacent node
SXAP	-	Secondary transmit path adjacent port
SXNP	-	Secondary transmit path adjacent port
SRAN	-	Secodnary receive path adjacent node
SRAP	-	Secondary receive path adjacent port
SRNP	-	Secondary receive path I/NP
SHOPS	-	Secondary receive path length

Fields used by XP PCB's:

XSTATE	-	Path state
DSTND	-	Path destination node
DSTPT	-	Path destination port
SRCND	-	Path source node
SRCPT	-	Path source port
SPD	-	Same as for I/TP
PRIO	-	Same as for I/TP
XAN	-	Transmit adjacent node
XAP	-	Transmit adjacent port
XNP	-	Transmit path outgoing I/NPO
RAN	-	Receive path incoming adjacent node
RNP	-	Receive path incoming I/NP

Congestion Control fields, present in all path substructures:

RRSWITCHPORT- See subsection on MPM$CCM

5.13.2.6 Diagnostic Data Substructures, Present in All Physical Port PCBs:

The Diagnostic Data Substructure is created by MDM whenever a port is taken off-line and returned to the free buffer pool if and when it is put back on-line. It is used for scratch storage by MDM. The fields of the Diagnostic Data Substructure are described in the subsection on MDM.

5.13.2.7 PCB Lock Bytes

In addition to its various substructures, the PCB of each physical or virtual port has a lock byte area. This is a set of contiguous lock bytes used for synchronization both among different mainframe modules and among separate tasks in the same module. Since all lock bytes are allocated permanently at system initialization, a dynamic PCB such as an XP, PCB may not have a lock byte area. In such cases, the OF$PCB:MAIN_LOCKS field is the main data substructure in left 0. Lock bytes are defined by offsets prefixed by OF$PCB:LOCK_ and bit fields within a PCB lockbyte are defined by bit masks prefixed by EQ$PCB:LOCK. The following is a list of all lock bytes used for all port types:

STATUS - This lock byte exists in every Intelligent Port (I/P) or Virtual Port (VP) PCB. It is used for synchronization between MDM and other mainframe modules when bringing a port on-line or taking it off-line. It contains a DOWN bit which is set when the port is considered off-line by MDM and a BUSY bit for each of four different modules. The BUSY bits are set by the respective modules when they are using the port. The DOWN and BUSY bits are described more fully in the subsection on MDM.

USERS - This lock byte is currently used only by the Mainframe Network Link (MNL) Module. It is used to synchronize among the MNL transmit and receive tasks and the MNL addressed packet message handler task when links are brought up and down. Its use is described more fully in the subsection on MNL.

PACKET - This lock byte is used for synchronization between the Mainframe Addressed Packet (MAP) Module and MDM when a physical port is brought on-line and when required addressed-packet cleanup is performed to take a physical port off-line. It contains the following bit fields defined by masks prefixed by EQ$PCB:LOCK_PACKET_:

DOWN - Cleanup bit set by MAP$CLEANUP at start of clean-up and cleared by MDM when the port is brought back on-line.

TIMER - Two bit timer field used by MDM to decide when a port has timed out. The field is reset by MAP to 0 whenever a packet is received from the port and is incremented by MDM every six seconds while port is on-line. When the timer is incremented to 2, the port is considered to have failed.

RCV & XMT - MAP addressed packet receiver and transmitter active bits, in that order. The appropriate bit is set by MAP$PINT when a packet receive or transmit interrupt occurs and is reset by MAP$PRCV (RCV bit) or MAP$PXMT (XMT bit) when handling of the interrupt is completed. The RCV or XMT bit is set if and only if the inbound or outbound packet BIC FIFO, respectively, is in by MAP. When MAP sets a FIFO flag at the end of interrupt processing, it must do so simultaneously with the clearing of the RCV/XMT bit.

UP - This bit is set if and only if the port is up for the purpose of receiving outbound packets. It is set by MDM after the initial wakeup packet from a newly operational port has been received and sent back to the port. It is cleared by MDM when the port is taken off-line. It is needed to guarantee that the first packet any physical port receives is the response to its wakeup packet.

6. INTELLIGENT PORT MODULE DEFINITIONS

The D814 intelligent port software consists of software in three general classes.

1. Intelligent Port Operating System (IPOS)
2. Common Software
3. Unique Protocol Software

The operating system software is common to all ports, though certain submodules in this module may not be present in every port subsystem. This module is described in Section 6.1

The common software consists of a collection of modules which are used by a general class of ports, but not by all ports. Software of this type is presented in Sections 6.3 through 6.5.

The unique protocol software is that part of a port which implements its specific communications personality in interfacing with the outside world. These modules are presented beginning with Section 6.6.

6.1 Intelligent Port Operating System

The Intelligent Port Operating System Module (IPOS) is organized as a set of submodules, each of which provides a related set of functions.

1. System Scheduler
2. Real-Time Clock and Timer
3. Batch Processor
4. Buffer Utilities
5. Queue Manipulation
6. Addressed Packet Handler
7. Utilities
8. IPOS Initialization
9. Processor Loading Calculation
10. Light Control
11. Memory Modification
12. Software Uploader
13. Background Checker

Descriptions of these modules are found in Sections 6.1.1 through 6.1.10.

IPOS itself is maintained as two closely knit units designated as IPOS and IPOS/09. The designate IPOS supports the M6800 processor and IPOS/09 supports the M6809 processor. While most Operating System modules for both the 6800 and 6809 are identical in operation, the Initialization, Real-Time Clock, and Scheduler are different; these modules are provided with separate descriptions. Task Control Blocks are also different for IPOS and IPOS/09.

6.1.1 Task Scheduler Submodule

The scheduler recognizes seventeen software process priority levels. The lowest level is zero and the highest level is sixteen.

Before continuing with the description of the scheduler, some terms need to be defined. A task is RUNNING when it has control of the processor. A task is INITIATED when it is placed in the task queue of a specific priority level. A task is STARTED when it is removed from the task queue and given control of the processor. A task is SUSPENDED when an interrupt is being serviced or when its machine state is stacked so that a higher level task may be run. A task is RESUMED when its machine state is unstacked and it is given control of the processor. A task TERMINATES when it informs the scheduler that it has no more work to do, and its actual machine state is removed from the stack by the scheduler. A FORK occurs when one task causes another task to be initiated, but the task which causes the initiation does not wish to be terminated. A TERMFORK occurs when one task wishes to terminate and wants another task to be initiated (either at the same or another priority level).

The functions to be performed by the task scheduler are:

a. Initiate a task (from an interrupt routine) at a specific priority level and enter scheduler.

b. Fork another task (from process level) at a specific priority level and enter scheduler.

c. Fork another task (from process level) at a specific priority level and terminate the forking task, entering the scheduler.

d. Terminate a (process level) task and enter scheduler.

e. Dispatching of hardware interrupt requests.

Task Initiation

Each task which is running or initiated in the system (or in a timer routine, see 6.1.2) must have a task control block (TCB) associated with it. Task control blocks will be buffers obtained from the buffer management module (see 6.1.4). These buffers will be linked into the task queues when the task is initiated and will be placed in a table of active TCBs when the task is started.

The IP software can be running in one of two modes at any instant in time: interrupt level or process level. Interrupt level is the level at which interrupts are serviced and tasks are initiated, started, and terminated. At interrupt level interrupts are always masked. Process level runs almost exclusively with interrupts enabled. Only during delicate operations like queue manipulation, which could leave a data structure in an unviable state if interrupted, will interrupts be masked. Note that, using these definitions of interrupt and process level, it is possible to enter interrupt level from process level without receiving an interrupt. This is the case when a task terminates and enters the scheduler to pick up a new task. This is an entry into interrupt level as we have defined it.

The port processor will spend as little time as possible with interrupts masked. This is accomplished by having the actual interrupt routines typically set up work for a process level task and enter the scheduler which decides which task to run next. Any volatile information is captured from the hardware by the interrupt routine and saved. This is information such as inbound terminal data from the 2651, 2651 status, BIC status, etc.

When an interrupt occurs, IPOS will dispatch to the proper interrupt handler. The interrupt handler may service the interrupt directly, or it may initiate a task to service the interrupt at some software priority level. It may be necessary to communicate some information to this task. If so, the information can be passed in the Task Control Block.

When a task is initiated, if the new task is of a higher priority than the task when was running, the new is started. If the new task is of equal or lower priority compared to the task which was running, the task which was running is resumed, and the new task is queued to a task level queue.

The scheduler takes total responsibility for "queued" (initiated) tasks and the associated priority level task queues. The same entry point may be successfully initiated and in the task queue more than once concurrently, even at several different priority levels, if this should be desirable. It is, however, the job of a multiply queued task to find and manipulate the correct data each time it is started.

Note that for IPOS/09, all task initiation is performed using system macros.

Task Termination

When a task runs to completion, it requests termination. At this time all task queues and the highest priority task which is suspended are scanned. The highest priority task of the above receives control.

Task Control Blocks

The format of an IPOS Task Control Block (TCB) is:

0 - 1	2 - 3	4 - 5	6	7	8 - 9	A - B	C - D	E - F
TSK_LNK	TSK_ADR IPOS TMR_LNIC IPOS/09	TMR_CNT	LVL	?	???	???	???	TSK_SP

10 - 11	12 - 13	14	15	16 - 17	18 - 19	1A - 1B	1C - 1D	1E - 1F
BQTOP	BQBOTM	RUN FLG	?	???	???	???	???	???

Note that TCBs for IPOS/09 are always 256 bytes long and aligned on a 256 byte boundary. The fields defined above are common to both IPOS and IPOS/09.

The fields are:

TSK_LNK - Used by the scheduler to link the TCB into the job queue. Used by the timer routines to link the TCB into the timer queue. Used as scratch area by other system routines (IPOS) reserved for scheduler (IPOS/09).

TSK_ADR - The entry point address of the task associated with the TCB when the TCB is queued to either the job queue or the timer queue. Used as scratch area by other system routines. This field is used only in reference to IPOS, not IPOS/09, which reserves these bytes for the TIMER (queuing) at all times.

LVL - This byte is the level byte. The bit assignments in the byte are:

0 - 4	Software priority level number
5 - 7	Unused currently

TMR_CNT - The timer delay count for the timer routines. (Used only during delay operations.)

TSK_SP - Contains a pointer to the task stack.

The above bytes (0 - F) of the active TCB (belonging to the task which is running) is mapped to locations X'0000' through X'000F' at all times by the IPOS scheduler to decrease IPOS overhead; IPOS/09 uses the 6809 direct page register to map locations X'0000' - X'00FF'.

The fields in bytes 10 - 14 of the TCB are used only by batch tasks (see Section 6.1.3); therefore, they are free and available in all TCBs except Batch TCBs (BTCBs).

BQTOP - Pointer to top entry queued to BTCB.
BQBOTM - Pointer to bottom entry queued to BTCB.
RUNFLG - Non-zero if BTCB task is running or queued to be run.

The entry points for obtaining TCBs are:

Entry Point - IP$SCHD:GTCB (IPOS, IPOS/09)

Function

Get a TCB of specified length and set it up. This call may be used only during initialization. Also clears the first 5 bytes of the second buffer in case the TCB is to be used as a Batch TCB (see 6.1.3). IPOS/09 will force 256 byte TCB alignment. Buffers lost in aligning TCBs are recoverable using IP$SCHD:GTCB_CLEANUP.

Entry Conditions - (JSR IP$SCHD:GTCB)

* A register contains the task priority.

* (IPOS) B register contains the number of buffers to be included in the TCB in addition to the mapped 16 bytes (may not be zero).

  (IPOS/09) B register N/A, all TCBs are 256 bytes long.

* X register contains the task entry point address.

Exit Conditions

* A, B registers destroyed
* X register points to the TCB

Entry Point - IP$SCHD:GTCB1 (IPOS only)

Function

Identical to IP$SCHD:GTCB, except that only one additional buffer is obtained. The B register need not be set on entry.

Entry Conditions (JSR IP$SCHD:CLEANUP) IPOS/09

None

Exit Conditions

A,B,U,X,Y destroyed.

Function

Release GTCB wasted buffers.

User Program Interface

The IP operating system maintains a job queue for each priority level. Tasks are removed from the queues on a first-in first-out basis. If there is a choice between running a suspended task or starting a new task of the same priority as the suspended task, the decision will always be to resume the suspended task. The routines for accessing the job queues and terminating tasks are given below.

Entry Point - IP$SCHD:FORKTCB

Function

Initiate a logically parallel task at a specified priority level (1 - 8).

Entry Conditions - JSR IP$SCHD:FORKTCB (IPOS);
System Macro FORK (IPOS/09)

* X Register points to the TCB to be forked
* TCB contains the entry address and the software level number

Exit Conditions

* All registers destroyed (IPOS)
* A,B,X registers destroyed (IPOS/09)
* CC:I= 0

Entry Points - IP$SCHD:TERMFORKTCB

Function

Initiate a task at a specified priority level after terminating the caller.

Entry Conditions - (JMP IP$SCHD:TERMFORKTCB) IPOS;
System Macros TFORK IPOS/09
TFORKI (Interrupt Level IPOS/09)

* X register points to the TCB to be forked
* TCB contains the entry address (IPOS) and the software level number

Exit Conditions

* None

NOTE: This routine does not return to the caller.

Entry Point - IP$SCHD:TERM and IP$SCHD:TERM_NOTCB
(Note: Entered via SWI3 for IPOS/09)
System Macros TERM and TERMN

Function

Terminates execution of the calling task and enters the scheduler for selection of the next task.

Entry Conditions

* None

Exit Conditions

* None

NOTE: This entry point does not return to the caller.

In addition to the above entry points, there are macro instructions which generate in-line code for forking task levels 9-16. This is known as a fast fork.

Two macros are available for performing fast forks, FSTFRK and FSTFRKR. They reside in MAC$>FRKMAC (IPOS) and MAC$>FORKMC (IPOS/09).

The form of FRKMAC is:

```
LABEL          FSTFRK          TCBPTR,LEVEL,*        (IPOS)

LABEL          FSTFRK          LEVEL                 (IPOS/09)
```

Where:

LABEL is an optional label which is to be assigned to the first instruction generated by FSTFRK.

TCBPTR is the name of a 2 byte location in memory which contains a pointer to the task control block to fork.

LEVEL is the number (9-16) of the IPOS software level at which to fork the TCB.

* Is an optional parameter indicating that the code is to be assembled in an interrupt routine of that interrupts are masked on entry to FSTFRK and are NOT to be unmasked on exit from FSTFRK. It is either left unspecified or is specified as an asterisk.

The form for FSTFRKR is:

```
     LABEL         FSTFRKR          TCBPTR,REG,*      (IPOS ONLY)
                   FSTFRKR          REG               (IPOS/09)
```

Where:

REG is either A or B, depending on which register contains the IPOS software level on entry to FSTFRKR.

All other parameters are as in FSTFRK above.

Fast fork macros destroy the X register (IPOS only; A-reg IPOS/09). FSTFRKR destroys the register specified as REG.

Examples of proper calls are:

```
     LBL24         FSTFRK          OF$IP$ITP:XYZTCB,EQ$IP$ITP:XYZLVL
                   FSTFRKR         OF$IP$ITP:ABCTCB,B,*
```

System Macros

IPOS/09 uses system macros to perform task initiation, task termination, and timer delay functions. The macros and parameters are:

FASTCB   TSKADR, LEVEL

This macro creates a TCB at a fast fork level (9-16) and places the TCB into the fast job queue. TSKADR is the TCB starting address, LEVEL is the TCB running level. If TSKADR is omitted, the starting address is assumed to already be in the X-register. LEVEL must be specified.

FORK   TCB, LEVEL

This macro queues a TCB for task initiation (levels 0-16). TCB is a pointer to a TCB (if specified); if omitted, the X-register is assumed to already point to the TCB. LEVEL is the task running level (if specified); if omitted, the task runs at the level last stored in the TCB.

TFORK TCB, LEVEL, RESTART

This macro terminates the currently active processor level task and forks another (or the same) processor level task. RESTART, if specified, indicates the next starting address for the task terminating; if not specified, the next instruction after the TFORK macro will receive control the next time the task is initiated. TCB and LEVEL are as for the FORK macro.

TFORKI TCB, LEVEL

This macro is equivalent to TFORK but terminates an interrupt level routine. TCB and LEVEL are as for the FORK macro.

TERM

This macrto terminates an interrupt level routine. There are no parameters.

TERMN RESTART

This macro terminates a processor level routine. RESTART is as for the TFORK macro.

BTERM

This macro is used by the batch task utility and is equivalent to TERMN exept that "RESTART" is automatically reset to the original task starting address. BTERM requires no parameters.

FSTFRK LEVEL

This macro initiates a fast fork processor level (9-16) task. Level (required) specifies the level at which the TCB was created (FASTCB).

FSTFRKR REG

This macro is equivalent to FSTFRK with the level in Register "REG".

DELAY TIME

This macro causes the processor level routine executing it to be delayed (suspended) for a specific time period. ""TIME" specifies the suspension period (in increment of 10 ms) if "TIME" is not specified TMR_CNT is assumed to be already set in the TCB.

FORKDELAY TIME, TCB, LEVEL

This macro causes a processor level routine to be forked after a delayed period of time. "TIME" is as for the DELAY macro. "TCB" and "LEVEL" are as in the FORK macro.

Interrupt Handling (IPOS)

When an interrupt occurs, IPOS determines which device is requesting service and dispatches through a table set up at assembly and link time to the appropriate interrupt routine. A predetermined (by equates) set of addresses are used to contain the entry point addresses for user program entry points to service interrupts which are to be handled by the protocol module(s).

Interrupt Handling (IPOS/09)

IPOS/09 interrupt handling makes use of the 6809 expanded capabilities to allow vectored interrupts. An I/O area address access immediately determines the location in a 128 entry vector table of the correct routine to handle all possible interrupts. The interrupt vector table is defined by user routines during module assembly. The IPOS/09 related entries (RTC, BIC 0) are handled by IPOS/09. Prioritization of system interrupts and BIC-1 interrupts are controlled via an assembled vector control byte.

Example

Last is a scenario, illustrated in Figure 6.1.1, of how this scheme works. The level assignments shown in the table below are assumed for the example.

Int.	2651 input, 2651 output, control signal change, real time clock update
8	Outbound FIFO data through to outbound protocol buffer
7	Outbound protocol through to 2651 output buffer
6	Inbound data from buffer to BIC FIFO
5	Inbound terminal buffer, protocol, to FIFO buffer
4	Outbound packet transfer for BIC packet queue
3	Inbound packet transfer into BIC packet queue
2	Packet processing
1	Real time clock dependent routines (batch)
0	Background diagnostics

The scenario is:

a. The port processor is running background diagnostics at level 0.

b. An interrupt from the BIC initiates a level 4 transfer from the Packet FIFO. Level 0 is suspended.

c. Before terminating, level 4 forks a level 2 packet processing task via a software interrupt.

d. Level 4 terminates and level 2 starts.

e. The real time clock interrupts and schedules a task at level 1. since we suspended level 2, the clock task at level 1 is flagged as active and level 2 is resumed.

f. Level 2 TERMFORKs and initiates a packet to be sent inbound at level 3.

g. An interrupt from the 2651 is received and a level 5 task is initiated to handle an inbound transfer. The level 3 task is suspended and the level 5 task starts.

h. Let us assume that this is a synchronous terminal and that this transfer is simply waiting for the end of a message. Level 5 terminates and level 3 is resumed.

i. Level 3 finishes and terminates and the level 1 clock task is started.

j. Level 1 terminates, no other tasks are active so level 0, background diagnostics, are resumed.

T = Terminate

I = Initiate

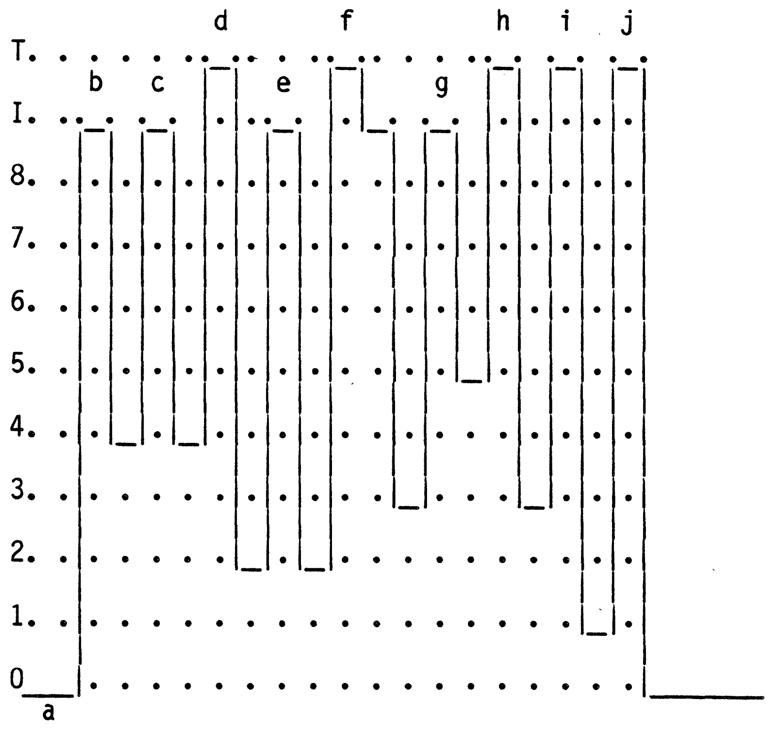


Figure 6.1.1

6.1.2 Real-Time Clock Submodule

The Real-Time Clock Submodule provides the timing capabilities of IPOS, including time-of-day, interval timing, timed task initiations, and task delaying.

Clock Functions

a. Provide an interval timer which continually increments every 10 milliseconds (24 bits).

b. Provide for initiating a task after a period of time has elapsed (in increments of 10 ms).

c. Provide for delaying a running task for a period of time (in increments of 10 ms).

Interval Timer

Routines will be provided to set and read the current value of the 24-bit interval timer for use in recording elapsed time, detecting delays, and any other need to know how much time has elapsed since a prior event. The user routine may utilize as many or as few of the 24 bits as it desires, or any subfield, to obtain the time interval/resolution trade-off it requires.

Entry Point - IP$RTC:SETIME

Entry Conditions

* B register = high 8 bits of 24-bit timer
* X register = low 16 bits of 24-bit timer

Exit Conditions

* All registers unchanged

Entry Point - IP$RTC:GETIME

Entry Conditions

* None

Exit Conditions

* A register unchanged (U,X IPOS/09)
* B register = high 8 bits of 24 bit-timer
* X register = low 16 bits of 24-bit timer

Timed Task Initiation

A 16-bit timed task initiation is started by setting a non-zero value into the timer count of a task's TCB and chaining it into the clock timer list via a routine to be provided. The timers decrement and the associated entry points are initiated when their timers reach zero. Time specification is in 10 millisecond units.

Entry Point - IP$RTC:FORKDELAY

Entry Conditions (IPOS)

* X register points to TCB to be forked
* TSK_TMR must be set in the TCB

Entry Conditions (IPOS/09)

Called using system macro FORKDELAY.

Exit Conditions

A, B registers destroyed (IPOS)
A register destroyed (IPOS/09)

Delaying a Task

A running task may ask to be delayed for a period of time (letting other tasks, even of the same or lower priority level run in the meantime) by calling a routine to be provided. Machine state (registers and condition codes) will not be preserved during a delay, but the delaying task may leave information on the stack or in the TCB.

Entry Point - IP$RTC:DELAY

Entry Conditions (IPOS)

* TSK_TMR must be set in the TCB

Entry Conditions (IPOS/09)

Called using system macro DELAY.

Exit Conditions

* All registers destroyed (IPOS)
  A,X registers destroyed (IPOS/09)
  TSK_ADR destroyed (IPOS)

IP$RTC:CHANGE

This routine alters the value of the OF$IP$TCB:TMR_CNT field of a task on the RTC timer queue. This effectively changes the restart time of a delayed (or delay-forked) task. Interrupts are masked during the procedure.

OF$IP$TCB:XBYTE is set at termination, and may be used as a 'valid' flag for routines which are cancellable.

Entry Point - IP$RTC:CHANGE

Entry Conditions:

X-reg - TCB to be updated
OF$IP$TCB:XSAVE - New delay interval

Processing

1. New TMR_CNT = time elapsed since last RTC interrupt + new delay interval (OF$IP$TCB:TMR_CNT = OF$IP$OS:TVP + OF$IP$TCB:XSAVE)

2. If this TCB is next to time out (i.e., OF$IP$TCB:TMR_CNT <OF$IP$TCB:XHOLD), then OF$IP$OS:THOLD = OF$IP$TCB:TMR_CNT

3. OF$IP$TCB:XBYTE = 1

Exit Conditions

A-reg, B-reg - destroyed
X-reg - unchanged
OF$IP$TCB:XBYTE = 1

Recommendations

Current ISTP$ASCII:BSC_TBL
ISTP$EBCDIC:BSC_TBL
ISTP$TRANSC:BSC_TBL
moved to
IP$BSCTBL:ASCII
IP$BSCTBL:EBCDIC
IP$BSCTBL:TRANSC

Since they will be used by both MSTP and SSTP_BSC.

6.1.3 Batch Processing Submodule

The purpose of batch processing is to allow one task with one task control block to sequentially process information in an order determined by the first-come first-served principle.

The functions performed by the batch processing submodule are:

a. Enqueue a request to a BTCB and fork the task if it is not running.

b. Dequeue a request from the current running BTCB and terminate if none.

Requests are in the form of buffers (or byte files or byte queues) queued (chained) to the BTCB.

Batch Task Control Blocks (BTCBs) must be obtained at initialization time because they depend on the ability to obtain contiguous memory buffers from the free buffer pool (see 6.1.1). BTCBs, however, are a minimum of three buffers long.

The fields in the second BTCB buffer are:

BQTOP	- Pointer to top entry queued to BTCB
BQBOTM	- Pointer to bottom entry queued to BTCB
RUNFLG	- Non-zero if BTCB task is running (or queued to be run, 6809)

The first 2 bytes of the entry's buffer are used to enqueue it to the TCB.

The routine for enqueueing a batch request is IP$BATCH:ENQ.

If the BTCB queue is not empty, the last entry is linked to the new entry and BQBOTM is set to point to the new entry. If the queue was empty, a pointer to the new entry is also saved in BQTOP. If enqueueing to an empty queue, the run flag is checked. If the flag is set, control is returned to the caller. Otherwise, the run flag is set, the BTCB is forked, and then control is passed to the caller.

Entry Conditions

* X register points to the BTCB to which the enqueue is to be performed

* A, B registers point to the element to be enqueued

Exit Conditions

* All registers destroyed

The routine for dequeueing batch requests (IP$BATCH:DEQTRM) is intended to be used by the batch tasks to obtain the requests queued to them.

IP$BATCH:DEQTRM - A dequeue is attempted from the current BTCB (pointed to be OF$IP$OS:TCBADR). If no entries are present, it pops the return address off the stack and performs an IP$SCHED:TERM_NOTCB (6800) or IP$SCHD:BATCH_TERM (6909) to terminate after clearing the run flag in the BTCB. If an entry is present, its pointer to the next entry is saved as BQTOP. IF this pointer is zero, the bottom pointer, BQBOTM, is also cleared.

Entry Conditions

* None

Exit Conditions

* X register points to the dequeued element
* A, B registers destroyed

6.1.4 Buffer Management Submodule

Introduction

The Buffer Management Submodule (FBMS) is part of the D814 IP operating system. This module contains utility routines for maintaining the port's free buffer pool. The buffers in this pool are the dynamic memory units which tasks can obtain and return in real-time to meet such memory requirements as temporary data storage, input/output character buffering, and inter-task communication message buffers.

General Description

The FBMS has two main functions. The first function is to maintain the D814 I/P free buffer pool and to keep the statistical information necessary for determining buffer utilization. The second function is to provide useful buffer utility features for the system in a central software module. Two buffering utilities are provided. The first is a general byte file utility and the second is a byte queue buffer utility.

1. Free Buffer Pool Management

The D814 software system maintains a pool of fixed size free buffers so that tasks in the system may be able to dynamically obtain memory resources. This pool is created at IPOS initialization time by the System Initialization Module and is maintained during system execution by the Buffer Management Module (FBMS). Tasks can obtain and return buffers from this pool by calling subroutines in the FBMS. The pool is kept as a queue so that a historical record of buffer use is available and so that background memory diagnostics which will test all of the buffer pool can be implemented. The buffer manager maintains a count of the total number of buffers in the free pool and a count of those presently allocated to software tasks. These numbers are used to calculate buffer utilization statistics.

The buffer pool will have two operating modes - normal and priority. When the number of free buffers in the pool is less than a specified threshold, the buffer pool goes into "priority" mode. In this mode only "priority" get buffer requests will be allowed to be successful. The purpose of the priority mode is to control buffer pool underrun. In priority mode, system software modules that need buffers but are low priority will suspend operation until the buffer pool builds back up again to an acceptable level and the pool reenters normal mode. When the pool goes into priority mode, a flag will be set so that a monitoring task can report the condition at some later time.

0 - 3	4 - 5	6 - D	E - F
QUEUE LINKS	LBPTR	...	LNKPTR

QUEUE LINKS	- 4 bytes for use by queue utility (see 6.1.5)
LBPTR	- Pointer to last buffer in list
LNKPTR	- Pointer to next buffer in list

The following operations will be available on free buffer pool:

A.

Routine GBUF - Obtains one buffer from the free buffer pool

Entry Point - IP$FBMS:GBUF_PRI - High priority entry

Entry Point - IP$FBMS:GBUF

Entry Conditions

* None

Exit Conditions

B,X registers destroyed

* If buffer available:

X-reg = address of buffer
CC:Z = 0
CC:I = 0

* If buffer not available:

CC:Z = 1 and CC:I = 0

B.

Routine GLIST - Get formatted linked list of buffers

Entry Point - IP$FBMS:GLIST

Entry Conditions

* B-reg = number of buffers in list (n<= 255, n=0=256)

Exit Conditions

```
*    X-reg = pointer to list header
*    B-reg = destroyed
     U-reg = destroyed (IPOS/09)
*    CC:I = 0
*    CC:Z = 1 if not successful
```

C.

Routine RBUF - Return one buffer to the free buffer pool

Entry Point - IP$FBMS:RBUF

Entry Conditions

* X-reg = Address of buffer

Exit Conditions

```
     B,X-reg = destroyed
     U-reg   = destroyed (IPOS/09)
*    CC:I    = 0
```

Entry Point - IP$FBMS:RBUF_SP

* Resets pointer to beginning of buffer

Entry Conditions

* X-reg = Any address in returned buffer

Exit Conditions

* Same as RBUF

D.

Routine RLIST - Returns list of 'n' buffers

Entry Point - IP$FBMS:RLIST

Entry Conditions

* X-reg = Address of list of buffers to be returned, LBPTR of first buffer = pointer to the last buffer in the list. NBUFS of first buffer = count of 'n' buffers in the list.

Exit Conditions

```
    A,B,X-reg = destroyed
    U-reg     = destroyed (IPOS/09)
*   CC:I = 0
```

E.

Routine RCHAIN - Returns list of buffers

Entry Point - IP$FBMS:RCHAIN

Entry Conditions

* X-reg = Address of returned list of buffers, LNKPTR of the last buffer in the list must be null.

Exit Conditions

* CC:I = 0

2. Byte File Buffer Utility

IP$BFILE provides a utility submodule for creating, deleting, and maintaining a byte file buffer system. These byte file buffers are not multiprocessor interlocked so only one task may be using a byte file buffer at any one time.

The structure of a byte file is that it has a header buffer pointing to a list of buffers, each linked to the next with the last 2 bytes. The format of the header buffer is:

0 : 1 : 2 : 3	4 : 5	6	7	8	9	A	B : C : D	E : F

BYTES 0 - 3	- Reserved for linking files to lists
BYTES 4 & 5	- Pointer to last buffer in file
BYTE 6	- Total # of buffers making up file
BYTE 7	- Number of bytes allocated in file body
BYTE 8	- Address of highest written byte
BYTE 9	- Address of last byte written
BYTE A	- Address of last byte read
BYTES B -> D	- Not used
BYTES E & F	- Pointer to first buffer of file body

The file body is composed of a linked list of buffers where the first 14 bytes of each buffer are byte file data storage and the last 2 bytes are a link pointer to the next buffer in the file body. The last link pointer in the file is zero.

The following functions will be provided for manipulating byte file buffers:

A.

Routine CREATE - Creates a byte file

Entry Point - IP$BFILE:CREATE

Entry Conditions

* None

Exit Conditions

* If available:

B-reg = destroyed
X-reg = file header address
CC:Z = CC:I = 0
OF$IP$TCB:XSAVE = ptr to byte file descriptor

* If not available:

B-reg = destroyed
CC:Z = 1
CCI = 0

B.

Routine DELETE - Deletes a file

Entry Point - IP$BFILE:DELETE

Entry Conditions

* X-reg = File header address

Exit Conditions

* All registers destroyed
* OF$IP$TCB:XSAVE = ptr to byte file descriptor
* CC:I = 0

C.

Routine READ - Reads byte 'n' from a given file

Entry Point - IP$BFILE:READ

Entry Conditions

* B-reg = Byte address 'n'
* X-reg = File descriptor address

Exit Conditions

* A-reg = Contents of the n^{th} byte
* CC:V = 'out of range' error
* OF$IP$TCB:XSAVE = ptr to byte file descriptor

D.

Routine SREAD - Reads sequentially 'next' byte from a file

Entry Point - IP$BFILE:SREAD

Entry Conditions

* X-reg = File header address

Exit Conditions

* A-reg = Contents of the 'next' byte
* B-reg = Address of 'next' byte in file
* CC:V = 'out or range' error
* OF$IP$TCB:XSAVE = ptr to byte file descriptor

E.

Routine WRITE - Writes into the n^{th} byte of file

Entry Point - IP$BFILE:WRITE

Entry Conditions

* A-reg = Data byte to be written to file
* B-reg = Byte address 'n'
* X-reg = File descriptor address

Exit Conditions

* CC:Z = 'unable to write' message
* CC:I = May be cleared to 0
* OF$IP$TCB:XSAVE = ptr to byte file descriptor

F.

Routine SWRITE - Writes sequentially into 'next' byte of file

Entry Point - IP$BFILE:SWRITE

Entry Conditions

* A-reg = Data byte to be written to file
* X-reg = File descriptor address

Exit Conditions

* B-reg = Address of 'next' byte
* CC:Z = 'unable to write' message
* CC:I may be cleared to 0
* OF$IP$TCB:XSAVE = ptr to byte file descriptor

3. Byte Queue Buffer Utility

IP$BQUE provides a utility submodule for creating, deleting, and maintaining byte queue data structures. The byte queues are multiprocessor interlocked so that one task may be putting bytes into a byte queue while another task may be removing bytes from the queue. Because of interlocking X'00' may not be stored in the byte queue. Byte queues have no maximum size.

The first buffer, known as the queue descriptor, has the following format:

0 - 3	4 - 5	6	7 - 8	9 - 10	11 - 13	14 - 15
N/A	LAST BUFFER	# BUFFERS	HEAD POINTER	TAIL POINTER	N/A	LINK POINTER

```
BYTES  0 -  3 - Are reserved for use by the Queue Utility routines.
BYTES  4 &  5 - Currently unused.
BYTES  6 &  7 - Contains the number of buffers  in the list.
BYTES  8 &  9 - Point to the next byte to get.
BYTES 10 & 11 - Point to the next byte to put.
BYTES 12 - 13 - Point to next byte to be read.
BYTES 14 & 15 - Currently unused.
```

The next byte to be written (pointed to by tail pointer) always contains binary zeroes. When a byte is to be written into the byte queue, the next byte is cleared to zero and then the new data byte is written. This allows the 'get' routine to check for an empty queue without having to disable interrupts and compare head and tail pointers. It simply gets the byte pointed to be the head pointer; if it is zero, the queue is empty.

The byte queue routines consists of four user called subroutines.

A.

Routine CREATE - Creates a byte queue

Entry Point - IP$BQUE:CREATE

Entry Conditions

* none

Exit Conditions

* If available:

X-reg = Address of queue descriptor
CC:Z = 0
CC:I = 0
OF$IP$TCB:XSAVE = ptr to byte queue descriptor

* If not available:

CC:Z = 1
CC:I = 0

B.

Routine DELETE - Deletes a byte queue

Entry Point - IP$BQUE:DELETE

Entry Conditions

* X-reg = Queue descriptor address

Exit Conditions

* CC:I = 0

C.

Routine PUTBYT - Puts a byte into a queue

Entry Point - IP$BQUE:PUTBYT

Entry Conditions

* A-reg = A byte of data
* X-reg = Queue descriptor address

Exit Conditions

* B-reg = destroyed
* CC:Z = 'unable to enqueue' message
* CC:I may be cleared to 0
* OF$IP$TCB:XSAVE = ptr to byte queue descriptor

D.

Routine GETBYT - Gets a byte from a queue

Entry Point - IP$BQUE:GETBYT

Entry Conditions

* X-reg = Queue descriptor address

Exit Conditions

* A-reg = Dequeued byte from queue
* CC:Z = 'Empty queue' condition
* CC:I may be cleared to 0
* OF$IP$TCB:XSAVE = ptr to byte queue descriptor

E.

Routine READBYT - non-destructive read of byte from a queue

Entry Point - IP$BQUE:READBYT

Entry Conditions

* X-reg - queue descriptor address

Exit Conditions

* A-reg = byte read from queue
* CC:Z = 'End of queue' condition
* CC:I = may be cleared to 0
* OF$IP$TCB:XSAVE = ptr to byte queue descriptor

F.

Routine READBYT_RESET - reset READBYT pointer to beginning of byte queue

Entry Point - IP$BQUE:READBYT_RESET

Entry Conditions

* X-reg = queue descriptor address

Exit Conditions

* A-reg = byte read from queue before reset or 0 if end
* B-reg = destroyed
* CC:Z = 'End of Queue' condition
* CC:I = may be cleared to 0
* OF$IP$TCB:XSAVE = ptr to byte queue descriptor

4. Ring Buffer Utility

IP$RING provides a utility submodule for creating and maintaining ring buffer data structures.

A ring buffer is formed from a list of buffers in contiguous memory spaces. The first buffer is used as a header for the ring. Its fields hold the pointers and values necessary for ring maintenance as defined below.

0 - 1	2 - 3	4 - 5	6 - 7	8	9	10 - 13	14 - 15
GET POINTER	PUT POINTER	UNUSED	TOP POINTER	SIZE	COUNT	UNUSED	LINK POINTER

Bytes

0 - 1:	Point to the next byte to be read.
2 - 3:	Point to the next byte to be written.
4 - 5:	Currently unused.
6 - 7:	Point to the last byte in the buffer list.
8:	Number of data bytes allowed in the ring.
9:	Number of data bytes currently in the ring.
10-13:	Currently unused.
14-15:	Standard buffer link to the first data buffer.

The structure is manipulated in a wrap-around fashion. Data is entered and removed freom the ring according to FIFO, but emptied buffers are not released. Instead, the Get and Put pointers follow each other around the ring. Since the buffers are contiguous, after either kind of access, the appropriate pointer is updated by merely incrementing it. (Thus, the next byte to be read is not always in the buffer indicated by the link pointer of the header.) When either pointer (Get or Put) indicates that the last byte in the list has been accessed (i.e., pointer = Top Pointer), it is updated to point to the first byte in the list (i.e., pointer <-- Link Pointer).

Note: A ring buffer may contain at most 255 bytes of data.

The ring buffer routines consist of four user called subroutines:

A.

Routine CREATE: creates a ring buffer

Entry Point - IP$RING:CREATE

Entry Conditions

B-reg = size of ring buffer to be obtained

Exit Conditions

X-reg = address of ring buffer header (If CC:Z = 0)
A-reg = destroyed
B-reg = destroyed
CC:Z = 0, successful completion
CC:Z = 1, no buffers available

B.

Routine PUT: puts a byte in the ring buffer

Entry Point - IP$RING:PUT

Entry Conditions

B-reg = byte to be put in buffer
X-reg = address of ring buffer header

Exit Conditions

A-reg = destroyed
B,X-regs = unchanged
CC:Z = 0, successful
CC:Z = 1, ring buffer full
CC:C = 0, ring <= half full
CC:C = 1, ring > half full

C.

Routine PUT2: puts 2 bytes in the ring buffer

Entry Point - IP$RING:PUT2

Entry Conditions

A-reg = first byte to be put in ring
B-reg = second byte to be put in ring
X-reg = address of ring buffer header

Exit Conditions

A-reg = destroyed
B,X regs = unchanged
CC:Z = 0, successful
CC:Z = 1, ring buffer full
CC:C = 0, ring <= half full
CC:C = 1, ring > half full

D.

Routine GET: gets a byte from the ring buffer

Entry Point - IP$RING:GET

Entry Conditions

X-reg = address of ring buffer header

Exit Conditions

```
A-reg = destroyed
B-reg = byte from ring buffer (if CC:Z = 0)
X-reg = unchanged
CC:Z  = 0, successful
CC:Z  = 1, ring buffer empty
CC:C  = 0, ring <= half full
CC:C  = 1, ring > half full
```

6.1.5 Queue Utility Submodule

The Queue Utility Submodule provides a standard and safe (interrupt masked) method of implementing and manipulating queues in the D814 IP software.

Queue Elements

Data structures of any size or shape may be queued by the Queue Utility Submodule. The only requirement is that the structure to be queued have a field within it for queueing purposes, and that it have such a field for every queue that it can be a member of simultaneously. It is helpful if these fields are at the beginning of the structure, as the queue manipulation routines accept and return pointers (called element pointers) to these fields.

Queue Descriptor Blocks

A queue descriptor block is a small area of storage used for bookkeeping on a queue. One is needed for each queue and should be allocated in a fixed place in memory, as the user will need to supply a pointer to the queue descriptor block whenever he wishes to manipulate the associated queue.

Entry Point - IP$QUEUE:INITQUEUE

Entry Conditions

* X register points to the queue descriptor to be initialized

Exit Conditions

* All registers unchanged

Queue Manipulation Functions

Enqueue an Element

Link a new element into a queue by placing it behind the current tail element. Pointers are required to the new queue element and the queue block to which it is to be queued.

Entry Point - IP$QUEUE:ENQUEUE

Entry Conditions

* A, B registers point to the new entry
* X register points to the queue descriptor block

Exit Conditions

* All registers unchanged
* CC:C = 1 if queue was formerly empty

Dequeue an Element

Remove an element from the head of a queue. A pointer to the queue descriptor block is required. A pointer to the dequeued element is returned, or null if the queue was empty. The forward and backward pointers of the dequeued element are zeroed.

Entry Point - IP$QUEUE:DEQUEUE

Entry Conditions

* X register points to the queue descriptor block

Exit Conditions

* X register points to the dequeued element, or zero
* CC:Z = 1 if queue was formerly empty
* CC:C = 1 if queue is now empty

Access a Queue Element

A routine will be supplied which "steps through" a queue, returning a pointer to the "next" element in the queue (starting with the head) each time it is called. Returns null if the queue is empty or the end has been reached. Requires a pointer to the queue descriptor block.

There is a routine for resetting the "next" element to be the first element in the queue.

Entry Point - IP$QUEUE:QUEUETOP

Entry Conditions

* X register points to the queue descriptor block

Exit Conditions

* All registers unchanged

There is an entry point for stepping through the queue.

Entry Point - IP$QUEUE:QUEUENEXT

Entry Conditions

* X register points to the queue descriptor block

Exit Conditions

* X register points to the "next" element
* CC:Z = 1 if the queue is empty or the end is reached

Remove Last Element Accessed

Remove the last element accessed by the QUEUENEXT routine and relink the queue as necessary (as the element removed may have been at any random point in the queue). Returns null if the queue was empty or the accessing routine was run to the end of the queue, otherwise returns a pointer to the element removed. The forward and backward pointers of the element removed are zeroed. Requires a pointer to the queue descriptor block.

Entry Point - IP$QUEUE:DEQUEUELAST

Entry Conditions

* X register points to the queue descriptor block

Exit Conditions

* X register Points to the element dequeued
* CC:Z = 1 if the queue was formerly empty
* CC:C = 1 if the queue is now empty

6.1.6 Addressed Packet Handler

The Addressed Packet Handler is responsible for receiving outbound packets from the BIC and sending inbound packets to the BIC. The outbound addressed packets are handled by the Addressed Packed Receiver and the inbound packets are handled by the Addressed Packed Transmitter. No logical connection is required between the two.

In addition, an Addressed Packet Router routine is supplied to distribute packets to their destination. The reader should be familiar with the addressed packet format described in Section 3.2.2.

The Addressed Packet Handler is comprised of three sections:

1. Addressed Packet Receiver routines
2. Addressed Packet Transmitter routines
3. Packet Router routines

The following sections describe these routines in more detail.

Addressed Packet Receiver

The Addressed Packet Receiver has two entry points.

1. Packet Receiver Interrupt Handler
2. Packet Receiver

The first entry is an interrupt entry initiated when a BIC outbound packet interrupt occurs. The second entry performs the actual packet reception and assembly.

Packet Receiver Interrupt Handler

This entry is initiated by the occurrence of an outbound packet interrupt. This routine runs at interrupt level as defined in Section 6.1.1. It disables the BIC outbound packet interrupt, forks the Packet Receiver, and exits.

Packet Receiver

This routine (which is forked by the Packet Receiver Interrupt Handler) reads the addressed packet segments sent through the BIC, assembles the segments into packets, and calls the distribution routine to distribute the packet to the appropriate IP routine.

Addressed Packet Transmitter

The Addressed Packet Transmitter contains three entry points:

1. Packet Transmit Queue Routine
2. Packet Transmitter Interrupt Handler
3. Packet Transmitter

The first entry is a subroutine used to queue a packet to the Packet Transmitter.

The second entry is an interrupt entry initiated when a BIC inbound packet interrupt occurs. The third entry performs the actual packet transmission.

Packet Transmit Queue Routine

This routine is called by the Packet Router when a packet is to be sent to the mainframe.

The routine queues the packet to the Packet Transmitter. If the transmitter is not already active, the inbound packet interrupt in enabled. In either case, the routine returns to the caller.

Entry Point - IP$APKT:XMTQUE

Entry Conditions

* X register points to the packet to be sent

Exit Conditions

?

Packet Transmitter Interrupt Handler

This routine is initiated when a BIC inbound packet interrupt occurs and it runs at interrupt level. It disables the inbound packet interrupt, forks the Packet Transmitter, and exits.

Packet Transmitter

This routine sends the addressed packet to the BIC inbound packet buffer.

If no packet is in progress, the routine obtains the first packet queued to it. If no packets are on the queue, the routine leaves the BIC inbound packet interrupts masked and terminates. Otherwise, the packet length is obtained and the Transmitter Packet Pointer is set to first byte of the packet.

Non-Local Packet Router Routine

This routine may be called by any routine in the IP to distribute a packet to its destination. It is passed the address of the packet buffer.

The routine first checks whether the packet is for a module in _this_ IP.

If the packet is not for a local module, the routine calls the Packet Transmit Queue routine and returns to the caller.

If the packet is for a module at this I/P, meaning it is either addressed to this physical port or to a virtual port within this port, a thread number is written in the destination port field of the packet. If the destination is the physical port the thread number 0 is used. The packet is then delivered, if possible, to the proper module using the local packet router routine. If the packet cannot be delivered it is either returned to the sender or, if that cannot be done, discarded.

Entry Point - IP$APKT:ROUTE

Entry Conditions

* X register points to the packet to be routed.

Exit Conditions

* All registers destroyed.

Local Packet Router Routine

This routine may be called by any IP subroutine to deliver a packet to a destination module within the local IP. Packets in a multi-threaded port being delivered by this routine must have the destination thread number in the packet destination port field rather than the destination port address. This routine rather than the non-local packet router must be used when delivering a packet to a particular thread within a multi-threaded port if the thread's VP address is unknown.

The routine first checks that the packet destination module is valid and, if not, returns to the caller with an error condition. Otherwise the packet is enqueued to the proper TCB using the batch enqueue routine of Section 6.1.3 and the routine returns to the caller.

Entry Point - IP$APKT:ROUTE_LOCAL

Entry Condition

X-register points to the packet to be delivered.

Exit Condition

CC:Z = set if and only if the packet could not be delivered

All registers destroyed.

Address Packet Utilities

There is an address packet utility (entry point IP$APKT:SDSWAP) which will swap the source and destination address fields for an Addressed Packet. This process is useful for returning a packet.

Entry Point - IP$APKT:SDSWAP

Entry Conditions

* X-register points to address packet header buffer

Exit Conditions

* All registers destroyed.

6.1.7 Utility Submodule

Routines in this module are general utility routines that are defined so as to require no local storage. They are almost totally interruptable. Any intermediate storage required is allocated on the stack or in the task's TCB scheduler bytes.

Multiply

The unsigned integer 8 x 8 bit multiply routine is IP$UTIL:MULT. There are no error returns possible from this operation.

Entry Point - IP$UTIL:MULT

Function

Multiply two 8-bit unsigned numbers to get a 16-bit unsigned number.

Entry Conditions - (JSR IP$UTIL:MULT)

* A-register contains multiplier
* B-register contains multiplicand

Exit Conditions

* A-reg = High order part of product
* B-reg = Low order part of product
* X-reg = Destroyed

Divide

The 16-bit by 8-bit unsigned integer divide routine is IP$UTIL:DIV. An 8-bit quotient and an 8-bit remainder are returned. The possible error returns are division by zero (C set on return) and quotient overflow by 8 bits exceeded (V set on return).

Entry Point - IP$UTIL:DIV

Function

Divide a 16-bit unsigned number by an 8-bit unsigned number producing an 8-bit unsigned quotient and an 8-bit unsigned remainder.

Entry Conditions - (JSR IP$UTIL:DIV)

* A-reg = Divisor
* X-reg = Dividend

Exit Conditions

* A-reg = Quotient
* B-reg = Remainder
* X-reg = Destroyed

NOTE: (Sets 'V' when quotient overflows 8 bits. The outputs are unpredictable in this case. 'C' is set for division by zero. Returned quotient and remainder are zero.)

Block Copy

The block copy routine is IP$UTIL:COPY. It will copy a string on any length possible in the system to another specified location. The routine is a little slow setting up and dismissing, but runs reasonably fast once under way. Thus, this routine is useful to copy blocks longer than about 8 bytes, but will take, proportionally, a long time to copy only a few bytes. The main feature of this routine is that it is re-entrant and totally interruptable.

Entry Point - IP$UTIL:COPY

Function

Copy a block of consecutive data bytes from a source area to a destination area.

Entry Conditions - (JSR IP$UTIL:COPY)

* X-reg = Pointer to a 6 byte parameter vector

BYTES 0 - 1 - Number of bytes to copy
BYTES 2 - 3 - Pointer to source area
BYTES 4 - 5 - Pointer to destination area

Exit Conditions

* A-reg = Destroyed
* B-reg = Destroyed
* X-reg = Destroyed
* CC:I = 0

NOTE: (ses 2 bytes on stack.)

Line Speed Encode/Decode

The utility package provides routines for encoding and decoding 16-bit link and path speeds into a 1-byte number in a sort of floating point format. The encoded speed is composed of a four-bit exponent (high order nibble) and a four-bit mantissa (low order nibble).

The actual speed is computed at:

$S = (16\ 1/2 + B)2^A - 16$ (truncated if not an integer)

where:

S = Actual speed
A = Exponent
B = Mantissa

The encoded speed exponent and mantissa are computed as:

$A = [\log_2 (S + 16)] - 4$ (truncated if not an integer)

$B = \frac{S + 16}{2^A} - 16$ (truncated if not an integer)

This encoding scheme results in accuracy better than ±6.3 percent for speeds greater than 15 and better than ±3.2 percent for greater than 1008.

Encoded speeds are continuous in that, if A and C are encoded speeds and $A < B < C$, then B is a valid encoded speed and the actual speed represented by B is less than that represented by C and greater than that represented by A.

The package has the following entry points:

Entry Point - IP$UTIL:SPD_ENCODE

Entry Condition

* A, B registers - contain 16-bit speed

Exit Condition

* A-reg = speed in "floating point" format
* B-reg = destroyed
* OF$IP$TCB:XSAVE = destroyed

Entry Point - IP$UTIL:SPD_DECODE

Entry Condition

* A-reg = speed in "floating point" format

Exit Condition

* A, B registers = Actual speed (if no overflow)
* OF$IP$TCB:XSAVE = destroyed
* CC:C = Set if and only if overflow out of 16th bit occurs in decoding. If CC:C is set, then A, B contain X'FFFF'

6.1.8 IPOS Initialization

IP$INIT:IPINIT is the initial entry point of the IPOS. It turns on all controllable lights on the IP rail briefly and then shuts them off. It initializes IPOS control variables, the task queues, the timer queue, and sets up the free buffer pool. It sets up TCBs for the timer queue task, the background diagnostics task, the addressed packet receiver and transmitter tasks, and the timeout packet sender task. The timeout packet sender is forked and the packet transmit queue is created and initialized.

Next the user initialization is started by jumping to OF$IP$VEC:USERINIT. The user initailization may perform any IPOS function except a delay at this time, and must return to IPOS initialization with an RTS instruction.

Next a BTCB is obtained for the software uploader (Section 6.1.11) and an entry in the Module Dispatch Table (MDT) added.

Next a TCB is obtained for the processor loading calculation task and the "wake up" initial addressed packet task. Both tasks are forked. A TCB is obtained for the receiver for the reply to the "wake up" packet is obtained and set into the AP module dispatch table.

Outbound packet interrupts are enabled and the real-time clock is started. The map register is enabled (IPOS only). The stack is set to appear as though the background diagnostic task was running and has been suspended. The scheduler is then entered as though an interrupt is being returned to cause task scheduling.

6.1.9 Light Manipulation Submodule

This module will control the lights on the rail on the D814 IP cards.

The IPOS light control routines (IP$LITE) are as follows:

IP$LITE:ON - Lights corresponding to 1 bits in the A register are turned on immediately. Lights corresponding to 0 bits are not affected.

IP$LITE:OFF - Lights corresponding to 1 bits in the A register are turned off immediately. Lights corresponding to 0 bits are not affected.

IP$LITE:CHANGE - Lights corresponding to 1 bits in the A register are changed in state (on becomes off, off becomes on). Lights corresponding to 0 bits are not affected.

IP$LITE:FLASH - Lights corresponding to 1 bits in the A register are turned on at the next tick for the real-time clock, and are turned off 1 tick later. This routine produces one flash, not continued flashing. However, if at least one flash is requested between each clock tick, the light will stay lighted continuously. Lights corresponding to 0 bits are not affected.

IP$LITE:FORCE - Lights corresponding to 1 bits in the A register are turned on immediately. Lights corresponding to 0 bits are turned off immediately. The entire light register is set according to the contents of the A register.

IP$LITE:BLINKON - Causes the lights corresponding to 1 bits in the A register to be reversed in state 10 times a second. This action continues until IP$LITE:BLINKOFF is called for the same light. Lights corresponding to 0 bits are not affected.

IP$LITE:BLINKOFF - Causes the lights corresponding to 1 bits in the A register to stop the action initiated by IP$LITE:BLINKON. This routine has no affect on a light which has not been set blinking by IP$LITE:BLINKON. Lights corresponding to 0 bits are not affected. Note that a light which is in the on state when blinking is stopped by IP$LITE:BLINKOFF will stay in the on state unless turned off by IP$LITE:OFF. Lights which are off will stay off unless turned on by IP$LITE:ON.

All of the above routines destroy the A register.

6.1.10 Processor Loading Calculation Submodule

The operating system will assume the responsibility for supplying the IP M6800 processor loading percentage to the application programs.

The processor loading calculation module will, at six second intervals, calculate a number which is the percentage of processing potential (0 percent - 100 percent) which is NOT used by the background task in the preceeding six seconds. This processing time will have been used either by the applications (user) program or by operating system overhead. A processor loading of 0 percent should be impossible, since it would imply that the operating system is not running.

The number will be supplied in a one-byte field, OF$IP$OS:PROCLOAD, and should always be between 0 and 100 decimal.

6.1.11 IPOS Memory Modification

IPOS maintains a batch task utility which supports the reading and writing of on-line port memory, destructively or non-destructively. Nondestructive operations insert zeros for data whenever I/O areas are specified. Destructive operations have no restrictions as to which locations can be accessed.

The module requires an operation code in the requesting addressed packets to indicate which operation is to be performed.

Operations:

1) Read memory, non-destructive
2) Read memory, destructive
3) Write memory, non-destructive
4) Write memory, destructive.

The requesting packets are of the form:

1) one byte packet length
2) 3 byte destination (node, port, module)
3) 3 byte source (node, port, module)
4) one byte operation code
5) 3 bytes unused by IPOS utility
6) one byte completion code
7) 2 bytes starting location
8) one byte data length (length/AP for read non-destructive)
9) a. one byte number of AP's (only for read non-destructive)
   b. reserved (zero) bytes (only for write, 2 bytes)
10) n bytes DATA (only in write case)

The returning packets are of the form:

1) one byte packet length
2) 3 byte destination (node, port, module)
3) 3 byte source (node, port, module)
4) one byte operation code (from request packet)
5) 2 byte starting location
6) one byte data length (length/AP for read non-destructive)
7) two bytes reflected from requesting packet
8) one byte software designation field (for reads only)
9) one byte sequence number (read non-destructive only)
10) one byte completion code
11) a. one byte number of bytes written (write non-destructive only)
    b. n bytes DATA (read cases only)
    c. reserved (write destructive only)
12) n bytes DATA (write cases)

Read destructive is not allowed to occur during software uploading or a read non-destructive. Writes cannot occur during software uploading or during any of the operations described above. Attempting to do so will result in an error code being set in the returned addressed packet completion code field.

NOTE: This batch task is added as the second slot of the module dispatch table (MDT) during IPOS initialization.

IPOS maintains timers (set when a read or write request is first accepted by the receiving port) which are used in conjunction with a software lockbyte to prevent simultaneous read (non-destructive) and write (destructive and non-destructive) or read destructive accessing of a port. If the timer expires, and no further requests have been received from a port, IPOS unlocks the port for all operations (if only one accessor) or decrements the count of active port readers (if multiple non-destructive readers).

6.1.12 IPOS Software Uploader

IPOS contains the software upload facility for D814 ports. Software uploading involves:

1. The port's standard loadblock header is sent upline (to the mainframe/node) upon request.

2. After the loadblock header is requested and received, port software is passed as a series of addressed packets whose number and length are specified by the requesting node's module. Multiple software requests are made before all ports software has been uploaded.

The requests and data are passed through the packet BIC. This upload facility is provided as a batch task which is set up during IPOS initialization and added as the second slot of the MDT.

Software uploads may be requested at anytime without affecting the operational port. Once uploading has begun, it will prohibit writes or destructive reads (Section 6.1.11) from occurring on the software supplying port. If the port is locked (already committed to write modification or destructive operation), the initial software upload request will be rejected.

Addressed Packet formats are found below.

Header Request

1. one byte packet length
2. 3 byte destination (node, port, module)
3. 3 byte source (node, port, module)
4. one byte operation code (header request code)

Header Response

1. one byte packet length
2. 3 byte destination (node, port, module)
3. 3 byte source (node, port, module)
4. one byte requested operation code
5. one bytes software designation field (normal port type software, specialized loadable software, specialized non-loadable software)
6. one byte processor loading

7. one byte completion code
8. one byte reserved
9. 8 byte loadblock header (2 byte starting address, 2 byte byte count, 2 byte loading address, 2 byte checksum)

Software Requests

1. one byte packet length
2. 3 byte destination (node, port, module)
3. 3 byte source (node, port, module)
4. one byte operation code (non-destructive read code)
5. 2 byte starting location
6. one byte data length per (maximum 140 bytes) port generated AP
7. one byte number of APs to be returned
8. 3 bytes to be reflected in response packets

Software Response

1. one byte packet length
2. 3 byte destination (node, port, module)
3. 3 byte source (node, port, module)
4. one byte operation code
5. 2 byte starting location
6. one byte data length
7. 3 bytes reflected
8. one byte sequence number
9. one byte completion code
10. n-bytes DATA

6.1.13 Background Checker

IPOS runs a non-terminating task at level 0 which performs checking functions to insure software integrity:

1. Buffers are obtained from the IPOS free buffer pools and checked to insure parity and RAM failures are detected.

2. Code space is checksummed to insure software integrity is maintained. Checksumming is restarted by the IPOS memory modification routine after each modification to program space.

6.2 Configuration Control

The Configuration Control Module (IPCC$>XMTRCV) must be included in all I/P's. This module handles the interface with the I/P online (running) configuration. It answers requests for I/P online parameter values and services parameter change requests. The format of configuration request packets is described in Section 5.7. The I/P Configuration Control Module utilizes the same format, of course. All requests are passed along to the mainframe Configuration Control Module (Section 5.7), which updates the offline configuration, if necessary, before returning the request to the sender. It is important to realize that most I/P parameters are not changeable in the online configuration. The I/P Configuration Control Module runs as a batch task with module number EQ$IP$MDT:CHAR_XMTRCV. The entry point is IPCC$XMTRCV:ENTRY.

Multi-threaded port IPCC request packets may be addressed either to the physical port or to a virtual port (VP) within the physical port. Packets addressed to the physical port are used to handle parameters for the port as a whole (for example buffer utilization threshold) while packets addressed to a virtual port are used to handle parameters for that particular VP's thread within the port (for example, compression efficiency threshold).

The parsing of the request packet is driven by tables called Code Lists. OF$IP$VEC:CODELIST+1 contains a pointer to the Code List for physical port requests and OF$IP$VEC:MT_CODELIST+1 contains a pointer to the code to the List for the virtual port requests. The pointer at OF$IP$VEC:MT_CODELIST is left 0 for single-threaded ports. Each Code List contains a list of applicable parameter codes. The write bit (EQ$MCM:CC_WRITE) is set in the code if the parameter is changeable online and not set if it is not changeable online.

Each Code List has associated with one or more of the Characteristics Lists. The Characteristics List contains a one byte entry with the value of each parameter in its Code List. The parameters in both tables are in exactly the same order. The physical port Code List has one Characteristics List. The pointer to this list resides at OF$IP$VEC:CHARLIST+1. The virtual port Code List, if it exists, has a Characteristics List for each VP in the port. These lists reside at offset OF$IP$THREAD:IPCC_LIST in the thread structures of the VP's. (The thread structure is a data area for the VP allocated at port initialization by module ITP$.)

Two command code errors are possible when a request packet is parsed:

a. If an invalid code (one not found in the list of parameter codes) is encountered, the error code EQ$MCM:EC_INVCMD is set on the command.

b. If a code is valid, but the write bit is set in the request packet, but not in the list of parameter codes, the error code EQ$MCM:EC_NOTONLN is set, this indicates that the parameter is not changeable online.

Under normal operation, requests for parameter values are filled in, and requested parameter value changes are made and the request packet is sent to the mainframe Configuration Control Module. This module ignores value requests, and value change requests which have been flagged with error codes. It makes changes in the offline configuration corresponding to all value changes with no error code. The packet is then returned to the originator.

6.3 Call Manager

The CMM is responsible for establishing and terminating calls. In addition, the CMM also establishes the network transmit path from its port to the remote port. Call establishment and disconnection is accomplished via addressed packets containing various command codes. The Call Manager is comprised of three sections:

1. Initialization
2. Addressed Packet Handler
3. Call Initiator and Terminator

The following sections describe these routines in more detail.

6.3.1 Initialize Call Manager Data Structure

The Initialize Call Manager Data Structure routine initializes the CMM data structure (for every thread connected to this port, if the port is multi-threaded) and contains one entry point:

CMM Initialization

This routine is initiated by the protocol initiation routine.

Entry Point - IP$CMM$INIT:ENTRY

Entry Conditions

Reg Y - Thread structure address - MTP only

Exit Conditions

Reg Y - Preserved - MTP only
All registers destroyed.

6.3.2 Call Manager Main Addressed Packet Handler

This module receives an addressed packet from the AP router. It first checks to see if the packet is being returned in error; if so, special handling is performed. The packet's command code is then looked up in the CC table and the correct module is called. In the case of a multi-threaded port, the thread number is obtained fromthe destination port field of the addressed packet to select the proper call data area. The routine contains one entry point:

IP$CMM$MAIN:AP

This entry point is initiated by receiving an AP which is routed on the basis of the entry in the MDT.

Entry Point - IP$CMM$MAIN:AP

Entry Conditions

None

Exit Conditions

All registers destroyed.

6.3.3 Call Manager Addressed Packet Handler

This routine contains a few submodules that process the addressed packets received by the Call Manager Main Packet Handler routine. Each submodule processes a specific AP command type. In the case of a multi- threaded port, the thread number is obtained from the destination port field of the addressed packet to select the proper call data area. The different AP command types which are processed include:

CREATE CALL AP - from Protocol
CALL ACCEPTED AP - from Protocol
ACTIVATED CALL ACKNOWLEDGE AP - from PMM
INITIATE CALL AP - from CMM
INITIATE CALL ACKNOWLEDGE AP - from CMM
XMIT PATH ACTIVE AP - from PMM
XMIT PATH ERROR AP - from PMM
RECV PATH ACTIVATED AP - from PMM
RECV PATH FAILURE AP - from Pre ARQ-Receive
HANG-UP CALL AP - from Protocol
CALL CLEARED AP - from PMM

Detailed information on the different AP commands are contained in the D814 Product Functional Specification, Appendix B.

The following two diagrams show call establishment and call termination. Call establishment is shown in the simplest case, with no errors. It shows the case where only one side is attempting to establish the call.

```
|              Call Initiator               |              Call Receiver                |
|                                           |                                           |
| Protocol |     CMM      |      PMM        |     PMM      |     CMM      | Protocol    |
|----------|--------------|-----------------|--------------|--------------|-------------|
| CRECALL--+--->(ER)      |                 |              |              |             |
|          |              |                 |              |              |             |
|          |   INITCALL--+-----------------+-------------+--->(EL)      |             |
|          |              |                 |              |              |             |
|          |              |                 |              |  CALLREQ---+--->         |
|          |              |                 |              |              |             |
|          |              |                 |              |   (AV)<---+-CALLACC     |
|          |              |                 |              |              |             |
|          |              |                 |         <---|---ACTCALL    |             |
|          |              |                 |              |              |             |
|          |              |                 | ACTCALLACK-|--->(IL)      |             |
|          |              |                 |              |              |             |
|          |    (IR)<----|------------     |------------ |-INITCALLACK  |             |
|          |              |                 |              |              |             |
|          |    ACTCALL--+---->            |              |              |             |
|          |              |                 |              |              |             |
|          |         <---|--ACTCALLACK     |              |              |             |
|          |              |                 |              |              |             |
|          |  ESTXMTPATH-+---->            |              |              |             |
|          |              |                 |              |              |             |
|          |        <----+--XMTPATHACT     | RCVPATHACT--+---->(AC)     |             |
|          |              |                 |              |              |             |
|          |              |                 |        <----+-ESTXMTPATH   |             |
|          |              |                 |              |              |             |
|          |              |                 |              |  CALLCRE--+---->         |
|          |              |                 |              |              |             |
|          |    (AC)<----+--RCVPATHACT     | XMTPATHACT--+--->          |             |
|          |              |                 |              |              |             |
|     <---|---CALLCRE    |                 |              |              |             |
|          |              |                 |              |              |             |
```

(ER) = Establishing Remote State
(EL) = Establishing Local State
(AC) = Active State
(AV) = Activating State
(IR) = Inactive Remote
(IL) = Inactive Local

```
|        Call Terminator             |                                    |
|                                    |                                    |
|Protocol |   CMM    |     PMM       |    PMM     |   CMM     | Protocol  |
|---------------------------------------------------------------------------|
|         |          |               |            |           |           |
| HANGUP--+--->(DC)  |               |            |           |           |
|         |          |               |            |           |           |
|         | CLRCALL--+--->           |            |           |           |
|         |          |               |            |           |           |
|         | (ID)<----+--CALLCLRD     | CALLCLRD--+---->(ID)  |           |
|         |          |               |            |           |           |
|  <-----+---CALLEND |               |            | CALLEND---+----->     |
|         |          |               |            |           |           |
```

(DC) = DisConnecting State
(ID) = IDle state

6.3.4 Protocol AP Interface

The formats of the body portion of the addressed packets used to interface to the protocol module are given below:

	Byte	Contents
CRECALL	7	EQ$IP$CMM:CC_CRECALL
	8	Protocol Dependent Characteristics Byte 1
	9	Protocol Dependent Characteristics Byte 2
	10	Protocol Dependent Characteristics Byte 3
CALLCRE	7	EQ$IP$CMM:CC_CALLCRE
CALLREQ	7	EQ$IP$CMM:CC_CALLREQ
	8	Protocol Dependent Characteristics Byte 1
	9	Protocol Dependent Characteristics Byte 2
	10	Protocol Dependent Characteristics Byte 3
CALLACC	7	EQ$IP$CMM:CALLACC
	8	Error Code or Zero
HANGUP	7	EQ$IP$CMM:CC_HANGUP
CALLEND	7	EQ$IP$CMM:CC_CALLEND
	8	Error Code or Ø

6.3.5 Remote Call Manager AP Interface

	Byte	Contents
INITCALL	7	EQ$IP$CMM:CC_INITCALL
	8	Protocol Dependent Characteristics Byte 1
	9	Protocol Dependent Characteristics Byte 2
	10	Protocol Dependent Characteristics Byte 3
	11	Port Generic Type
	12	Local Characteristics
INITCALLACK	7	EQ$IP$CMM:CC_INITCALLACK
	8	Error Code or Zero

6.3.6 Path Manager AP Interface

The formats of the body portion of the addressed packets used to interface to the Mainframe Path Manager are given below:

	Byte	Contents
ACTCALL	7	EQ$MPM:CC_ACTCALL
	8	Remote Node
	9	Remote Port
	10	Initial Estimate Effective Speed
ACTCALLACK	7	EQ$IP$CMM:CC_CALLACT
	8	Error Code or Zero
ESTXMTPATH	7	EQ$MPM:CC_ESTXMTPATH
	8	Local Port Speed
	9	Path Priority/Routing Option
	10	Transfer Adjacent Node
	11	Transfer Adjacent Port
	12	Transfer Network Port
XMTPATHACT	7	EQ$IP$CMM:CC_XMTPATHACT
	8	Number of Hops in Path
XMTPATHERR	7	EQ$IP$CMM:CC_XMTPATHERR
	8	Error Code
RCVPATHACT	7	EQ$IP$CMM:CC_RCVPATHACT
	8	Number of Hops in Path
	9	Path Priority Code
RCVPATHFAIL	7	EQ$IP$CMM:CC_RCVPATHFAIL
CLRCALL	7	EQ$MPM:CC_CLRCALL
CALLCLRD	7	EQ$IP$CMM:CC_CALLCLRD

6.4 Single-Threaded Data Movement

Single-threaded data movement refers to the transfer of data between terminals over a line that is effectively point-to-point, whether it be a permanent or switched connection. In the D814 system, this implies that no two physical terminals share a single set of I/TP software, hence messages are not interleaved.

The D814 is responsible for upholding data integrity during its movement across the network. It does not, however, use an explicit end-to-end error detection scheme in order to maintain acceptable levels of data accuracy. Instead, an ARQ mechanism has been designed which is activated when a link or a node failure has been detected. This, in conjunction with the checking done by the INP, insures minimal data loss across the link.

The ARQ and flow control measures are implemented at those points at which data is transferred to and from the BIC. The modules which are responsible for this are discussed in the following two subsections.

6.4.1 BIC FIFO Handler (Module IP$FIFO$)

The BIC FIFO Handler is responsible for the transfer of data INTO the BIC Inbound and OUT of the BIC Outbound FIFO's. The module is composed of routines to accomplish the following tasks:

1. FIFO initialization
2. FIFO interrupt handling

The means used to accomplish these tasks are described in the following sections.

6.4.1.1 FIFO Initialization (Submodule IP$FIFO$INIT)

This submodule is called for two reasons,

1. for original FIFO data structure initialization, and
2. for reinitialization while the IP is active.

Accordingly, the submodule is comprised of two routines. The first is called by the protocol module during its own initialization phase, the second at CALLEND, again being invoked by the protocol module. The entry points and functions of each are defined below.

Entry Point - IP$FIFO$INIT:ENTRY

Function

Gets TCB's for FIFO Interrupt handlers, disables Inbound and enables Outbound FIFO's, and clears FIFO data structures.

Entry Conditions

None

Exit Conditions

All registers destroyed.

Entry Point - IP$FIFO$INIT:REINIT

Function

Disables BIC Inbound FIFO and enables BIC Outbound FIFO and clears OB FIFO of residual data.

Entry Conditions

None

Exit Conditions

All registers destroyed.

(In actuality, the second routine is merely a subset of the instructions of the routine performing the original initialization tasks.)

6.4.1.2 FIFO Interrupt Handling (Submodule IP$FIFO$INT)

This submodule handles Inbound and Outbound FIFO interrupts and contains two entry points, one to handle each type of IRQ. The first is entered when a BIC Inbound FIFO interrupt occurs; the second by the occurrence of a BIC Outbound FIFO interrupt.

Entry Point - IP$FIFO$INT:XMT

Function

Disables BIC Inbound FIFO Interrupts and forks IP$FLOW$XMIT which performs the Inbound FIFO transmit function.

Entry Conditions

None

Exit Conditions

None

Entry Point - IP$FIFO$INT:RCV

Function

Disables Outbound FIFO interrupts and forks IP$FLOW$RECV which performs the Outbound FIFO receiver function.

Entry Conditions

None

Exit Conditions

None

6.4.2 Flow Control and ARQ (Module IP$FLOW)

The Flow control and ARQ module is responsible for controlling data flow into the network and insuring proper data flow from the network. It contains an ARQ mechanism which is activated when a link or node failure has been detected or if the free buffer pool becomes exhausted. This mechanism is linked to the end-to-end flow control procedure.

The Flow Control and ARQ Module is comprised of routines to accomplish the following five tasks:

1. Flow control and ARQ initialization
2. Inbound pre-ARQ flow control
3. Inbound post-ARQ flow control
4. Outbound pre-ARQ flow control
5. Outbound post-ARQ flow control

The following sections describe in more detail the routines which see to the completion of these tasks.

6.4.2.1 Flow Control and ARQ Initialization (Submodule IP$FLOW$INIT)

This submodule is called for two reasons,

1. for original FLOW data structure initialization, and
2. for reinitialization while the IP is active.

Accordingly, the submodule is comprised of two routines. The first is called by the protocol module during its own initialization phase, the second at CALLEND, again being invoked by the protocol module. The entry points and functions of each are defined below.

Entry Point - IP$FLOW$INIT:ENTRY

Function

Initializes Flow Control and ARQ data structures and creates Inbound and Outbound data buffers.

Entry Conditions

None

Exit Conditions

All registers destroyed.

Entry Point - IP$FLOW$INIT:REINIT

Function

Deletes previous Inbound and Outbound data buffers, reinitializes Flow Control data structures, and creates new Inbound and Outbound buffers.

Entry Conditions

None

Exit Conditions

All registers destroyed.

(In actuality, the first routine is merely a subset of the instructions belonging to the routine performing the reinitialization task.)

6.4.2.2 Inbound Pre-ARQ Flow Control (Submodule IP$FLOW$PXMT)

Function

Called by the Inbound Protocol Module (IBP) at interrupt level on a per character basis. This submodule moves unencoded data into the IB byte queue chain, which is composed of blocks of "Alpha" length.

Entry Points - IP$FLOW$PXMT:ENTRY1
IP$FLOW$PXMT:ENTRY2

Entry Conditions

ENTRY1: A-reg = data byte

ENTRY2: A-reg = 1st data byte
B-reg = 2nd data byte

Exit Conditions

A-reg = preserved
All other registers destroyed.

6.4.2.3 Inbound Post-ARQ Flow Control (Submodule IP$FLOW$XMIT)

Function

Obtains data from the corresponding Inbound Data Buffer. It vectors to ADC with the data to be encoded. The encoder returns with an encoded data byte which is compacted and then written to the BIC IBFIFO. When the BIC IBFIFO is full, FLOW$XMIT enables BIC IBFIFO half empty interrupts and terminates the task. This submodule is initiated when the BIC IBFIFO is able to accept a byte of data. It is responsible for sending ACK ICS's, and for resending unacknowledged blocks, in the event of link or node failure or buffer overflow. In addition, it is responsible for signalling processing speed changes to downstream nodes.

Entry Point - IP$FLOW$XMIT:ENTRY

Entry Conditions

None

Exit Conditions

All registers destroyed.

6.4.2.4 Outbound Pre-ARQ Flow Control (Submodule IP$FLOW$RECV)

Function

Obtain data from the BIC OBFIFO and pass it to ADC to be decoded. The decoder returns the decoded character which is sent to the Outbound Protocol Module (OBP) to be moved to the corresponding Outbound Data Buffer. This submodule is responsbile for re-enabling BIC OBFIFO half full interrupt when the BIC IBFIFO is empty, deleting associated buffer and enabling BIC IBFIFO when a block is acknowledged, and performing outbound error recovery procedures after the receipt of an OVF (buffer overflow) ICS or a KILLFAIL ICS which indicates a link or node failure at the remote data transmitter. EBK (end of block) ICS bytes cause this submodule to enable the BIC IBFIFO and inform FLOW$XMIT to send an ACK ICS, via an escape, escape sequence sent to outbound protocol module.

Entry Point - IP$FLOW$RECV:ENTRY

Entry Conditions

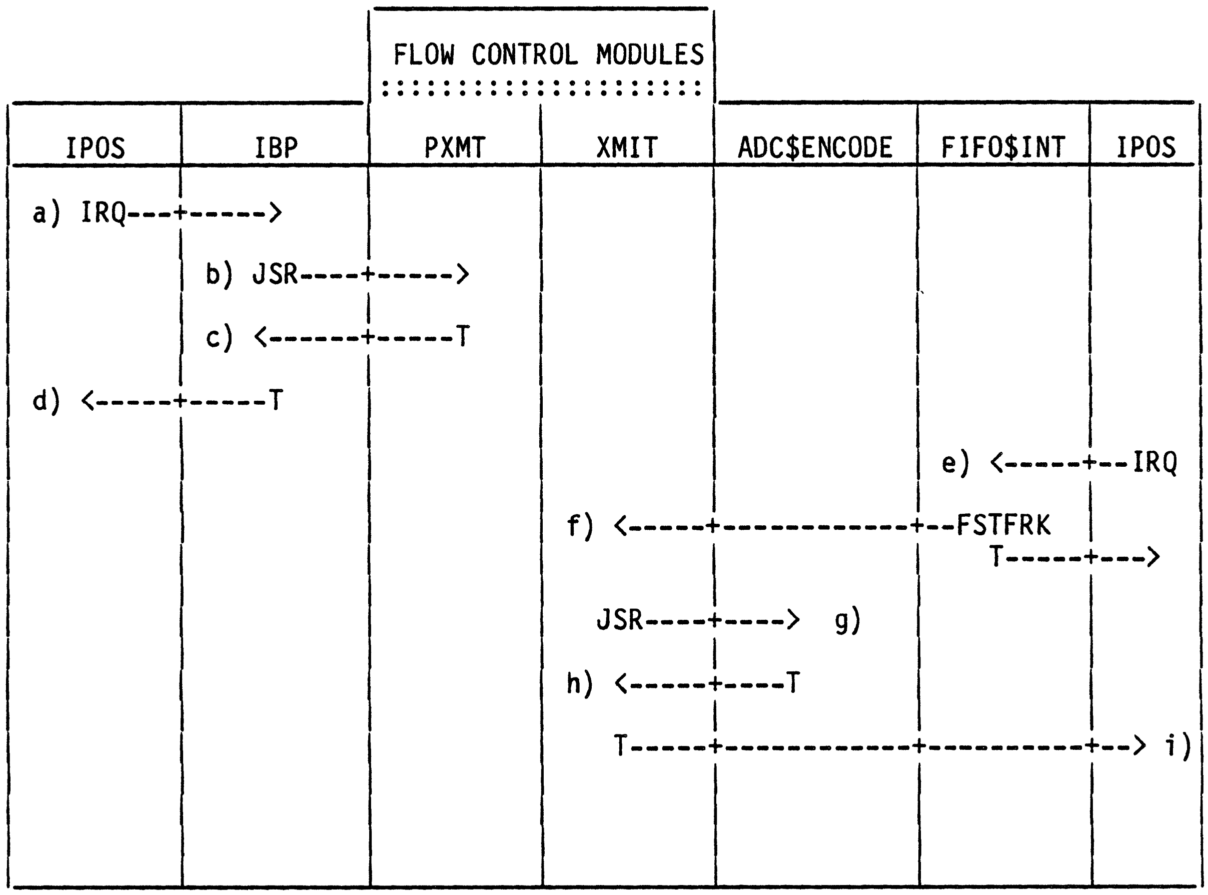
None

Exit Conditions

All registers destroyed.

==

Further understanding of the ARQ and Flow Control method used in the D814 system can be achieved by studying the following diagrams. These illustrate program control flow and briefly describe the processing done in both normal and recovery modes.



Inbound Data -- Normal Transmission

Figure 6.4-1

ARQ-FLOW CONTROL

a) IPOS gets 2651 receiver ready IRQ and vectors to IBP at interrupt level to obtain receiver data byte.

b) IBP calls FLOW$PXMT with data byte.

c) FLOW$PXMT moves the data to the IB ARQ buffer chain.

d) IBP finishes interrupt handling and returns control to IPOS.

e) IPOS gets IB FIFO IRQ, routes it to FIFO$INT.

f) FLOW$XMIT gets data byte from the IB ARQ buffer chain.

g) FLOW$XMIT calls ADC to encode the data byte.

h) ADC$DECODE returns with encoded byte which FLOW$XMIT moves to the BIC IBFIFO and proceeds to step f) as long as IB data buffer is not empty and BIC IBFIFO is not full.

i) FLOW$XMIT terminates the task with IB ARQ buffer chain empty, BIC IB FIFO full, or transmitter metered off.

<u>Note</u>: Where a block in the chain becomes completely transmitted, additional processing takes place (e.g., insertion of ICS's, metering off).

<u>Note</u>: When the flow control transmitter is called with the "send ACK" flag set, FLOW$XMIT merely decrements the flag and writes an ACK ICS to the BIC IBFIFO as data.

```
|   IPOS     |  FIFO$INT  |  FLOW$XMIT  |
|            |            |             |
| a) IRQ---+----->        |             |
|            | FSTFRK---+----->         |
|    <-----+----T         |             |
|            |            |             |
| b) <-----+----------+----T            |
|            |            |             |
| c) IRQ---+----->        |             |
|            | FSTFRK---+----->         |
|    <-----+----T         |             |
|            |            |             |
| d) ------+----------+----T            |
|            |            |             |
| e) IRQ---+----->        |             |
|            | FSTFRK---+----->         |
|    <-----+----T         |             |
|            |            |             |
| f) <-----+----------+----T            |
|            |            |             |
|            |            |             |
```

Inbound Data -- Recovery Processing

Figure 6.4-2

ARQ-FLOW CONTROL

(This sequence is initiated whenever the flow control Transmitter is entered with the transmit recovery flag set to 4. Its function is to retransmit the data blocks pointed to by the recovery buffer pointer.)

a) Transmit Recovery Flag = 3

IPOS gets BIC IBFIFO IRQ and routes it to FIFO$INT. FIFO$INT disables BIC IBFIFO IRQ's, FSTFRKS FLOW$XMIT and terminates task.

b) Recovery Phase 3

FLOW$XMIT sets the transmit recovery flag = 3 and writes a REC ICS to the BIC IBFIFO, to indicate recovery parameters follow.

c) Transmit Recovery Flag = 2

Repeat of a) if FIFO was full after step b), otherwise step c) is omitted and control is continued to step d).

d) Recovery Phase 2

FLOW$XMIT sets the transmit recovery flag = 2 and writes the status of the local free buffer pool to BIC IBFIFO.

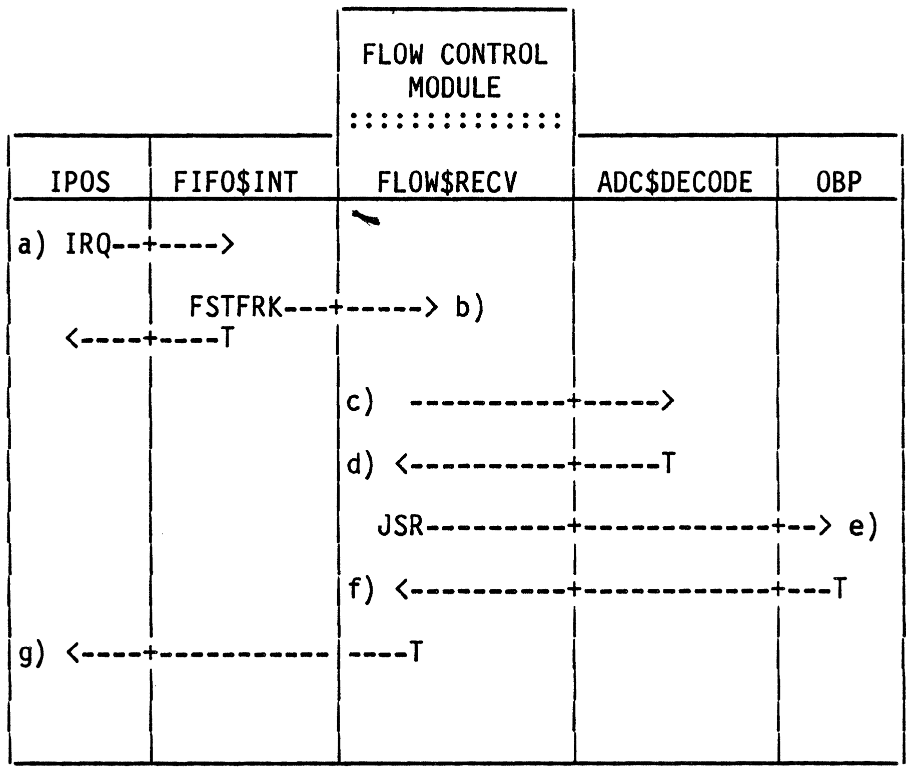
e) Transmit Recovery Flag = 1

Repeat of a) if FIFO was full after step d), otherwise omit step e).

f) Recovery Phase 1

FLOW$XMIT sets the transmit recovery flag = 0 and writes the local transmitter and receiver status byte to the BIC IBFIFO. The local receiver status nibble provides the remote transmitter with the knowledge of the last block transmitted in full. The local transmitter status nibble provides the remote receiver with the knowledge of the last block the remote receiver acknowledged. The current transmit block pointer is set to the recovery block pointer and the BQUE read pointer for this block is reset to the beginning of the block.

FLOW$XMIT is now in normal mode and continuously reads from the current block to be retransmitted and writes each byte to the BIC IBFIFO until the IB data buffer is empty or the BIC IB FIFO is full.



Outbound Data -- Normal Transmission

Figure 6.4-3

ARQ-FLOW CONTROL

a) IPOS gets BIC OBFIFO IRQ, routes it to FIFO$INT. FIFO$INT disables Outbound FIFO interrupts, FSTFRKs FLOW$RECV.

b) FLOW$RECV gets a byte from the OBFIFO and tests the byte received:

   If ACK ICS, updates count for last block ACK'ed, enables the BIC IBFIFO if necessary, deletes byte queue pointed to by the recovery buffer pointer and sets next link as new recovery buffer pointer.

   If EBK ICS, clears count of last character received, updates number of blocks received, and calls OBP through vector with escape, escape sequence.

   If KILLFAIL ICS, sets receive recovery flag = 5.

   If OVF ICS, causes local transmitter recovery initialization and toggles the remote overflow indicator.

   If data:

c) Calls ADC$DECODE to decode on a per nibble basis.

d) ADC$DECODE returns with decoded character or asks for next nibble.

e) FLOW$RECV calls OBP with decoded character.

f) OBP buffers the character in the appropriate OB data buffer.

g) Go to step b) unless BIC OBFIFO is empty. When the FIFO is empty, re-enables BIC OBFIFO interrupts and terminates the task.

```
 ________________________________________________
|                |               |               |
|      IPOS      |   FIFO$INT    |   FLOW$RECV   |
|________________|_______________|_______________|
|                |               |               |
|  a) IRQ---+---->               |               |
|                | FSTFRK---+---->               |
|     <-----+----T               |               |
|                |               |               |
|  b) <-----+-----------+----T                   |
|                |               |               |
|  c) IRQ---+---->               |               |
|                | FSTFRK---+---->               |
|     <-----+----T               |               |
|                |               |               |
|  d) <-----+-----------+----T                   |
|                |               |               |
|  e) IRQ---+---->               |               |
|                | FSTFRK---+---->               |
|     <-----+----T               |               |
|                |               |               |
|  f) <-----+-----------+----T                   |
|                |               |               |
|  g) IRQ---+---->               |               |
|                | FSTFRK---+---->               |
|     <-----+----T               |               |
|                |               |               |
|  h) <-----+-----------+----T                   |
|                |               |               |
|  i) IRQ---+---->               |               |
|                | FSTFRK---+---->               |
|     <-----+----T               |               |
|                |               |               |
|  j) <-----+-----------+----T                   |
|________________|_______________|_______________|
```

Outbound Data -- Recovery Processing

Figure 6.4-4

ARQ-FLOW CONTROL

(This sequence is initiated whenever FLOW$RECV is entered with the receive recovery flag set to 5. Its function is to ignore all data received until the block and character last correctly received is received again.)

a) Receive Recovery Flag = 5

IPOS gets BIC OBFIFO IRQ, routes it to FIFO$INT. FIFO$INT disables BIC OBFIFO IRQ's, FSTFRKS FLOW$RECV and terminates.

b) Recovery Phase 5

FLOW$RECV discards all data until a REC ICS is received, then it sets receive recovery flag = 4.

c) Receive Recovery Flag = 4

Repeat of a) if BIC OBFIFO is empty, otherwise control proceeds to step d).

d) Recovery Phase 4

FLOW$RECV reads parameter byte from BIC OBFIFO and compares the remote buffer pool status against the local version. If unequal, buffer overrun recovery is initiated at the local transmitter; the receive recovery flag is set to 3.

e) Receive Recovery Flag = 3

Repeat of step a) if BIC OBFIFO is empty, otherwise, control proceeds to step f).

f) Recovery Phase 3

FLOW$RECV reads parameter byte from BIC OBFIFO and computes the number of blocks to discard and the number of lost acknowledgements. It deletes the block the recovery pointer is pointing to and transfers the link to the next BQUE to the recovery pointer for each lost ACK. In addition, the number of blocks not outstanding and the number of last block ACK'ed are updated for each lost ACK. Finally the Receive Recovery Flag is set to 2.

g) Receive Recovery Flag = 2

Repeat of step a), if BIC OBFIFO is empty, otherwise, control proceeds to step h).

h) Recovery Phase 2

FLOW$RECV gets data byte from the BIC OBFIFO.

If EBK ICS, tests the number of blocks to discard. If equal to zero, post end of recovery since EBK was the last byte received properly. If the number of blocks to discard was not equal to zero, the number is decremented and control returns to loop check the FIFO for more data.

If data byte, and the number of blocks to discard is non-zero, the byte is discarded and control returns to loop check the FIFO for more data.

If data byte and the number of blocks to discard is now equal to zero, the number of nibbles received properly prior to failure becomes the number of nibbles to discard, and the receive recovery flag is set to one. If there were no nibbles received this block, the receive recovery flag is reset and the data is buffered normally.

i) Receive Recovery Flag = 1

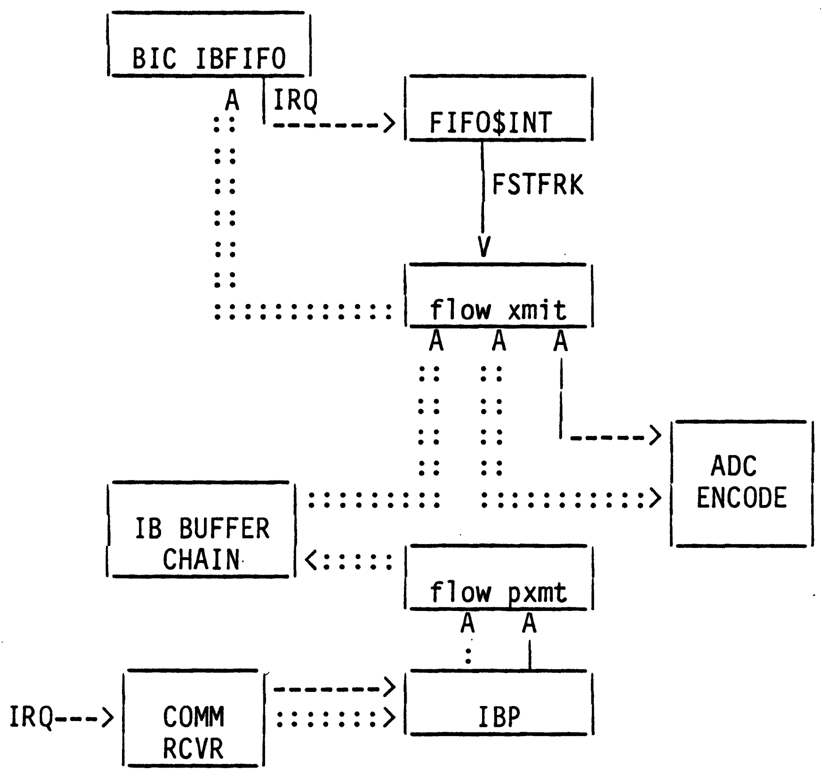
Repeat of a) if BIC OBFIFO is empty, otherwise, control proceeds to step j).

j) Recovery Phase 1

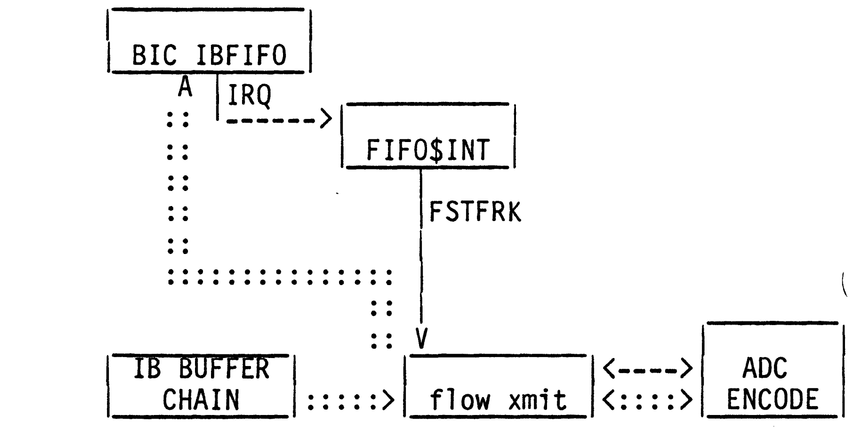
FLOW$RECV gets data byte from BIC OBFIFO and updates the count of nibbles to discard for a single or double data nibble. If the count remains non-zero, control returns to loop check the FIFO for more data. When the count of nibbles to discard becomes zero the receive recovery flag is set to zero. If the count becomes zero on the first half of a double data nibble, only the first half is discarded, the high order nibble is shifted to the low order position and is buffered as good data.

==

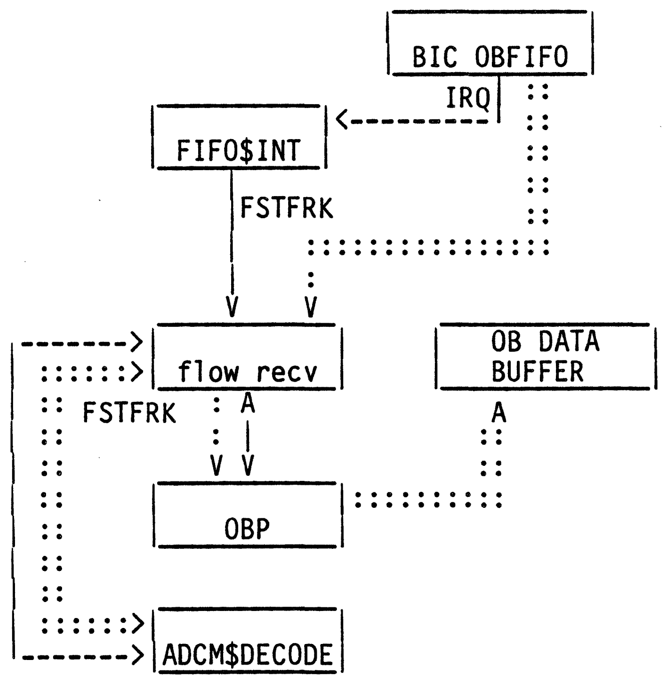
In order to delineate the path of data transfer for quick reference, the following diagrams have been supplied. In each, "::::>" indicates data movement; "--->" shows a change in program control.



ARQ-FLOW CONTROL Operation -- INBOUND



ARQ-FLOW CONTROL Operation -- INBOUND RECOVERY



ARQ-FLOW CONTROL Operation -- OUTBOUND

6.4.3 Adaptive Data Compression Scheme

Following is a complete description of the adaptive data compression algorithm, including timing estimates. There are 5 parts to this section:

1) General Description
2) Transmitter
3) Receiver
4) Initialization
5) Timing Estimates

6.4.3.1 General Description of the Algorithm

The D814 data compression algorithm attempts to encode frequent characters into short codewords, and less frequent characters into larger codewords, so as to reduce the average number of bits used per character. It is adaptive in that it does not require a prior statistical description of the source but will search for a code matched to the source. This objective requires two distinct actions.

First, the characters must be kept ordered by relative frequencies (high frequency <-> low rank). This is done simply by exchanging the ranks of the ith ranked and (i-1)th ranked characters when the ith ranked character occurs. The precise algorithm is given in Section 6.4.3.2.2.

Secondly, the best code must be computed and a method must be found to encode and decode. This section describes the code structure. The encoding and decoding algorithms are explained in Sections 6.4.3.2.3 and 6.4.3.3.3.

For ease of implementation, codewords are required to have a length of 4, 8 or 12 bits (1, 2 or 3 nibbles). Moreover, the 0 nibble is forbidden, as it is reserved for control purposes at lower levels in the network structure.

We can view the code as a 15-ary tree, with branches labeled 1 to F and leaves at levels 1, 2 or 3 (Figure 6.4.3.1).

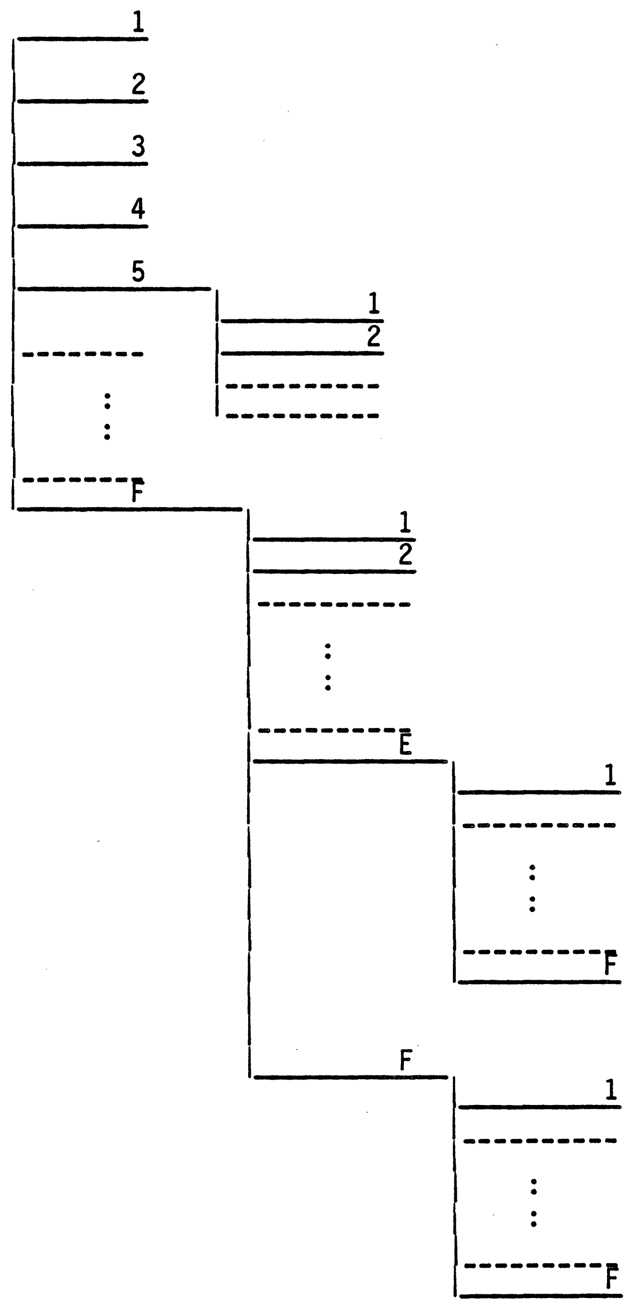


Figure 6.4.3.1

N4, N8 and N12 denote the numbers of possible leaves (not necessarily actual characters) at levels 1, 2 and 3 respectively. We want the tree to be complete (all nodes have 15 outgoing branches) so that N4 + N8 + N12 has the form 15 + 14k, for some integer k, chosen so that N4 + N8 + N12 is at least as large as the number of symbols in the source alphabet.

Also, as the code is complete, Kraft equality must be satisfied, thus N4 (15^{-1}) + N8 (15^{-2}) + N12 $(15^{-3}) = 1$. This relation, together with the one about N4 + N8 + N12, leaves only one degree of freedom for the tree. We choose N4 as that parameter.

From the previous relations, one can see that every time N4 is increased by 1, N8 decreases by 16 and N12 increases by 15. This suggests that if the most likely character encoded into an 8 bit codeword occurs more frequently than the 15 least likely characters encoded into 8 bit codewords, then N4 should be increased so as to reduce the average codeword length. Similarly, if the 15 most likely characters encoded ito 12 bit codewords occur more frequently than the least likely character encoded in a 4 bit codeword, then N4 should be decreased. The optimality of this search rule in the quest for the best code can be established by means of a convexity argument. The precise implementation is described in Section 6.4.3.2.3.

From the previous discussion, we should maintain:

DRIFT1 =	# occurrences of least likely 1 nibble character, minus # of occurrences of 15 most likely 3 nibble characters.
DRIFT2 =	# occurrences of 15 least likely 2 nibble character, minus # occurrences of most likely 2 nibble character.

when DRIFT1 reaches 0, N4 should be decreased
DRIFT2 reaches 0, N4 should be increased

Instead, the algorithm maintains DRIFT=DRIFT1-DRIFT2.

when it underflows, N4 should be decreased
overflows, N4 should be increased

This increases the speed of convergence, but has the disadvantage that no steady state code can be reached. This is not a problem, however, as a source is never really stationary.

6.4.3.2 Transmitter (IP$ADCM$ENCODE:ENTRY)

The transmitter forms a single subroutine. It is entered with the actual character in Register B, and terminates with the codeword in Registers A and B as follows:

	A	B
1 nibble:	0 0	0 H
2 nibble:	0 0	H M
3 nibble:	0 H	M L

It then jumps to the routine that puts the nibbles in the inbound buffer.

<u>Entry Point</u> - IP$ACDM$ENCODE:ENTRY

<u>Entry Conditions</u>

B Reg contains character to be encoded

The detailed description follows. For simplicity, we divide the transmitter into 3 well defined routines that that are described separately. They are:

Transmitter Code Update
Transmitter Rank Update
Transmitter Encoding

6.4.3.2.1 Transmitter Code Updating

This is the first transmitter routine. It starts and ends with the actual character to be encoded in Register A. In between it checks if the code must be changed due to the transmission of the previous character by comparing DRIFT with 128 and 0.

If a code update is necessary, the following parameters must be computed:

N4
N4 + N8 - 15
N4 + N8
N4 + N8 + 15
- N12 - 16
EORF
(N4 + 1) 15
$(N4 + 1)\ 15 + 2 + \left[\frac{N8 - 1}{15}\right]$

where [] denote integer part.

In addition, DRIFT must be reset to 64. N4 + N8 + N12 is set at initialization and never changes. The algorithm uses the old values of N4 and N4 + N8 - 15 in order to compute the new ones. It then recomputes all other parameters from N4, N4 + N8 - 15 and N4 + N8 + N12.

Computations are trivial, except the one of

$$\left[\frac{N8 - 1}{15}\right]$$

If $N8 = a\,(16) + b = a\,(15) + (a + b)$
(note that $a + b < 30$, as $N8 < 255$)

then $\left[\frac{N8 - 1}{15}\right] = a$ if $a + b < 16$

$= a + 1$ if $a + b \geq 16$

$\left[\frac{N8 - 1}{15}\right]$ is computed as follows:

put N8 in Reg B
transfer Reg B to Reg A
shift Reg B to the right 4 times. Now Reg B = a.
add Reg B to Reg A
If the result is more than 15, increment Register B

<u>Timing</u> (# Cycles)

If DRIFT=0 or 128, no change necessary, 7 cycles.

If DRIFT=0 or 128 but
DRIFT=0 and N4 already 0
or DRIFT=128 and N8 already <16 (i.e., (N4 + N8 -15) <15) then no change is actually made, only 48 cycles are used.

If a change must be made, it requires 162 cycles.

6.4.3.2.2 Transmitter Rank Update Routine

This routine starts with a character in Register A. It updates tables as explained below and finishes with the character rank in Register B.

The transmitter maintains 2 tables to keep track of the ordering of the characters. Tables are aligned with page boundaries.

P-table:	Indexed by character number. Contains the rank of a character, with very frequent characters being low rank.
IP-table:	Indexed by rank. Its ith element contains the name of the ith ranked character.

When character NCHAR occurs, the routine consults the P-table to find its rank PN, which is used by the transmitter encoding module. If PN is not zero, NCHAR is swapped with the next most likely character. Thus,

```
P(NCHAR)     <---PN-1
P(IP(PN-1)) <---PN
IP(PN)       <---IP(PN-1)
IP(PN-1)     <---NCHAR
```

In this way the most likely characters tend to drift to low rank positions.

Timing (# cycles)

Normal character:	76
Character with rank 0:	19

6.4.3.2.3 Transmitter Encoding

This routine starts with the character rank in Register B. It computes the codeword and checks if the code should be changed later. It terminates with the codeword in Registers A and B and returns to the calling program.

Timing (# Cycles)

4 bit codeword	normal	20
	least likely	30
8 bit codeword	most likely	34
	normal (15/16)	40
	normal (1/16)	56
	15 least likely (15/16)	50
	15 least likely (1/16)	66
12 bit codeword	15 most likely (15/16)	50
	15 most likely (1/16)	60
	normal (15/16)	40
	normal (1/16)	66

Following is a detailed explanation. Numbers refer to the tests in the flow chart.

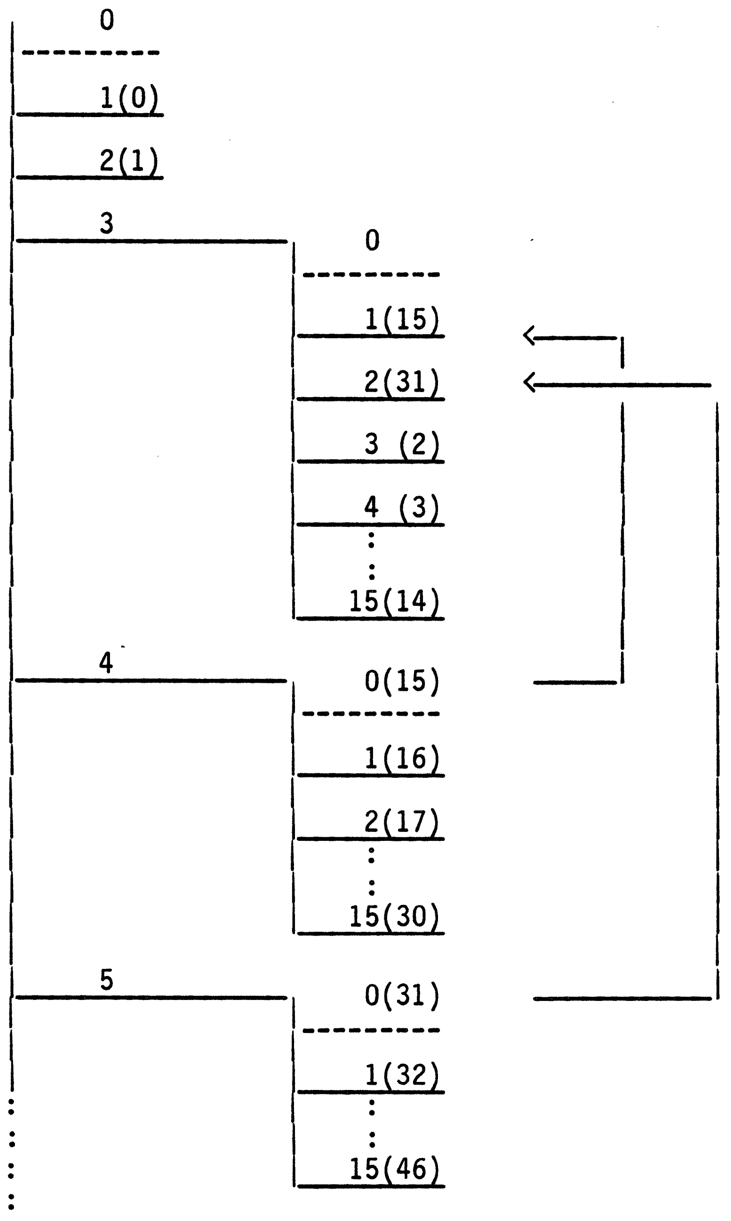
N4 = # of 1 nibble codewords
N8 = # of 2 nibble codewords in the completed code tree.
N12 = # of 3 nibble codewords in the completed code tree. Note that N12 is always a multiple of 15.

1) Compare B with N4. If less, then B is encoded into a 4 bit codeword by incrementing it. If it is the least likely 4 bit codeword, DRIFT is increased.

2) Uses the Z flag from 1. If set, B corresponds to the most likely 8 bit codeword and DRIFT is increased.

3) Compare B with N4 + N8. If less, it corresponds to an 8 bit codeword, else to a 12 bit codeword

4) Check if B corresponds to one of the 15 least likely codewords. If so, DRIFT is decreased. Note that if N8 is less than 16, the most likely 8 bit codeword is also one of the 15 least likely 8 bit codewords. To save processing, this is not checked here, but is taken care of by the transmitter code update routine which will not change the code if DRIFT=128 but N8 is less than 16.



i nibble #
(i) character rank
---- illegal nibble

Figure 6.4.3.2

Example: N4 = 2

$$\left\lceil \frac{N8 - 1}{15} \right\rceil = 2$$

An 8 bit codeword is generated by adding B to a base, then checking if the resulting codeword terminates with the 0 nibble. If so, the codeword is "folded", see Figure 6.4.3.2. One can show that the number of folded codewords is the integer part of

$$\left[\frac{N8 - 1}{15}\right] .$$

Thus the most likely 8 bit codeword (B = N4) is encoded into

$$(N4 + 1)\ 16 + 1 + \left[\frac{N8 - 1}{15}\right] .$$

so that base to which B should be added is

$$(N4 + 1)\ 15 + 2 + \left[\frac{N8 - 1}{15}\right] .$$

5) If the codeword must be folded, it is shifted right four times then added to (N4 +1) 15. It is guaranteed that a folded codeword will never terminate with the 0 nibble, as

$$\left[\frac{N8 - 1}{15}\right] \leq 15$$

6) We must generate a 12 bit codeword. If it is one of the most 15 likely, DRIFT is decremented.

The 8 LSB of the codeword are determined by subtracting (N4 + N8 + N12), so that if B is equal to (N4 + N8 + N12) -1 (i.e., the least likely character in the completed alphabet), the 8 LSB of the codeword are FF. The 4 MSB of the result are never 0000, as $N12 \leq 240$ and $B \geq N4 + N8$, thus $(256 +)\ B - (N4 + N8 + N12) \geq 16$. The 4 LSB can be 0000, thus leading to an illegal codeword.

7) If the 4 LSB are 0000, B is shifted right 4 times, and the result is added to X, where X = 240 if N12 = 240, and X = -N12 -16 otherwise. Thus if the result of the subtraction (in 6) is F0 (i.e., the least likely unlawful 12 bit codeword), the resulting codeword is 15 - N12 - 16 = (N4 + N8 -1) - (N4 + N8 + N12), i.e. just above the most likely 3 nibble codeword (N4 + N8) - (N4 + N8 + N12).

The most significant nibble of 3 nibble codewords is always F, except if Test 7 is true and N12 = 240, in which case it is E. EORF contains 0 or 1, depending on N12.

6.4.3.3 Receiver (IP$ADCM$DECODE:ENTRY)

The receiver consists of one subroutine. It is entered with a non zero nibble in the lower part of Register A. It returns either with Z = 0, in which case no complete codeword has been received, or with A cleared and Z = 1, and a character in B.

Entry Point - IP$ADCM$DECODE:ENTRY

Entry Conditions

A Reg with non-zero nibble to be decoded in lower part

Exit Conditions

If complete codeword received:

```
A Reg = 0
B Reg = decoded character
X Reg = destroyed
Z     = 0
```

If not complete codeword:

```
A Reg = destroyed
B Reg = destroyed
X Reg = intact
Z     = 1
```

The detailed description follows. For simplicity we divide the receiver into 3 well defined routines that are described separately. They are:

Receiver Decoder
Receiver Rank Update
Receiver Code Update

6.4.3.3.1 Receiver Decoder Routine

This routine starts with a nibble in Register A, and decodes it. If a complete codeword has been received, the rank of the character is placed in Register A and the program continues with the receiver rank update routine, else control is returned to the calling program.

Timing (# Cycles)

			First Nibble	Second Nibble	Third Nibble
4 bit	c.w.	normal	24		
		least likely	32		
8 bit	c.w.	most likely	21	60	
		normal (15/16)	21	56	
		normal (1/16)	21	68	
		15 least likely (15/16)	21	66	
		15 least likely (1/16)	21	78	
12 bit	c.w.	15 most likely (15/16)	21	27	64
		15 most likely (1/16)	21	27	72/76
		normal (15/16)	21	27	54
		normal (1/16)	21	27	62/66

The variable STATE is defined as follows:

STATE = (00) if first nibble H is being processed
(0H) if second nibble M is being processed
(HM) if third nibble L is being processed
Note: If STATE = (H,M) then H = E or F and bit 7 always 1.

Following are the details of the algorithm.

1) If the first nibble $\leq$ N4, we have the complete word. If =N4, it is the least likely 4 bit codeword and DRIFT1 must be incremented. The rank is obtained by decreasing by 1.

2) Compute (H,M). The result is compared with

$$(N4 + 1)\ 16 + N8 + \left\lceil \frac{N8 - 1}{15} \right\rceil ,$$

which is the least likely 8 bit codeword or an illegal codeword smaller than the most significant 8 bits of any 12 bit codeword. This test determines whether we have a complete 8 bit codeword.

3) Compare with $$(N4 + 1)\ 16 + 1 + \left\lceil \frac{N8 - 1}{15} \right\rceil .$$

which is the most likely 8 bit codeword.

If =, it is the most likely 8 bit codeword. Decrease DRIFT2 and set the rank to N4.

If less than, we undo the second phase of the encoding by adding N4 + 1 and multiplying by 16. We thus get the illegal codeword that was the output of the first phase of the encoding.

We undo the first phase of the encoding by subtracting

$$(N4 + 1)\ 15 + 2 + \left[\frac{N8 - 1}{15} \right]$$

and obtain the rank.

4) If one of the 15 least likely 12 bit codewords, increase DRIFT2.

5) Check if H was E or F and compute the least significant byte of the 3 nibble codeword.

6) Compares that byte with the final byte of the most likely 12 bit codeword, N4 + N8 - (N4 + N8 + N12) = - N12.

   A "less than" result indicates that the first encoding operation resulted in an illegal codeword. We can recover that illegal codeword by subtracting -N12 and multiplying by 16. The first phase of the encoding is undone by adding N4 + N8 + N12.

7) Finally DRIFT1 is updated if the codeword is one of the 15 most likely 12 bit codewords.

6.4.3.3.2 Receiver Rank Update Routine

This routine starts with a rank in Register A. It updates a table as explained below, computes the character corresponding to the rank, places it in Register B and returns to the calling program with Register A cleared.

The receiver contains only one table, the IP-table. It is updated exactly as in the transmitter. Thus, if rank PN occurs and is not zero, IP(PN) and IP(PN-1) are swapped, and the old value of IP(PN) is the output.

Timing (# cycles)

Normal	:	41
If rank =0	:	12

6.4.3.3.3 Receiver Code Update

This routine performs a role similar to the Transmitter Code Update routine, using the same algorithm and about the same number of cycles.

If a code update is necessary, the following parameters must be computed:

N4
N4 + N8 + 15
N4 + N8 - 15
- N12

$$(N4 + 1)\ 16 + 1 + \left[\frac{N8 - 1}{15}\right]$$

$$(N4 + 1)\ 15 + 2 + \left[\frac{N8 - 1}{15}\right]$$

$$(N4 + 1)\ 16 + N8 + \left[\frac{N8 - 1}{15}\right]$$

DRIFT must be reset.

The routine has two entries, DECN4 and INCN4, depending whether N4 should be decreased or increased.

6.4.3.4 Initialization (IP$ADCM$INIT:ENTRY)

The following parameters must be initialized.

6.4.3.4.1 Transmitter

Alphabet Size	32	64	128	256
Parameters				
N4	13	12	10	0
N4 + N8 - 15	28	41	66	207
N4 + N8 + N12	43	71	141	11 (= 267)
DRIFT	128	128	128	128
P table	Natural Numbering, 255, 0, ... 254			
IP table	Natural Numbering, 1, ... 255, 0			

6.4.3.4.2 Receiver

The initialization is the same as for the transmitter, except that the P table is not defined for the receiver, but STATE must be set to 0.

In addition the routine INCN4 must be executed to compute all the parameters used in the decoder routine. The similar operations in the transmitter routine will be executed automatically when the first character is encoded, as DRIFT was initialized to 128.

6.4.3.5 Timing Estimates (# Cycles)

Estimates regarding the transmitter encoder and receiver decoder routines have been obtained by adding 10 to Linde's original estimates.

<u>Transmitter</u>			
Code Update			11
Rank Update			76
Encoder (8 bit code)	Uniform alphabet		44
	Skewed alphabet		32
		Total:	119-131
<u>Receiver</u>			
Code Update			1
Decoder (8 bit code)	Uniform alphabet		92
	Skewed alphabet		57
Rank Update			41
		Total:	134-150
<u>Grand Total</u> (Transmit and Receive):			243-281

With a full-duplex 1200 cps terminal, this would mean 336×10^3 cycles per second, or 17 percent of the processor capacity. The previous figures are somewhat pessimistic, simulation results on actual files, averaged over 1000 characters, hover between 93 cycles/character when long strings of 0's are transmitted and 250 cycles/character for files containing object code, with 210 cycles/character being typical (13 percent of processor capacity).

6.5 Multi-Threaded Data Movement

Multi-threaded data movement refers to the transfer of data between terminals over a line that is effectively shared. In the D814 system, this implies that many physical terminals share a single set of I/TP software, hence messages are interleaved.

The D814 is responsible for upholding data integrity during its movement across the network. It does not, however, use an explicit end-to-end error detection scheme in order to maintain acceptable levels of data accuracy. Instead, an ARQ mechanism has been designed which is activated when a link or a node failure has been detected. This, in conjunction with the checking done by the INP, insures minimal data loss across the link.

The ARQ and flow control measures are implemented at those points at which data is transferred to and from the BIC. The modules which are responsible for this are discussed in the following two subsections.

6.5.1 BIC FIFO Handler

The multi-threaded data movement module uses the same BIC handler as the single-threaded data movement described in Section 6.4.1.

6.5.2 Flow Module (IP$MFLOW)

The multithreaded data flow module is responsible for multiplexing the data streams of individual threads into the multithreaded data stream format, MTHSDI (described in Section 3.2.3.2 of this specification).

It is comprised of routines to accomplish the following tasks:

1. Flow module initialization
2. Pre-transmit
3. Transmit
4. Receive

The following sections describe in more detail the routines which see to the completion of these tasks.

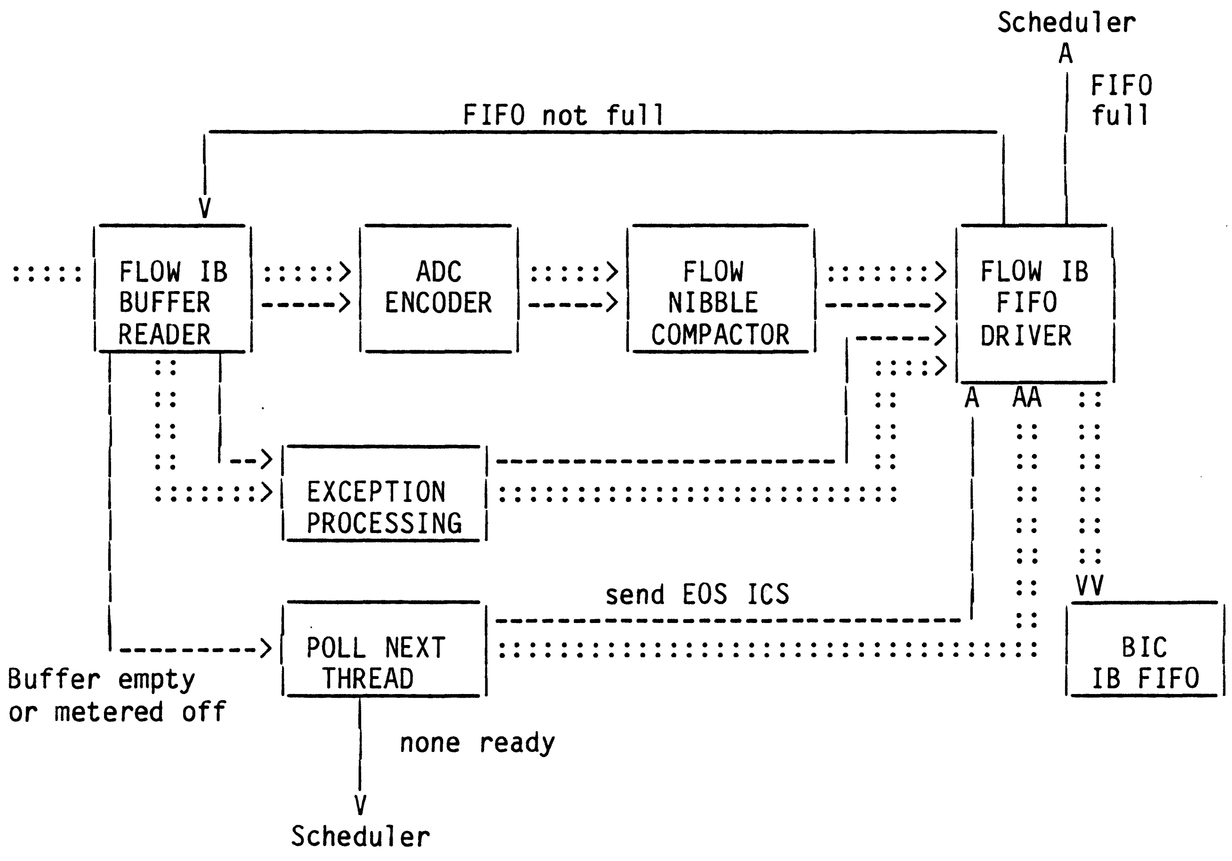
Inbound Flow Interface

(Section I)

```
                                                  Thread          _
                                                       1      ____-||
                                                                  |    ||
                                                  :::::::::>|    |:::::::::
                                                  ::          |____|        ::
Thread                                            ::                 _      ::
1                                                 ::              ____-||   ::
 ::::::::                                         :: 2        |    ||   ::
        ::                  ____       ____       :::::::::>|    |::::::::
2       ::                 |    |     |    |      ::          |____|        ::
 ::::::::::::::>| IB |----->| FLOW |::::::::                             :::::::::
        ::                 |PROT|:::::>| PXMT |     ::     Individual      ::
16      ::                 |____|     |______|     ::      IB BQUE        ::
 ::::::::                                         ::       Chains         ::
                                                  ::                       ::
                                                  ::                 _     ::
                                                  ::              ____-||  ::
                                                  :: 16       |    ||   ::
                                                  :::::::::>|    |::::::::
                                                              |____|
```

Inbound Flow Interface

(Section II)



6.5.2.1 Flow Initialization (Submodule IP$MFLOW$INIT)

This submodule is called for two reasons,

1. for original FLOW data structure initialization, and
2. for reinitialization while the IP is active.

Accordingly, the submodule is comprised of two routines. The first is called by the protocol module during its own initialization phase, the second at CALLEND, again being invoked by the protocol module. The entry points and functions of each are defined below.

Entry Point - IP$MFLOW$INIT:ENTRY

Function

Initializes data structures common to physical port and then uses the thread index table to locate the thread structure addresses of each thread and sequentially initializes Flow Control and ARQ data structures and creates Inbound and Outbound data buffers for each potential thread which may ultimately be connected to this port.

Entry Conditions

None

Exit Conditions

All registers destroyed.

Entry Point - IP$MFLOW$INIT:REINIT

Function

Deletes previous Inbound chain and Outbound data buffers, reinitializes Flow Control data structures, and creates new Inbound and Outbound buffers for the thread at CALLEND.

Entry Conditions

Y-Reg - points to the thread at CALLEND

Exit Conditions

All registers destroyed.

6.5.2.2 Inbound Pre-ARQ Flow Control (Submodule IP$MFLOW$PXMT)

Function

Called by the Inbound protocol module when the DTE has generated a character. This submodule moves the data into the Inbound byte queue chain. In addition, it performs the function of chaining individual byte queues, associated with a particular thread, and for maintaining the size of individual byte queues to the block size dynamically determined by the network in the form of the network variable "alpha".

Entry Points - IP$MFLOW$PXMT:ENTRY1
IP$MFLOW$PXMT:ENTRY2

Entry Conditions

```
ENTRY1:  A-Reg = data byte
ENTRY2:  A-Reg = MS data byte
         B-Reg = LS data byte
         Y-Reg = Thread structure address
```

Exit Conditions

```
A, B, X-Reg = destroyed
      Y-Reg = preserved
```

6.5.2.3 Inbound Post-ARQ Flow Control (Submodule IP$MFLOW$XMIT)

Function

Obtains data from the Inbound byte queue chain corresponding to the thread polled, calls the data compression encoder and places the resulting codeword in the BIC IBFIFO. This submodule is initiated when the BIC IBFIFO is able to accept at least a half FIFO of data, and is structured as a loop as typically it will transfer many bytes in one call. It terminates when either all the Inbound buffers are empty or are metered OFF, or when the IBFIFO is full.

Secondary functions include the retransmission of unacknowledged blocks in the event of link or node failures or buffer overflow, and the transmission of ICS's.

Its function can be decomposed in 5 submodules:

- Inbound Buffer Reader
- Exception Processing
- Next Thread
- Compactor
- IBFIFO Handler

6.5.2.3.1 Inbound Buffer Reader Submodule

Entry Point - IP$MFLOW$XMIT:ENTRY

Entry Conditions

None

Exit Conditions

A-Reg = Data to encode
Y-Reg = Thread structure index

The Inbound Buffer Reader submodule first checks if an exception condition exists, in which case control is passed to the Exception processing submodule.

Its main function is to pass a data byte to the adaptive encoder module. The data byte is either a leftover form the previous use of the compactor submodule or is obtained from the IB data buffer.

If none is available, control is transferred to the "Next Thread" submodule.

6.5.2.3.2 Exception Processing Submodule

This submodule is normally called by the Inbound Buffer Reader submodule when an exception condition is detected. Processing the exception can result in:

```
- Generating an ICS_ACK
                  _SPDUP or SPDDN
                  _BUFOV
                  _REC
- Sending a slot address
- Sending recovery parameters
- Reinitializing submodules in case of failure detection
```

In addition, the submodule interfaces with the Call Manager, statistics and outbound protocol modules, as those can request the transmission of ICS's SPDUP, SPDDN, REC and ACK. The processing of those requests consist in setting internal flags. The following entries should be used:

Entry Point - IP$MFLOW$XMIT: SPEED

Entry Conditions

A-Reg set to the number of ICS_SPDUP (if > 0) or ICS_SPDDN (if < 0) to be sent.

Exit Conditions

A-Reg	= Wiped out
Y, B, X-Reg	= Unchanged

Entry Point - IP$MFLOW$XMIT:ACK

Entry Conditions

None

Exit Conditions

A-Reg	= Wiped out
Y, B, X-Reg	= Unchanged

6.5.2.3.3 Next Thread Submodule

This submodule is called by the Inbound Reader submodule when a buffer is empty or metered OFF. It determines from which thread data should be sent next. It then generates the ICS_End of Slot and passes it to the IBFIFO Handler submodule. It sets the exception condition and prepares the Next Slot address for future use by the IB BIC FIFO driven routines.

6.5.2.3.4 Compactor Submodule

This submodule processes the return from the adaptive encoder module. It updates the number of encoded nibbles and removes as many 0 nibbles as possible before passing control to the IBFIFO Handler. Its point of entry depends on the number of nibbles in the codeword.

Entry Point - IP$MFLOW$XMIT:SNGL
IP$MFLOW$XMIT:DBL
IP$MFLOW$XMIT:TRPL

Entry Conditions

SNGL and DBL:	A-Reg is data
TRPL :	A-Reg is MS nibble
:	B-Reg is LS byte

6.5.2.3.5 IB FIFO Handler Submodule

Called by the compactor, exception processing and next thread submodules. Writes the data in the IBFIFO. Returns control to scheduler if FIFO full, else to the IB buffer reader submodule.

6.5.2.4 Receive (Submodule IP$MFLOW$RECV)

Function

Obtains data from the OBFIFO, calls the data compression decoder and calls the protocol module. This submodule is initiated when the BIC OBFIFO contains data and is structured as a loop, as typically it will transfer many bytes in one call. It terminates when the OBFIFO is empty.

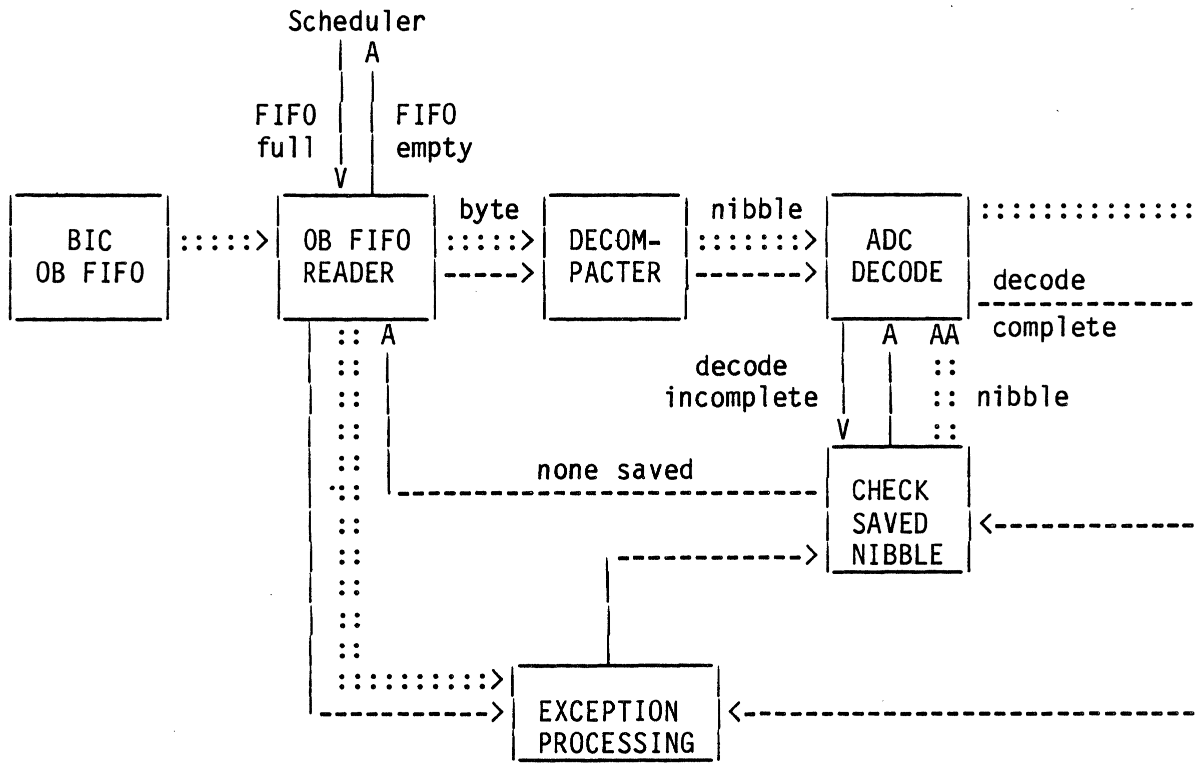
Secondary functions include the orderly processing of retransmitted data.

Its function can be decomposed in 5 submodules:

- Outbound FIFO Reader
- Decompactor
- More Nibble
- Post Exception Check
- Exception Processing

Outbound Flow Interface

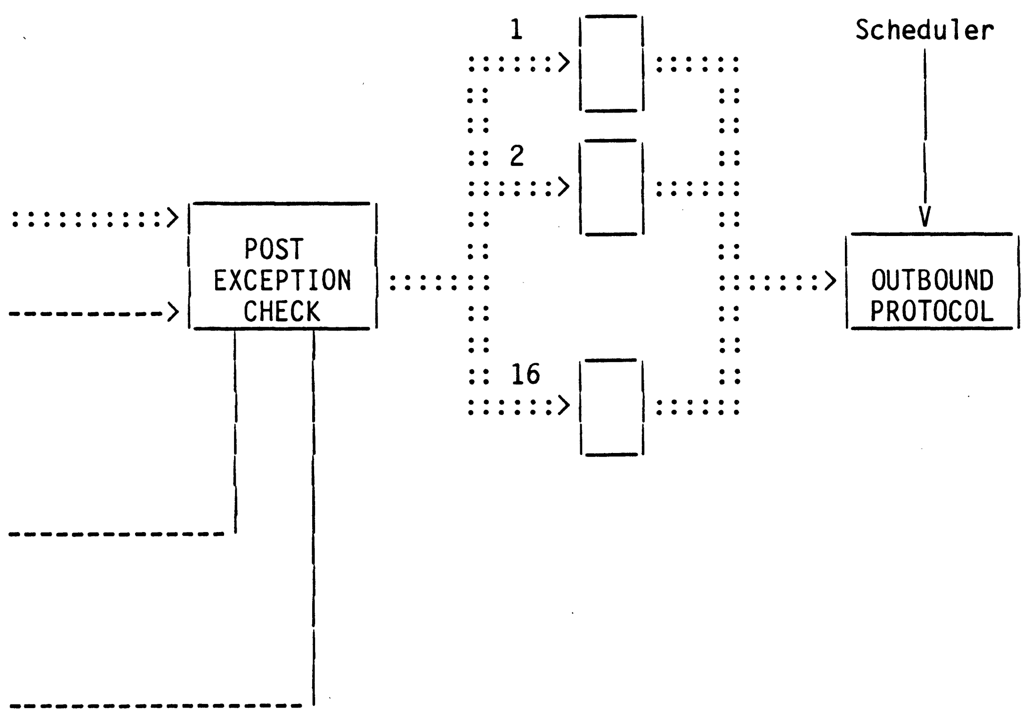
(Section I)



```
:::::> data
-----> control
```

Outbound Flow Interface

(Section II)



6.5.2.4.1 Outbound FIFO Reader

Entry Point - IP$MFLOW$RECV:ENTRY

Entry Conditions

None

Exit Conditions

A-Reg = Data from FIFO
Y-Reg = Thread structure address

The Outbound buffer reader submodule takes data from the OBFIFO and passes control to the data decompactor if no exception condition exists, else to the exception processing submodule. If no data is present, it returns control to the scheduler, after enabling Outbound interrupts from the DTE.

6.5.2.4.2 Decompactor

The decompactor disassembles bytes into nibbles and calls the adaptive data compression decoder.

6.5.2.4.3 Check Saved Nibble

Entry Point - IP$MFLOW$RECV:MORE

Entry Conditions

Y-Reg = Thread structure index

Exit Conditions

Y-Reg = Thread structure index
A-Reg = Nibble to decode

This submodule is called either by the ADC decode or the protocol module when the previous nibble has been processed. If there is an outstanding nibble in the decompactor, it calls the ADC decode, else the Outbound FIFO reader.

6.5.2.4.4 Post Exception Check

Entry Point - IP$MFLOW$RECV:CHAR

Entry Conditions

A-Reg = Decoded character

This submodule is called by the ADC decode when a character has been completely decoded. If the receiver is in retransmission mode, control is passed to the Exception Processing submodule, else to check for a saved nibble.

6.5.2.4.5 Exception Processing

This submodule is called when an exception condition (Retransmission, ICS or Slot Address Present) is met.

6.6 Intelligent Control Terminal Port (I/CTP)

The I/CTP is organized as a set of functionally independent table-driven submodules which support the operator interface to the D814 Network. As such, its primary goal is flexibility of structure and external clarity.

The I/CTP modules perform the following functions:

1. Operator command processing
2. Report control
3. Statistics control
4. System service monitoring
5. Protocol control
6. Device control

6.6.1 Output

The I/CTP supports two output devices, a terminal or display screen which is always present and an optional line or character printer. Either device may run at speeds ranging from 75 baud to 19.2K baud.

The I/CTP supports a hardcopy mode in which all terminal input and output are displayed at the printer. Statistics, reports, and messages may be optionally printed at either device or both. When statistics, messages, or reports are being displayed at the line printer, if the hardcopy mode is also in effect, only one copy of the text is produced at the line printer.

In hardcopy mode, control character inputs are displayed at the line printer as "CTL-X" except for carriage return and line feed. Editing functions (character delete, line delete) are performed before the line is sent to the printer.

The following defines the available input/output options:

1. Statistics

   a) repetitive (automatic, periodic)

      Printer required. Always displayed at line printer, never at terminal.

   b) single (one-time request)

      Always displayed at terminal. May also be displayed at printer.

2. Reports

   Always displayed at printer if printer exists. Displayed at terminal if no printer or as explicit option.

3. Messages

   Same as reports.

6.6.2 Operator Command Processor

The I/CTP receives operator commands (see D814 Program Product Specification) from an asynchronous terminal and interprets them. Command interpretation is defined in the Network Control Language table (NCL) which contains command parameters as expected from a terminal, including all optional valid abbreviations. The NCL is based on the Language Descriptor Table (LDT) which contains the lists of operator commands, command control bytes for internal usage, and routine address pointers for command interpretation and execution.

Command parameters are processed one-by-one with value verifications performed on those parameters which require data. Unrecognized commands (a command not found in the LDT) are rejected with a bell rung at the operator console. Invalid parameters (parameters not found in the NCL) are rejected with an indication of the specific invalid keyword parameter.

As commands are parsed, they place the I/CTP command interpreter at different command context levels. Each command level has one or more subcommands which are acceptable at that level. Thus a complete command may be entered on one line at the terminal or as a series of commands where each subsequent command places the I/CTP a further level down. This is analogous to an N-level tree structure where every node specifies both an action to the operator screen and a tree substructure. The minimal action at nodes other than the lowest is to display the context level at which the command just entered places the interpreter.

Certain control and special character sequences from the terminal are interpreted specially by the I/CTP. Character deletion and line deletion are supported. Other control sequences allow movement, one or more levels, along the command tree structure (including return to the top) and along parallel nodes of the same level.

Additionally, a command inquiry mode is supported. Command-lines, terminated with the inquiry character, produce text explanations of valid command parameters.

The commands available at the top level are:

1. Message (MSG) - Send operator text to another control port or datagram port.

2. Statistics (STAT) - Request statistics from a network port.

3. Configuration Editor (CED) - Modify or display network configuration information.

4. System Services (SS) - Perform operator services and utilities.

5. Mode - Set I/CTP command mode. All commands are not available from all modes.

6. Reset - Reboot the I/CTP to an operator-defined speed.

7. Report (REP) - Print all queued reports.

8. Boot - Reboot the local node.

6.6.3 Report Control

The D814 nodes and ports have the facility to generate reports. Severe condition reports are sometimes referred to as alarms. Reports are used as the notification mechanism between the node/port and the network operator(s).

Report packets are converted to printable format using the Report Descriptor Table (RDT). The RDT defines the range of report values acceptable to the I/CTP and the text and parameter display format of each report. Report strings may be assembled by the I/CTP as a function of parametric values contained in the report itself.

If a report is received and ready to be displayed at the operator terminal, the terminal prompt character is changed to notify the operator that he should return to the top context level and request report printing. If multiple reports for the terminal are to be queued and the source node and port and the report code are the same, they are combined into a single report with a multiplicity indicator appended. The string displayed has the parameter value(s) of the first received report.

If the terminal keyboard has been idle for 10 seconds when the first terminal report is ready to be displayed, a message as displayed at the terminal (with a bell indication). Reports printed at the terminal are produced a page at a time. If more reports remain, the prompt character does not revert.

Entry Point - ICTP$REPORT:SPOOLER

Function

Print reports, if any, to terminal or printer.

Entry Conditions

I/CTP initialization completed.

Exit Conditions

Does not terminate, delays until reporting requested.

Entry Point - ICTP$REPORT:RPTQ (Batch Task)

Function

Queue received reports to report spooler queue.

Entry Conditions

Report-type addressed packet batched.

Exit Conditions

Prompt character set. Report queued (if unique) or multiplicity indicator incremented and packet freed.

6.6.4 Statistics

The I/CTP has the facility to request operational statistics form any network node or port and to display the returned value(s) at the I/CTP screen or printer. Statistics are maintained by all D814 ports and mainframes and may be requested once only or automatically at periodic intervals.

Statistics requests (in the form of IPOS addressed packets) are constructed and sent by the I/CTP with no response guaranteed, i.e., the I/CTP does not require a response from the port under inquiry. When the request is returned by the inquired port, it is sent to report control to be displayed. If, however, the port inquiried does not respond or the response (or original I/CTP request) are undeliverable due to network conjestion or failure of a single pathed I/NP, then no indication will occur.

The I/CTP receives statistics addressed packets from the various D814 network nodes and ports in response to a statistics request from the I/CTP. The I/CTP maintains self-statistics of processor loading, memory and buffer utilization. The statistics are timestamped upon receipt by the I/CTP. The I/CTP message function (dalagram interface) is also handled by this module.

Reports contain a one-byte field which is used to locate the print format of the statistics report in the Statistics Descriptor Table (SDT). The SDT is a table of format lists specifying the text, variable locations, lengths and display format.

Entry Point - ICTP$STATS:SPOOLER

Function:

Display statistics and messages.

Entry Conditions:

I/CTP initialization completed. Runs continuously printing when statistics or message packets are queued.

Exit Conditions:

None. Does not terminate.

6.6.5 System Services

The I/CTP has system service utilities which may be run on an active D814 network, although extreme caution should be exercised to prevent unpredictable perturbations in the active network. The utilities allow handling of the I/FDP files (file and directory listings, disk modification and editing), node and port memory examination and modification, port diagnostic loading, and single node or port rebooting. The I/CTP system services also has the capacity to build and send operator defined IPOS addressed packets.

Memory display/examination is supported in three modes:

1. Block memory display

   In this mode up to 64 consecutive bytes of node or port RAM, ROM or PROM may be displayed in hexadecimal. If the zone overlaps a node lock byte area, the integrity of that area will be maintained. If the zone overlaps an I/O area or the master controller area, unpredictable results may occur if the node is active.

2. Byte memory display/modification

   In this mode a single byte of node or port RAM, ROM, or PROM may be examined and RAM modified. If modified, an acknowledgement of the change is made. Non-acknowledgement mode may be selected to allow manipulation of I/O areas. In acknowledge mode changes to ROM, PROM, or I/O areas willnot verify.

3. Write byte/block memory

   In this mode from one to four bytes of port or node RAM may be written. Verification is optional. Writing to ROM, PROM, or I/O areas will not verify.

Port diagnostics loading or an active D814 node is available from any D814 software floppy disk (with the default disk being the current BOOT software floppy disk). Any floppy file on a software disk may be loaded to any port whose address is even (except the loading I/CTP port). Node diagnostics loading is not supported via the I/CTP.

Single node and individual port rebooting is also supported by the utilities. A request to the specified node is created, sent, and acknowledged to the I/CTP terminal.

The I/FDP floppy disk manipulator utilities are provided as part of the system services. A floppy disk may be:

1. Listed

   Listings of disk filenames and current status are requestable to the terminal or optionally to the printer. The floppy can optionally be locked into a static mode or allowed to remain active while directory listings are collected. General floppy status information (number of files, number of spare records, etc.) may also be listed.

   Specific lists (item by item) of free sectors and error records may be listed.

2. Dump/Modify

   From one to 256 bytes of floppy disk sector data may be displayed at the I/CTP and modified with verification acknowledgement. Data may be requested by file (where applicable) or by track and sector.

3. Verification/Recovery

   The Error Records Pool (ERP) may be tested. Each entry is tested multiply to verify the error condition which originally caused the entry to be placed in the ERP. Records which no longer cause errors may be listed or returned to the free sectors pool.

   Physical verification may also be performed in one of two modes:

   1. Full-disk verification

      Every sector of every track is read, written, read and compared to establish physical integrity. Records not comparing or causing I/O errors are noted in a list.

2. Full-disk testing

   Every sector of every track is read multiply. Records causing I/O errors are listed.

   A garbage collect utility is also provided to analyze disk integrity and report on anomalies if encountered.

4. Log file manipulations

   D814 log files may be printed to the terminal or printer. No arithmetic transformations are performed. Fields, other than those standard to all log file records (see Section 6.7) are printed as single byte hexadecimal fields.

The system service utilities also provide a byte file creation function. An entire IPOS byte file addressed packet may be assembled (excluding the length byte which is maintianed by the utility) and queued for transmission. This allows single operation testing of addressed packet-driven functions.

6.6.6 Protocol

The I/CTP interfaces with the I/FDP using a simplified protocol. Commands sent to the floppy contain ARQ sequence codes. Commands returned from the I/FDP (responses) contain an ARQ ACK/NAK indicator, the ARQ sequence code, and a command return code. I/O error counts are also indicated in responses.

The I/CTP also has a protocol associated with the datagram message facility. Normally an acknowledgement is returned by the receiving I/CTP or I/DGP. If the message is returned in error, the reason is displayed as a NAK on the message. If no ACK/NAK is received, no indication of this fact is printed.

6.6.7 Device Control

The I/CTP may operate two Signetics 2651 communications chips. One is used for the terminal and the other, optionally, for the printer. The terminal and printer are initialized as defined by configuration memory.

Data bytes are transferred one by one through the 2651. Input characters, except nulls, are stored in a circular buffer after any required editing. Null characters are ignored. If the buffer is full, the input character is ignored. Overrun characters are replaced in the buffer by a default garble character. Terminal breaks detected or loss of CTS cause the I/CTP to reboot itself (only). The same effect is achieved by entering the I/CTP reset command.

Output characters are taken from a circular buffer and transmitted a byte at a time. If a flyback character is detected in the buffer, 3 nulls are sent immediately following the character. When the buffer is empty, the transmitter interrupt is disabled and request-to-send set low. The interrupt and RTS are reset when the first character is placed in the circular buffer.

Entry Point - ICTP$CTIO:S2651_INT

Function:

2651 terminal interrupt handler.

Entry Conditions:

2651 interrupt outstanding; buffer pointer set.

Exit Conditions:

2651 interrupt serviced; buffer pointer advanced (if required).

Special Note:

The CTIO submodule has multiple entry points not listed here. Each of the entry points performs a single simple function such as screen clearance, cursor control, single of multiple line feeds and a carriage return.

6.6.8 Report Formats

A list of D814 report/alarm text is provided in Appendix I of the D814 Product Functional Specification. This section provides the format of the address packet expected by the I/CTP to produce that text. Parameters received by the I/CTP may be displayed as hexadecimal, binary, decimal, or ASCII values, under control of the RDT specification of the particular report.

The report packets are listed in the same order as the Product Functional Specification. Each report is prefixed by the Standard IPOS Addressed Packet header (length byte; destination node, port and module bytes; source node, port and module bytes) and a one-byte field which defines the type of port (mainframe or node, I/ATP, I/STP, etc.). This code is used to generate a report prefix when the report is displayed at the terminal or printer.

The type byte is followed by a one-byte selector which defines the report to which the parameters apply. This field is the report number. Following the report number are any parameters required for the report. The reports are listed below. All fields, unless otherwise designated, are one-byte long.

REPORT 0 - System Diagnostic (any node or port)

1. N-bytes of parameters (displayed in hexadecimal byte by byte)

REPORT 1 - Boot Complete (any node)

1. Running configuration (displayed in decimal)

2. Initiating node number (displayed in hexadecimal)

3. Boot code

   This field is used to define what type of Boot complete report is to be displayed. The meaning of the remaining parameters is a function of this code. These parameters are displayed in hexadecimal after the message text.

   a) Power Up

      No further parameters.

   b) Automatic Boot

      3 one-byte fields of zeroes. Initiating port number (hexadecimal).

   c) Operator

      Running software revision number (hexadecimal).
      Running software release number (hexadecimal).
      Software source node (hexadecimal)
      Software source port (hexadecimal)

REPORT 2 - Processor Utilization Exception (any node or port)

1. Current processor loading percentage value (displayed in decimal)

REPORT 3 - Buffer Utilization Exception (any node or port)

1. Current buffer utilization percentage (decimal)

REPORT 4 - Link Up (I/NP)

1. Initiating I/NP node (hexadecimal)
2. Initiating I/NP port (hexadecimal)
3. Adjacent I/NP node (hexadecimal)
4. Adjacent I/NP port (hexadecimal)
5. 2-byte speed (hexadecimal)

REPORT 5 - Link Down (I/NP)

1. Initiating I/NP node (hexadecimal)
2. Initiating I/NP port (hexadecimal)
3. Adjacent I/NP node (hexadecimal)
4. Adjacent I/NP port (hexadecimal)
5. 2-byte speed (hexadecimal)
6. Cause code

    a) Remote I/NP request
    b) Node request
    c) No ACK timeout
    d) XMT clock failure
    e) RCV clock failure
    f) CTS dropped
    g) DCD dropped
    h) System failure

REPORT 6 - Unexpected Link Initialization (I/NP)

1. No parameters

REPORT 7 - Overrun (I/NP)

1. No parameters

REPORT 8 - Unexpected Link Activation (I/NP)

1. No parameters

REPORT 9 - Data Threshold Exceeded (any port)

1. 2-byte current user data rate per second (displayed in decimal)

REPORT 10 - Error Density Exceeded (any port)

1. 2-byte count of errors per second (displayed in decimal)

REPORT 11 - Master Directory Invalid (I/FDP)

1. Drive number
2. 8-byte volume ID in ASCII
3. 16-byte file name in ASCII
4. Track number (displayed in decimal)
5. Sector number (decimal)

REPORT 12 - Subdirectory Invalid (I/FDP)

1. Drive number
2. 8-byte volume ID in ASCII
3. 16-byte file name in ASCII
4. Track number of subdirectory record (decimal)
5. Sector number of subdirectory record (decimal)
6. Cause code

   a) Bad forward pointer
   b) Bad backward pointer
   c) Invalid data record
   d) Invalid sequence number
   e) Invalid filename

   NOTE: Report packets with cause codes a - c are followed by two one-byte fields which are the pointer (or data record track and sector) and are displayed in hexadecimal. Packets with cause code d are followed by a one-byte sequence number which is displayed in hexadecimal.

REPORT 13 - Free Records Pool (Type 1; I/FDP)

1. Drive number
2. 8-byte volume ID in ASCII
3. Track under dispute (decimal)
4. Sector under dispute (decimal)
5. 16-byte ASCII filename

REPORT 14 - Free Records Pool (Type 2; I/FDP)

1. Drive number
2. 8-byte ASCII volume ID
3. Track of unallocated record (decimal)
4. Section of unallocated record (decimal)

REPORT 15 - WRITE ERROR (I/FDP)

1. Drive number
2. 8-byte ASCII volume ID
3. 16-byte ASCII filename (or keyword, i.e., MD, ERP, FSM, FLR2, etc.)
4. Track of write failure
5. Sector of write failure
6. Cause code

   a) Not ready
   b) Write protect
   c) Record not found
   d) Lost data
   e) CRC error - ID
   f) CRC error - data
   g) Write fault (should never occur)

REPORT 16 - Read Error (I/FDP)

1. Drive number
2. 8-byte ASCII volume ID
3. 16-byte ASCII filename (or keyword, i.e., MD, ERP, FSM, VLR2, etc.)
4. Track of read failure
5. Sector of read failure
6. Cause code - see report 15

REPORT 17 - Insufficient Memory (any port)

1. 2-byte memory size (decimal)

REPORT 18 - I/FDP Initialized (long form)

1. Drive number
2. 8-byte ASCII volume ID
3. Density indication code:

   a) Single density
   b) Dual density

4. Sides indication code:

   a) Single sided
   b) Dual sided

5. 2-byte count of Master Directory Sectors (decimal)

6. MD validity code:

   a) Null (blanks)
   b) Valid
   c) Invalid

7. 2-byte count of subdirectory sectors (decimal)
8. Subdirectory validity code - see MD validity code
9. 2-byte count of free sectors (decimal)
10. Free sectors validity code - see MD validity codes
11. 2-byte count of error records entries (decimal)
12. ERP validity code - see MD validity code
13. 2-byte count of physically and logically consistent files
14. Write protect code:

   a) Yes
   b) No

REPORT 19 - Directory Error (I/FDP)

1. Drive number
2. 8-byte ASCII volume ID
3. Directory type code:

   a) Null (no character)
   b) Sub-

4. 16-byte ASCII filename
5. Error track (decimal)
6. Error sector (decimal)
7. Accessing node (hexadecimal)
8. Accessing port (hexadecimal)
9. Cause code:

   a) Recoverable I/O error on READ
   b) Unrecoverable I/O error on READ

REPORT 20 - Disk Full (I/FDP)

1. Drive number
2. 8-byte ASCII volume ID
3. 16-byte ASCII filename (or keyword, i.e., FSM, ERP, MD, etc.)

REPORT 21 - Invalid Log Port

1. Node of specified I/CTP (hexadecimal)
2. Port specified as I/CTP (hexadecimal)

REPORT 22 - I/FDP Initialization (short form)

1. Drive number
2. 8-byte ASCII volume ID

1. Mainframe Statistics

1.	Processor loading percentage value	(decimal display)
2.	Buffer utilization percentage value	(decimal display)
3.	2-byte apparent node throughput	(decimal display)
4.	2-byte statistical node throughput	(decimal display)
5.	2-byte capacity in KCP's	(decimal display)

2. I/NP Statistics

1.	Processor loading percentage value	(decimal display)
2.	Buffer utilization percentage value	(decimal display)
3.	Retransmitted frames count	(decimal display)
4.	Normal frame transmission count	(decimal display)
5.	2-byte user data bytes count	(decimal display)
6.	Average delay in MS	(decimal display)
7.	NAK count	(decimal display)
8.	Average ARQ	(decimal display)
9.	Maximum buffers count (2 bytes)	(decimal display)

3. I/ATP Statistics

1.	Processor loading percentage	(decimal)
2.	Buffer utilization percentage	(decimal)
3.	2-byte buffer maximum count	(decimal)
4.	Compression efficiency percentage	(decimal)
5.	Error rate value	(decimal)

4. I/CTP Statistics

1.	Processor loading percentage	(decimal)
2.	Buffer utilization percentage	(decimal)
3.	2-byte maximum buffers count	(decimal)

5. <u>I/FDP Statistics</u>

1.	Drive number	(decimal)
2.	8 byte volume ID	
3.	Processor loading percentage	(decimal)
4.	Buffer utilization percentage	(decimal)
5.	2-byte maximum buffers count	(decimal)
6.	Average number of READ I/O commands per second	(decimal)
7.	Average number of WRITE I/O commands per second	(decimal)
8.	Number of accessors	
9.	Number of accessor commands processed	
10.	Volume ID and drive number	
11.	Write protect status	
12.	Average READ I/O error rate per minute	
13.	Average WRITE I/O error rate per minute	
14.	Number of files opened for READ	
15.	Number of files opened for WRITE	

6. <u>I/SSTP-BSC Statistics</u>

To be defined.

7. <u>I/SSTP-HASP Statistics</u>

To be defined.

8. <u>I/MXP Statistics</u>

To be defined.

9. <u>I/MATP Statistics</u>

To be defined.

10. <u>I/STP Statistics</u>

To be defined.

6.7 Intelligent Floppy Disk Port (I/FDP)

The I/FDP is organized as a set of functionally independent submodules which allow guaranteed storage and retrieval of data in a D814 network. The time and memory required to support these functions is of secondary importance to the assurance of the total accuracy of the data transfer and file integrity.

The I/FDP modules perform the following functions:

1. Initialization
2. Protocol Management
3. File Management
4. Device/Line Control

Function 4 is really two functions. Device control refers to an I/FDP which is maintaining its files on a real floppy disk attached to a port. Line control refers to a floppy which is maintaining its files on a floppy emulator referenced over a communications line. For any I/FDP, only one of the two functions will exist.

6.7.1 I/FDP Data Structures

6.7.1.1 Commands and Responses

The I/FDP uses commands to transfer information between the floppy disk and a requestor. A requestor may be any node or port in a D814 network. Commands contain protocol fields (sequence numbers, ARQ, return codes) and information required by the file management system. A command, having been processed by an I/FDP is returned to the requestor. A returned command is referred to as a response. Commands are of the following format:

L	DEST	SOURCE	F#	I/O CTR	CC	S#	RC	Pn

where:

L is a one byte length specification (equivalent to the IPOS addressed packet length field, physically).

DEST is the 3 byte addressed packet destination field. It defines the node, port and module to which the command is issued.

SOURCE is the 3 byte addressed packet source field, specifying the command initiator node, port, and module.

F# is a one byte field specifying the file reference number and drive selected.

I/O CTR is a one byte field returned to the requestor containing the count if I/O errors encountered in attempting to execute the command. The count is the number of errors to read or write I/O commands required to execute the requestor command. If more than one read or write was required to satisfy the requestor command it is the maximum error count of all I/O commands. The count does not indicate I/O errors to reads or writes of directory type records.

CC is a one byte command code. The I/FDP command codes are:

1. OPEN - create a logical connection between a requestor and a floppy disk file.

2. CREATE - create a logical connection between a requestor to allow the creation of a floppy disk file.

3. READ - read a record from a floppy file.

4. WRITE - write a record to a floppy disk file.

5. RESET - set the next read or next write location pointer for a file to a specified value.

6. CLOSE - sever the logical connection between a floppy file and a requestor.

7. DELETE - erase a floppy disk file.

8. UPDATE - read and write a file. Write operation is performed on last read record and for the same count of characters as last read operation.

9. LOCK (drive) - prevent accessing of a drive in write, and/or read mode by all requestors other than the command initiator, or allow certain types of normally prohibited accessing simultaneous to normal accessing modes.

10. UNLOCK (drive) - reverse a LOCK command.

11. STATUS FLOPPY - return the status information about a particular floppy.

12. STATUS FILE - return the status information about a particular floppy disk file.

13. LOG - add a record to the system log file.

14. ADD - add a record to a normal floppy disk file.

15. TEST TRACK - test a specified track on the floppy, and turn a readability indicator for each sector of that track. This testing is non-destructive and does not cause a report to be issued by the I/RFDP when an error occurs reading a sector for a track.

16. NOP - hold the connection between the requestor and the floppy file open, do not timeout and automatically sever the link.

RC is a one byte field returned to the requestor indicating the final status of the command operation.

Pn is a variable length command code dependent parameter string.

Responses are created by setting the I/O counter and RC fields and replacing the parameter field with data or status information, as required.

6.7.1.2 The FILE-ID Table (FID)

This table associates requestors to I/FDP File Control Blocks (FCBs). The table is of the form:

FCB 1 ptr	FCB 2 ptr	...	FCB N ptr

Association is a function of the F# field of the command.

6.7.1.3 File Control Blocks (FCB)

An FCB is used to maintain pointers, status, and counters for an active requestor/floppy file association. A requestor is active once it has sent an OPEN, CREATE, ADD, LOG, or LOCK command, and that command has been accepted and processed by the I/FDP. Processing of these commands causes an FCB to be assigned for a requestor. A requestor is said to be nonactive when it CLOSEs a file (implicitly or explicitly), or it UNLOCKS a drive (implicitly or explicitly). Explicit CLOSEs result from CLOSE commands from the requestor, implicit CLOSEs result from command timeout conditions.

The FCB has the following format:

RN	RP	RM	NS	HS	EC	ST	LC	CMD PTR	NR	NW

FRT POINTER	FS	CS	OM	MDP	TIMER FIELDS

where:

RN is the one byte requestor node from the OPEN, CREATE, ADD, LOG or LOCK command which established the connection.

RP is the one byte requestor port from the OPEN, CREATE, ADD, LOG, or LOCK command which established the connection.

RM is the one byte requestor module from the requestor OPEN, CREATE, LOCK, LOG, or ADD command which established the connection.

NS is a one byte field containing the next expected sequence number.

HS is a one byte field containing the highest sequence number allowed. HS-NS defines the command sequence number window.

EC is a one byte error counter field.

ST is a one byte status field.

LC is a one byte field containing the last successfully complete command from the requestor.

CMD PTR is a two byte pointer to the command currently being executed by the I/FDP.

NR is a two byte pointer to the next-read location for the floppy disk file.

NW is the two byte pointer to the next-write location for the floppy disk file.

FRT POINTER is a two byte pointer to the FRT (see 6.7.1.4).

FS is a two byte pointer to the first subdirectory record for the file (memory pointer).

CS is a two byte memory pointer to the current subdirectory being used for the floppy file (memory pointer).

OM is a two byte field with the first byte specifying the open mode, and the second byte containing the security mode. The I/FDP supports three types of security:

1. Low security - the current subdirectory record is written to disk only when all data record pointers in it are filled.

2. High security - the current subdirectory is written to the floppy disk whenever a data record is added to the file.

3. Maximum security - same as high security, plus every data record written is read back and verified before declaring the operation successfully completed.

MDP is a 4 byte pointer into the Master Directory. This entry is really two 2 byte Master Directory pointers. The first 2 bytes specify the sequence number and record index of the entry for the file, and the second 2 bytes specify the track and sector of the same record on the floppy disk.

Timer Fields - This field is used to maintain I/FDP command timers. Timeout conditions cause an implicit CLOSE (or UNLOCK) to be performed.

6.7.1.4 Former Requests Table (FRT)

The FRT provides the I/FDP with a limited capacity to satisfy duplicate (or overlapping) requests for data from the same file without requiring actual disk I/O. The table is composed of daisy chained entries of the form:

K	#R	#E	DATA pointer	Next FRT pointer

where:

K is a one byte key used to identify the file to which this FRT refers.

#R is a one byte count of requestors currently accessing the file in read-type mode.

#E is a one byte count of the number of entries currently in this FRT entry's data chain.

DATA pointer is a two byte pointer to the first entry of an FRT data chain.

Next FRT pointer is a two byte pointer to the next FRT entry or zero, if the current FRET entry is the last.

The data chain is of the form:

D-ID	LEN	DATA ptr	Next data block pointer

where:

D-ID is a two byte field containing the track and sector from which the data was taken.

LEN is a one byte data length indicator.

DATA ptr is a two byte pointer to the actual data.

Next data block pointer is a two byte pointer to the next data block entry, or zero if the current data block is the last.

6.7.1.5 The Command Holding Queue (CHQ)

The CHQ is a queue pair (two associated queues) used to hold ADD and LOG commands received by an I/FDP when the drive specified by the command is LOCKED, or all available FCBs are in use. Commands are protocol prevalidated prior to enqueueing to the CHQ. When a drive UNLOCKS (or an FCB becomes available), the commands are dequeued and processed.

Note that if an ADD or LOG command is received, and the appropriate CHQ is nonempty, that command, after validation, will also be queued to the CHQ (after timestamping) to maintain data order. All LOG file commands are time-stamped by the I/FDP.

6.7.1.6 The Virtual Master Directory Table (VMDT)

The VMDT is used to minimize I/O to the Master Directory. It is composed of four entries (per drive) of the form:

MD S#	RES	MD data pointer

where:

MD S# is a one byte sequence number.

RES is one reserved byte.

MD data pointer is a two byte memory pointer to the MD record.

6.7.2 I/FDP File Structures

6.7.2.1 The Volume Label Records (VLRs)

Every D814 floppy disk has two VLRs. The first (VLR-1) is at a fixed location on every floppy disk, and contains:

1. 8 byte ASCII label (blank filled)
2. One byte disk type identifier (logger, program etc.)
3. One byte density/sides indicator
4. Two byte pointer to the second VLR record (VLR-2).
5. 64 bytes of optional disk identification text.

The VLR-2 record contains:

1. One byte record identifier.
2. 8 byte ASCII label (same as VLR-1).
3. Two byte pointer to the first Master Directory record.
4. Two byte pointer to the first Error Records Pool (ERP) record.
5. Two byte pointer to the last ERP record.
6. Two byte count of the number of subdirectory type records currently in use.
7. Two byte count of the number of data records currently in use.
8. Two byte count of the number of free (unallocated) sectors currently available.
9. Two byte count of the number of ERP entries.
10. Two byte number of the next available sector on the RST track for emergency CLOSE procedures. The RST track (Reserved Sector Track) is the last track of a D814 floppy. This track is reserved for use by the I/FDP when a disk full condition would prevent the final CLOSE command processing for a file.
11. A one byte indicator of the validity of the count fields. When this field is zero, the count fields are possibly invalid (file OPENed and currently under processing).

6.7.2.2 The Master Directory

Every D814 floppy disk contains at least one MDR (Master Directory Record). MDRs contain the filename, availability indicator, and first sub-directory pointer for D814 disk files. The availability indicator is used to mark physically or logically impaired disk files, and deleted files. A file is logically impaired when an invalid field is detected by the I/FDP software. A file is physically impaired when one or more records of that file are unreadable.

Each MDR also contains a record ID (1 byte), a sequence number (1 byte), and a forward pointer (2 bytes). The pointers contain track and sector values.

6.7.2.3 Subdirectory Records

Subdirectory records or File Directory Records (FDRs) contain pointers to the data records of a disk file, and forward and back pointers to the other FDRs of that file. In addition, FDRs contain a record ID (1 byte), and a sequence field (1 byte).

The first FDR also contains a 4 byte file length, a 3 byte indicator of the last requestor to access the file in read-type mode (node, port, and module), a 3 byte indicator of the last requestor to access the file in write-type mode (node, port, and module), a 3 byte indicator of the requestor which created the file (node, port, and module), a count of the FDRs which compose the file (1 byte), a count of data records (2 bytes), and a 2 byte pointer to the last FDR for the file (track, sector).

6.7.2.4 The Error Records Pool (ERP)

The ERP is a directory containing a list of those sectors on the D814 floppy which have caused unrecoverable I/O errors (refer 6.7.5). Each entry is 3 bytes long, containing the track and sector pointer and a one byte indication of whether the I/O failure resulted from a read operation or a write operation. If the record was added to the ERP as the result of a read operation the record type is also contained in the indicator byte (e.g. MDR, FDR, etc.).

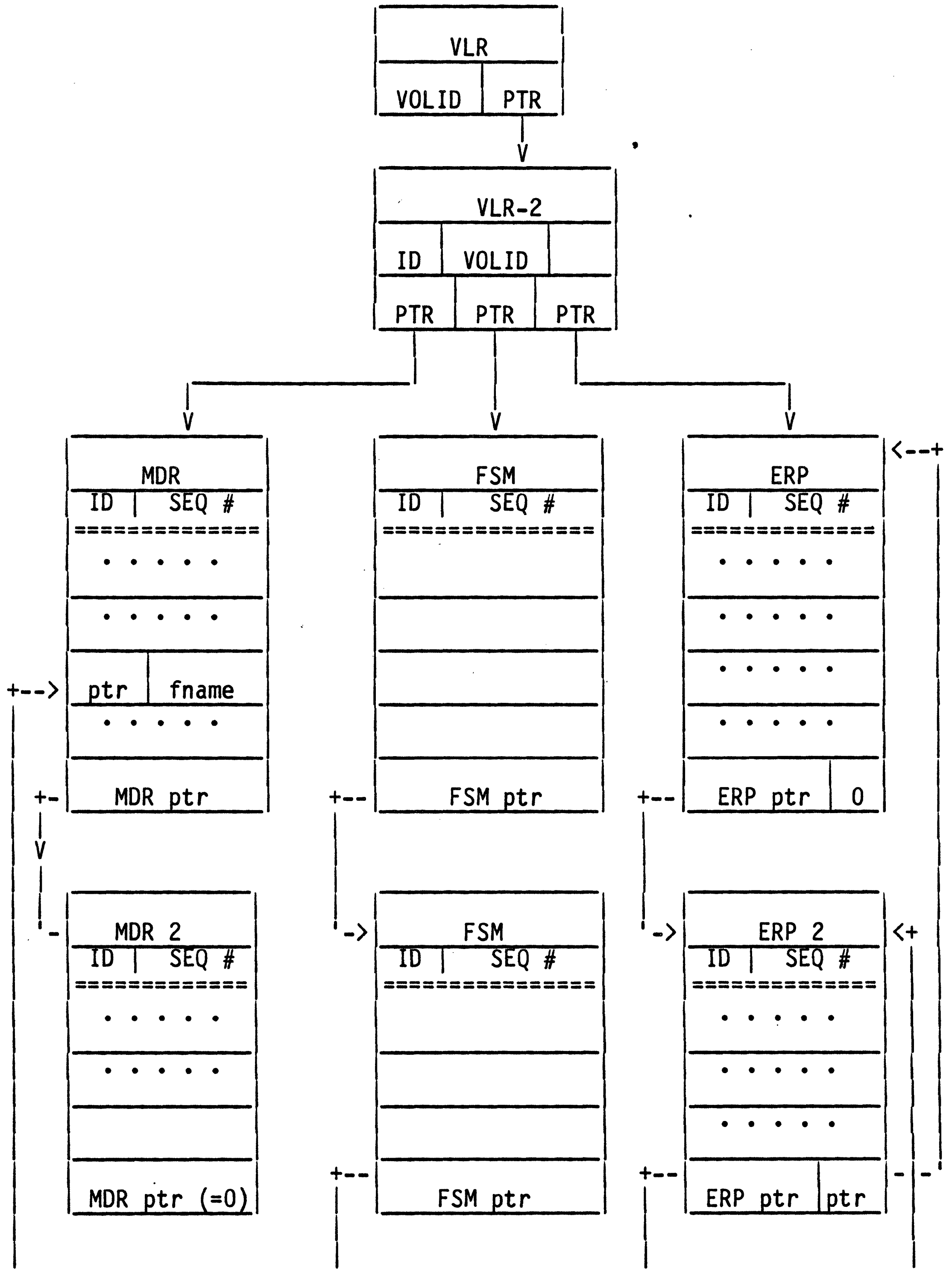
Each ERP directory record also contains a record identifier (1 byte), a one byte sequence number, a two byte forward pointer, and a two byte back pointer.

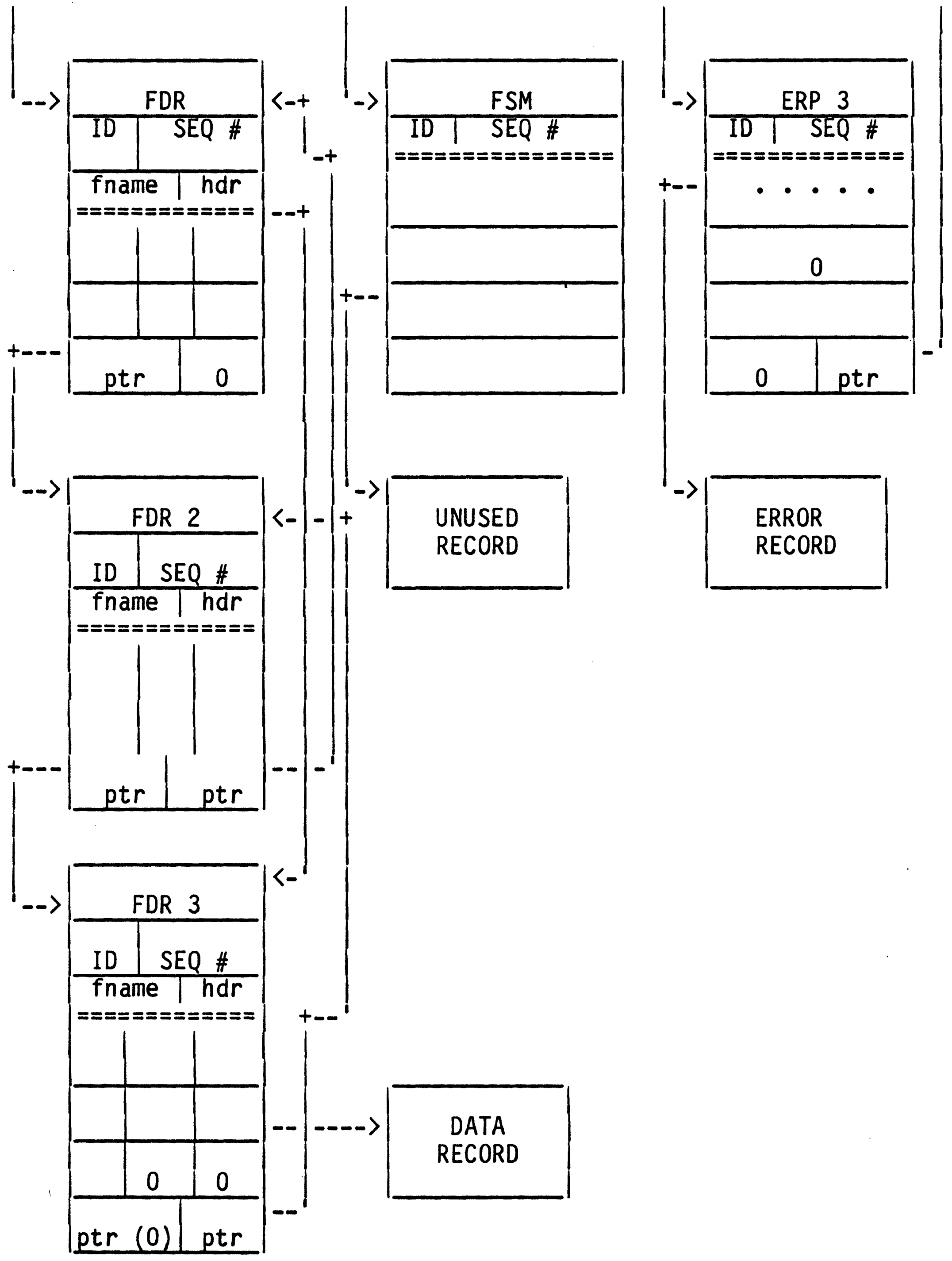
6.7.2.5 The Free Sectors Map (FSM)

The FSM contains a bit map corresponding to the sectors of a D814 floppy disk. Allocation and deallocation of sectors on the floppy by the file management system is keyed from this map. Each FSM contains a one byte record ID, a one byte sequence number, and a two byte forward pointer, in addition to the map bits.

The map bits are grouped to correspond to tracks on the disk, i.e. 26 bits for a single density drive (using 4 bytes with the low order 6 bits unused), and 52 bits for a dual density drive (using 7 bytes with the low order 4 bits unused). A 1 bit in the FSM indicates that the corresponding sector on the floppy is available.

6.7.2.6 I/FDP Logical Floppy Layout





Note: For ease of graphical representation, some of the record fields have been positionally shifted on some records.

6.7.2.7 I/FDP Command Packet Layouts

I/FDP OPEN Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL -	byte 0:	Packet length (EQ$IFDP:CMD_OPEN_XLEN)
Dn -	byte 1:	Destination node
Dp -	byte 2:	Destination port
Dm -	byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn -	byte 4:	Source node
Sp -	byte 5:	Source port
Sm -	byte 6:	Source module
FREF -	byte 7:	File Reference number (Drive number). Filled in by I/FDP whenever FCB allocated, Must be copied to all subsequent commands for that file.
CC -	byte 8:	Command code (EQ$IFDP:CMD_OPEN) OPEN an existing file.
RC -	byte 9:	Return code
SEQ# -	byte 10:	Sequence number (modulo 8)
IOERR# -	byte 11:	Count of I/O errors during command execution

PARMS:

OPEN MODE -

byte 12: Options selected from combinations:

OPEN for write (ERQ$IFDP:ACMODE_OPENWR)
Lock drive against READY/WRITE (EQ$IFDP:ACMODE_LOCKRD). CP-type.
Lock drive against WRITE (EQ$IFDP:ACMODE_LOCKWR). CP-type.
Lock drive allowing READS and WRITES (EQIFDP:ACMODE_LOCKO). CP-type.
Access file in DIRECT mode (EQ$IFDP:ACMODE_DIRECT).
Access file in UPDATE mode (EQ$IFDP:ACMODE_UPDATE).
Access file in MOD mode (EQ$IFDP:ACMODE_MOD).
Access file in CP mode (EQ$IFDP:ACMODE_CP).

Note: 1) If no bits are selected, the access mode would be READ, sequential.
2) Open mode DIRECT WIRTE is not supported; this combination is equivalent to UPDATE DIRECT or UPDATE DIRECT MOD.
3) CP type commands lock the entire drive, not just a file.

SECURITY MODE -

byte 13:

LOW Security - Update FDR (subdirectory) record to disk only when an entire FDR record is filled. (EQ$IFDP:FCB_OM.SECLOW).

HIGH Security - Update FDR to disk after each write. (EQ$IFDP:VCM_OM.SECHIGH).

MAXIMUM Security - High security plus verify each write. IF written records does not verify, write a new record. Repeat until record verifies.

FILENAME - bytes 14-29.

ASCII filename, left adjusted, blank (X'20') filled. Alphanumeric unique (by drive) file name.

RESPONSES:

Operation completed (RC=EQ$IFDP:RC_DONE). Following fields appended.

byte 30 - Sides/density byte. Bits possibly set are: Double sided (EQ$IFDP:VLRI_SIDEN_DS); double density (EQ$IFDP:VLRI_SIDEN.SD); bit not set indicates single.

bytes 31-33 - File length (in bytes).

Operation incomplete (RC=EQ$IFDP:RC_UNAVAIL). File already accessed in a mode which prohibits access as specified.

byte 30 - Same as for completed operation.

bytes 31-33 - Current accessing node, port, and module.

Operation incomplete (RC=EQ$IFDRP:RC_MAXREQ). Maximum number of requestors for specified drive has been reached. No commands requiring a new FCB will be excepted until requestors currently accessing file relinquish their FCBs.

Operation incomplete (RC=EQ$IFDP:RC_LOCKED). Drive specified has been locked previously against mode specified (READ or WRITE).

Operation incomplete (RC=EQ$IFDP:RC_LOGBAD). File specified has been found to be logically inconsistent. Accessing prohibited unless in CP mode.

Operation incomplete (RC=EQ$IFDP:RC_PHYBAD). File specified has been found to be physically impaired. Accessing prohibited unless in CP mode.

Operation incomplete (RC=EQ$IFDP:RC_SFTPROT). Disk has been software write protected.

Operation incomplete (RC=EQ$IFDP:RC_SECMODE). Security mode specified as invalid.

Operation incomplete (RC=EQ$IFDP:RC_INV_FNAME). Specified filename invalid.

Operation incomplete (RC=EQ$IFDP:RC_DISKFULL). Disk is full. May occur in either read or write access commands.

Operation incomplete (RC=EQ$IFDP:RC_NOTRDY). Drive not ready.

Operation incomplete (RC=EQ$IFDP:RC_IOERR). I/O error occurred and did not vanish after repeated retries (see I/O err counter for number of retries).

Operation incomplete (RC=EQ$IFDP:RC_NOFILE). Filename specified not found.

Operation incomplete (RC=EQ$IFDP:RC_INV_MODE). Access mode bit configuration illegal (e.g., READ,MOD).

Operation incomplete (RC=EQ$IFDP:RC_INV_DRV). Drive specified invalid.

Operation incomplete (RC=EQ$IFDP:RC_INV_MD). File not found, invalid (logically or physically) Master Directory chain.

Operation incomplete (RC=EQ$IFDP:RC_FORCED_CLOSE). Connection to drive or file severed. An I/CTP has FORCED the requestor or the specified drive has been physicall opened. This response may occur at any time and contains the last ACKed sequence number.

Operation incomplete (RC=EQ$IFDP:RC_WRPROT). Drive has physical write protect. This response may occur to either read or write accessing modes.

Operation incomplete (RC=EQ$:FDP:RC_OFFLINE). Drive offline.

I/FDP CREATE Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:CMD_CREATE_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number (Drive number)
CC - byte 8:	Command code (EQ$IFDP:CMD_CREATE) Create a new file.
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

byte 12:	Create mode (must be OPENNR, see open command).
byte 13:	Security code (see OPEN command).
bytes 14-29:	Filename (see OPEN command).
bytes 30-52:	Optional text. ASCII, blank filled.

RESPONSES:

No bytes appended.

Return codes as defined for OPEN command, plus:
Operation incomplete (RC=EQ$IFDP:RC_EXISTS). File already exists on specified drive.

I/FDP LOCK Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL	- byte 0:	Packet length (EQ$IFDP:CMD_LOCK_XLEN)
Dn	- byte 1:	Destination node
Dp	- byte 2:	Destination port
Dm	- byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn	- byte 4:	Source node
Sp	- byte 5:	Source port
Sm	- byte 6:	Source module
FREF	- byte 7:	File Reference number (Drive number)
CC	- byte 8:	Command code (EQ$IFDP:CMD_LOCK)
RC	- byte 9:	Return code
SEQ#	- byte 10:	Sequence number (modulo 8)
IOERR#	- byte 11:	Count of I/O errors during command execution

PARMS:

byte 12: Lock mode:

Lock drive against all READ and WRITE requestors (prevent new accessors). EQ$IFDP:ACMODE_LOCKRD.

Lock drive against new WRITE type requestors (EQ$IFDP:ACMODE_LOCKWR).

Lock drive allowing simultaneous accessing with normal requestor (EQ$IFDP:ACMODE_LOCKO).

RESPONSES:

Command complete (RC=EQ$IFDP:RC_DONE). Drive LOCKed into specified mode. CP-type commands may be specified.

Command incomplete (RC=EQ$IFDP:RC_WAITLOCK). Drive locked but currently has active user(s) in the prohibited state. Periodic checks for those users disconnecting will be made. When all such users are disconnected, the command completed response code is given; otherwise this response code is given. No timeouts will occur while waiting. Note: The response packets will all have the same sequence numbers.

Command incomplete (RC=EQ$IFDP:RC_INV_PORT). Requestor is not an I/CTP.

Command incomplete (RC=EQ$IFDP:RC_LOCKED). Drive already locked; bytes 13-15 contain locking node, port, and module.

OTHER RESPONSES:

See OPEN command.

I/FDP READ Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:CMD_READ_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number (From OPEN response)
CC - byte 8:	Command code (EQ$IFDP:CMD_READ) READ bytes from an OPENed file.
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

bytes 12-14:	Byte location. Offset into file where the first byte is location 0.
byte 15:	Read length; Number of bytes to read. (Less than Ø29 and multiple 16)
byte 16:	Unused. May be used for information requestor wishes to have returned with response for identification purposes.

RESPONSES:

Operation complete (RC=EQ$IFDP:RC_DONE).

byte 17-n Data read.

Operation incomplete (RC=EQ$IFDP:RC_FORCED_CLOSE). Connection to file has been severed by an I/CTP or other CP-type requestor. File must be re-OPENed.

Operation incomplete (RC=EQ$IFDP:RC_POST_CMDTO). Command completed but I/FDP has timed out on commands. All commands received but not yet processed will have this return code if successfully completed. New commands not accepted; file must be reOPENed.

Operation incomplete (RC=EQ$IFDP:RC_INV_ACC). Command invalid in OPEN specified mode.

Operation incomplete (RC=EQ$IFDP:RC_TOO_HIGH). Maximum number of commands (EQ$IFDP:CMD_WINDOW) allowed to be outstanding has been exceeded. Command rejected until commands received are processed.

Operation incomplete (RC=EQ$IFDP:RC_NDOWN). File pointed to by file reference number field was not OPENed by this requestor. FREF field may not have been copied from OPEN or CREATE command.

Operation incomplete (RC=EQ$IFDP:RC_INV_LOC). Read location not sequential to last read location (Mode ≠ DIRECT). Bytes 17-19 contain expected read location.

Operation incomplete (RC=EQ$IFDP:RCGTEOF). Read location extends across or exceeds end of file. If extends across, all data to end of file will be included.

OTHER CODES:

See OPEN command.

I/FDP READC Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:CMD_READC_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number
CC - byte 8:	Command code (EQ$FDP:CMD_READ)
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

bytes 12-14:	Read location
byte 15:	Length (multiple of 16, less than or equal to 128)

RESPONSES:

Command Complete

byte 16-N:	Data (length requested)
bytes N-n+8:	Filename

OTHER RESPONSES:

Same as READ command.

I/FDP WRITE Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL	- byte 0:	Packet length (EQ$IFDP:CMD_WRITE_XLEN)
Dn	- byte 1:	Destination node
Dp	- byte 2:	Destination port
Dm	- byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn	- byte 4:	Source node
Sp	- byte 5:	Source port
Sm	- byte 6:	Source module
FREF	- byte 7:	File Reference number (from CREATE/OPEN response)
CC	- byte 8:	Command code (EQ$IFDP:CMD_WRITE) Write a block of data to a file.
RC	- byte 9:	Return code
SEQ#	- byte 10:	Sequence number (modulo 8)
IOERR#	- byte 11:	Count of I/O errors during command execution

PARMS:

bytes 12-14:	WRITE location (file offset, relative to first byte=displacement 0).
byte 15:	Data length (multiple of 16, less than or equal to 128)
byte 16:	Unused. May be used by requestor for identification of response.
byte 17-n	Data to be written to disk.

RESPONSES:

Command completed (RC=EQ$IFDP:RC_DONE). Data bytes not returned.

Command incomplete (RC=EQ$IFDP:RC_INV_LEN). Length specified does not match data byte count (17-n).

Command incomplete (RC=EQ$IFDP:RC_INV_LOC). Mode ≠ DIRECT. Write location not sequential to last write. Next expected write location is stored at bytes 17+128 (145).

OTHER RESPONSES:

See OPEN and READ commands.

I/FDP RESET Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:CMD_RESET_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number (from OPEN command)
CC - byte 8:	Command code (EQ$IFDP:CMD+RESET)
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

bytes 12-14: Reset-to location.

RESPONSES:

Command complete (EQ$IFDP:RC_DONE). Next read and write locations set to specified value.

Command incomplete (EQ$IFDP:RC_GTEOF). RESET LOCATION exceeds file length.

OTHER RESPONESES:

See OPEN and WRITE commands.

I/FDP UPDATE Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL	- byte 0:	Packet length (EQ$IFDP:CMD_UPDATE_XLEN)
Dn	- byte 1:	Destination node
Dp	- byte 2:	Destination port
Dm	- byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn	- byte 4:	Source node
Sp	- byte 5:	Source port
Sm	- byte 6:	Source module
FREF	- byte 7:	File Reference number (from OPEN response)
CC	- byte 8:	Command code (EQ$IFDP:CMD_UPDATE)
RC	- byte 9:	Return code
SEQ#	- byte 10:	Sequence number (modulo 8)
IOERR#	- byte 11:	Count of I/O errors during command execution

PARMS:

bytes 12-14:	File byte reference (location)
byte 15:	Length (less than 129 and multile of 16)
byte 16:	Type (READ=EQ$IFDP:UPDATE_RD; WRITE=EQ$IFDP:UPDATE_WR)
byte 17-n	Data (write update)

RESPONSES: See WRITE, READ comands

CP-Update:

byte 12:	Track
byte 13:	Sector
byte 14:	Unused
byte 15-n	As for normal update

RESPONSES:

Command complete. (Data read or written)

Command incomplete (RC=EQ$IFDP:RC_INV_TRK). Invalid track value.

Command incomplete (RC=EQ$IFDP:RC_INV_SEC). Invalid sector value.

Command incomplete (RC=EQ$IFDP:RC_INV_UPDCMD). Invalid update command sequence (e.g. WRITE, WRITE)

OTHER RESPONSES:

See READ, WRITE commands.

I/FDP CLOSE Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL	- byte 0:	Packet length (EQ$IFDP:CMD_CLOSE_XLEN)
Dn	- byte 1:	Destination node
Dp	- byte 2:	Destination port
Dm	- byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn	- byte 4:	Source node
Sp	- byte 5:	Source port
Sm	- byte 6:	Source module
FREF	- byte 7:	File Reference number (from OPEN command)
CC	- byte 8:	Command code
RC	- byte 9:	Return code
SEQ#	- byte 10:	Sequence number (modulo 8)
IOERR#	- byte 11:	Count of I/O errors during command execution

PARMS:

RESPONSES:

Command complete (RC=EQ$IFDP:RC_DONE). Connection with file completed. Active records have been written to disk.

OTHER RESPONSES:

See OPEN command.

I/FDP UNLOCK Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL	- byte 0:	Packet length (EQ$IFDP:CMD_UNLOCK_XLEN)
Dn	- byte 1:	Destination node
Dp	- byte 2:	Destination port
Dm	- byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn	- byte 4:	Source node
Sp	- byte 5:	Source port
Sm	- byte 6:	Source module
FREF	- byte 7:	File Reference number (from LOCK response)
CC	- byte 8:	Command code (EQ$IFDP:CMD_UNLOCK)
RC	- byte 9:	Return code
SEQ#	- byte 10:	Sequence number (modulo 8)
IOERR#	- byte 11:	Count of I/O errors during command execution

PARMS:

byte 12: Unlock mode:

Unlock from LOCKRD to LOCKWR (permit read-type accessors) EQ$IFDP:UNLOCK_MODE2

Unlock from LOCKRD or LOCKWR to LOCKØ (permit read and write type accessors) EQ$IFDP:UNLOCK_MODE1

Unlock complete (through issuing CP-type commands) EQ$IFDP:UNLOCK_RELSE

RESPONSES:

Command complete (RC=EQ$IFDP:RC_DONE). Drive state changed as specified.

Command incomplete (RC=EQ$IFDP:RC_NOTLOCK). Drive not locked by requestor.

OTHER RESPONSES:

See OPEN command.

I/FDP QUERY ACCESSOR Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:CMD_QUERYACC_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number (Drive number IF standalone) or from OPEN/LOCK response.
CC - byte 8:	Command code (EQ$IFDP:CMD_QUERRYACC). List active requestor for drive.
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

byte 12: Drive number being querried.

RESPONSES:

Command complete (RC=EQ$IFDP:RC_DONE).

bytes 13-52: Node, port, and module of all requestors accessing specified drive. If node=∅, not active and port and module should be ignored.

OTHER RESPONSES:

See OPEN command.

NOTE: This command may be issued standalone (not preceded by OPEN or LOCK command) in which case the FREF field is used only to find a free FCB on any drive, or by an active user. When issued in standalone mode, the connection is severed upon command completion.

I/FDP STATUS FLOPPY Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:CMD_STATFL_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number (Drive number, if standalone, else from OPEN/LOCK command.
CC - byte 8:	Command code
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

byte 12:	Drive number

RESPONSES:

Command completed (RC=EQ$IFDP:RC_DONE).

byte 13:	ONLINE/OFFLINE indicator (Ø=OFFLINE). If offline, other fields are not returned.
bytes 14-21:	Volume label (ASCII)
byte 22:	Disk type: Software (EQ$IFDP:DTYPE_PGM); Logger (EQ$IFDP:DTYPE_LOG); General (EQ$IFDP:DTYPE_GEN).
byte 23:	Software revision (Ø if not software disk)
byte 24:	Software release (Ø if not software disk)
byte 25:	Side/density byte (see OPEN command)
byte 26:	Next available sector of Reserve Sectors Track
bytes 27-29:	Free sectors count
bytes 30-31:	ERP entry count
bytes 32-33:	Data records count
bytes 34-35:	FDR (subdirectory) records count
bytes 36-37:	File count
bytes 38-101:	Volue label identification text (ASCII)

OTHER RESPONSES:

See QUERY ACCESSORS command.

I/FDP FILE STATUS Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:STATFI_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number (from OPEN command, or Drive number, if standalone)
CC - byte 8:	Command code (EQ$IFDP:CMD_STATFI). Return file statistics
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

byte 12:	Drive number
byte 13:	Unused
bytes 14-29:	Filename (see OPEN command)

RESPONSES:

Command completed (RC=EQ$IFDP:RC_DONE)

byte 30: Number of accessors currently using file (issued OPENs)
byte 31: File validity byte. Bits:

EQ$IFDP:DISK_DELETE -	File marked deleted. (file may not be found if Master Directory slot reused).
EQ$IFDP:DISK_PHYBAD -	File marked physically bad (caused I/O error upon reading).
EQ$IFDP:DISK_LOGBAD -	File marked logically bad (pointer invalid.

bytes 32-34:	File length (in bytes)
byte 35:	Number of FDR records in file
bytes 36-37:	Data records count
bytes 38-40:	Creating node, port, and module of file Ø,Ø,Ø=Prime emulator
bytes 41-43:	Last reading node, port, and module (last OPEN with MODE=READ)

bytes 44-46:	Last writing node, port and module (last OPEN with MODE=WRITE, UPDATE)
bytes 47-69:	Optional file text (ASCII)

OTHER RESPONSES:

See QUERY ACCESSORS command.

I/FDP TEST TRACK Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

```
    PL - byte  0:  Packet length (EQ$IFDP:CMD_TESTRK_XLEN)
    Dn - byte  1:  Destination node
    Dp - byte  2:  Destination port
    Dm - byte  3:  Destination module (EQ$IP$MDT:COMMAND_FDP)
    Sn - byte  4:  Source node
    Sp - byte  5:  Source port
    Sm - byte  6:  Source module

  FREF - byte  7:  File Reference number (from LOCK command)
    CC - byte  8:  Command code (EQ$IFIDP:CMD_TESTRK) Non-destructive track
                   read.
    RC - byte  9:  Return code
  SEQ# - byte 10:  Sequence number (modulo 8)
IOERR# - byte 11:  Count of I/O errors during command execution
```

PARMS:

```
         byte 12:  Track to test
         byte 13:  Drive to test
```

RESPONSES:

Command complete (RC=EQ$IFDP:RC_DONE):

```
 bytes 14_40:  Returned status bytes (1 per sector):

               EQ$IFDP:TTRK_SEEKERR - Seek ERROR occurred reading sector.
               EQ$IFDP:BADREC       - Record was readable after multiple
                                      retries.  Retry value is low nibble
                                      of status byte.
               EQ$IFDP:FREEREC      - Record is marked FREE (available)
                                      in FSM.
               EQ$IFDP:RESERVE      - REserved, should always be zero.
```

Command incomplete (RC=EQ$IFDP:RC_NOTLOCK). Requestor has not LOCKED drive. This command is valid from all LOCK (CP-type) states.

OTHER RESPONSES:

See OPEN command.

NOTE: I/O errors encountered during execution of a test track command do not generate system reports.

I/FDP CHANGE OPTIONAL FILE TEXT Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte	0:	Packet length (EQ$IFDP:CMD_CHFTX_XLEN)
Dn - byte	1:	Destination node
Dp - byte	2:	Destination port
Dm - byte	3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte	4:	Source node
Sp - byte	5:	Source port
Sm - byte	6:	Source module
FREF - byte	7:	File Reference number (from LOCK response)
CC - byte	8:	Command code (EQ$IFDP:CMD_CHFTX)
RC - byte	9:	Return code
SEQ# - byte	10:	Sequence number (modulo 8)
IOERR# - byte	11:	Count of I/O errors during command execution

PARMS:

bytes 12-34:	New file text (blank fill, ASCII)

RESPONSES:

Command complete (RC=EQ$IFDP:RC_DONE). Optional text replaced and record(s) written disk.

OTHER RESPONSES:

See WRITE command.

I/FDP CHANGE VOLUME LABEL TEXT Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:CMD_CHVTXT_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number (from LOCK response)
CC - byte 8:	Command code (EQ$IFDP:CMD_CHVTXT)
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

byte 12 :	Drive #
Bytes 13-20:	Volume
bytes 21-84:	New optional label volume label text (ASCII, blank filled (X'20').

RESPONSES:'

Command complete (RC=EQ$IFDP:RC_DONE). VLR1 record updated and written to disk.

Command incomplete (RC=EQ$IFDP:RC_INV_LEN). Packet length incorrect.

Command incomplete (RC=EQ$IFDP:RC_NOFILE). Volume label does ot match for specified drive.

Command incomplete (RC= EQ$IFDP:INV_DRV). Invalid drive #.

OTHER RESPONSES:

See LOCK command.

I/FDP CHANGE VOL-ID Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte	0:	Packet length (EQ$IFDP:CMD_CHVOL_XLEN)
Dn - byte	1:	Destination node
Dp - byte	2:	Destination port
Dm - byte	3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte	4:	Source node
Sp - byte	5:	Source port
Sm - byte	6:	Source module
FREF - byte	7:	File Reference number (from LOCK response)
CC - byte	8:	Command code (EQ$IFDP:CMD_CHVOL)
RC - byte	9:	Return code
SEQ# - byte	10:	Sequence number (modulo 8)
IOERR# - byte	11:	Count of I/O errors during command execution

PARMS:

byte 12 :	Drive number
bytes 13-20:	Current VOL-ID
bytes 21-28:	New VOL-ID (ASCII, left adjusted, blank filled (X'20')

RESPONSES:

Command complete (RC=EQ$IFDP:RC_DONE). New volume label copied to VLR1 and VLR2 records and updated to disk.

Command incomplete (EQ$IFDP:RC_NOFILE). Current VOL-ID does not match specified value.

Command incomplete (EQ$IFDP:RC_INV_LEN). Current or new VOL-ID has incorrect length.

OTHER RESPONSES:

See OPEN command.

I/FDP DELETE Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:CMD_DELETE_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number (Drive number, if standalone, else from OPEN/LOCK command)
CC - byte 8:	Command code (EQ$IFDP:CMD_DELETE) Delete on existing disk file.
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

byte 12 :	Drive number, if standalone, else zero.
byte 13 :	Unused. May be used by requestor for command identification purposes.
bytes 14-29:	Filename (see OPEN command).

RESPONSES:

Command completed (EQ$IFDP:RC_DONE). All data records and subdirectory records have been returned to the availability pool (FSM); the Master Directory entry has been marked available, and all records updated have been written to disk.

Command incomplete (EQ$IFDP:RC_DONE_WITH_ERRORS). While releasing records a logical or physical inconsistency was detected. The process was discontinued and the Master Directory entry for the file marked available and logically or physically impaired.

OTHER RESPONSES:

See OPEN command.

I/FDP ADD Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL - byte 0:	Packet length (EQ$IFDP:CMD_ADD_XLEN)
Dn - byte 1:	Destination node
Dp - byte 2:	Destination port
Dm - byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn - byte 4:	Source node
Sp - byte 5:	Source port
Sm - byte 6:	Source module
FREF - byte 7:	File Reference number (Drive number)
CC - byte 8:	Command code (EQ$IFDP:CMD_ADD). Add a block of data to existing file.
RC - byte 9:	Return code
SEQ# - byte 10:	Sequence number (modulo 8)
IOERR# - byte 11:	Count of I/O errors during command execution

PARMS:

byte 12:	Data length (multiple of 16; less than 128)
byte 13:	Unused
bytes 14-29:	Filename (see OPEN command)
byte 30-n	Data

RESPONSES:

Command completed (RC=EQ$IFDP:RC_DONE). Data added to file and written to disk. Command response may be delayed indefinitely if drive specified is LOCKED when command received by I/FDP (in which case it was queued until the specified drive was UNLOCKed). Command will also be queued if no FCBs are available when received and processed when an FCB becomes available. The file is automatically CLOSED upon command termination.

NOTE: This implies data sequentiality may be lost in rare instances.

ADD command release the FCB upon completion.

OTHER RESPONSES:

See OPEN, WRITE, and CLOSE commands.

I/FDP LOG Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL	- byte 0:	Packet length (EQ$IFDP:CMD_LOG_XLEN)
Dn	- byte 1:	Destination node
Dp	- byte 2:	Destination port
Dm	- byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn	- byte 4:	Source node
Sp	- byte 5:	Source port
Sm	- byte 6:	Source module
FREF	- byte 7:	File Reference number
CC	- byte 8:	Command code (EQ$IFDP:CMD_LOG). Add data to System Logging File.
RC	- byte 9:	Return code
SEQ#	- byte 10:	Sequence number (modulo 8)
IOERR#	- byte 11:	Count of I/O errors during command execution

PARMS: (Written to disk)

byte 12:	Initiator node
byte 13:	Initiator port
byte 14:	Initiator module
byte 15:	Initiator port type (EQ$MISC:GTYPE_XXX)
bytes 16-19:	Reserved (used by I/FDP for time-stamping)
byte 20:	Operation code (e.g., OPEN, LOCK, CALL, HANGUP, CLOSE, UNLOCK)
bytes 21-27:	Varied by port type

For I/FDP LOG entries:

byte 21:	Drive number
byte 22:	Read command count MSB (CLOSE, UNLOCK) Master Directory Sequence number (OPEN) Lock mode (LOCK)
byte 23:	Master Directory Index (OPEN) Read command count LSB (CLOSE, UNLOCK) Unused (LOCK)
byte 24:	Master Directory Accessor count (OPEN) Write command count MSB (CLOSE, UNLOCK) Unused (LOCK)
byte 25:	Master Directory validity indicator (OPEN) Write command count LSB (CLOSE, UNLOCK) Unused (LOCK)
byte 26:	Total I/O performed MSB (CLOSE, UNLOCK) Unused (OPEN, LOCK)
byte 27:	Total I/O performed LSB (CLOSE, UNLOCK) Unused (OPEN, LOCK)

NOTE: The UNLOCK command code is recognized by the I/FDP as the only command code which is negative. The operation code, if UNLOCK is composed of two components:

B'XXXXXXXX' where "1" indicates unlock, and "XXXXXXX" represents the UNLOCK node.

For ITPs:

byte 21:	Call type (e.g., leased, TP dial)
byte 22:	Calling or called node
byte 23:	Calling or called port
byte 24:	Thread number (multi-threaded ports)
bytes 25-27:	Unused

For NPs:

byte 21:	Remote node
byte 22:	Remote port
bytes 23-24:	Speed
bytes 25-27:	Unused

I/FDP FORCE Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL	- byte 0:	Packet length (EQ$IFDP:CMD_FORCE_XLEN)
Dn	- byte 1:	Destination node
Dp	- byte 2:	Destination port
Dm	- byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn	- byte 4:	Source node
Sp	- byte 5:	Source port
Sm	- byte 6:	Source module
FREF	- byte 7:	File Reference number (from LOCK command)
CC	- byte 8:	Command code (EQ$IFDP:CMD_FORCE). FORCE disconnection of active requestor.
RC	- byte 9:	Return code
SEQ#	- byte 10:	Sequence number (modulo 8)
IOERR#	- byte 11:	Count of I/O errors during command execution

PARMS:

byte 12:	Drive of requestor being forced
byte 13:	Node of requestor being forced
byte 14:	Port of requestor being forced
byte 15:	Module of requestor being forced

RESPONSES:

Command completed (RC=EQ$IFDP:RC_DONE). Requestor linkage CLOSED/UNLOCKED. Requestor notified.

OTHER RESPONSES:

See QUERY ACCESSORS command.

I/FDP FORCE DRIVE REINITIALIZATION Command Layout

PL	Dn	Dp	Dm	Sn	Sp	Sm	FREF	CC	RC	SEQ#	IOERR#	PARMS

PL -	byte 0:	Packet length (EQ$IFDP:CMD_FRCINIT_XLEN)
Dn -	byte 1:	Destination node
Dp -	byte 2:	Destination port
Dm -	byte 3:	Destination module (EQ$IP$MDT:COMMAND_FDP)
Sn -	byte 4:	Source node
Sp -	byte 5:	Source port
Sm -	byte 6:	Source module
FREF -	byte 7:	File Reference number (from LOCK command)
CC -	byte 8:	Command code (EQ$IFDP:CMD_FRCINIT)
RC -	byte 9:	Return code
SEQ# -	byte 10:	Sequence number (modulo 8)
IOERR# -	byte 11:	Count of I/O errors during command execution

PARMS:

byte 12: Drive to initialize

RESPONSES:

Command completed (RC=EQ$IFDP:RC_DONE). Drive initialization started all requestors on drive (if online) are automatically forced before initialization starts (including requestor) for that drive.

Command incomplete (RC=EQ$IFDP:RC_DONE_WITH_ERRORS). Drive already online.

OTHER RESPONSES:

See QUERY ACCESSORS command.

6.7.3 Initialization

I/FDP initialization is provided as two components: (1) port or software initialization, and (2) disk verification/initialization.

Port initialization includes allocation of the FCBs and FID, the CHQ, BTCBs and MDT, setting/clearing of accumulators and counters for statistics and system variables, configurational memory (CMEM) retrieval and storage, and initiation of disk verification/initiation. Port initialization is called by the operating system during IPOS user initialization.

Disk verification/initialization includes physical and logical verification of all mounted and ready floppy drives and disks, determination of floppy volume IDs and status, creation of the RST and VMDT, disk verification (normal or optional extended verification), notification to the report node of the drive and floppy initial status, and enabling of the software which validates and queues received commands. Disk verification is forked by port initialization.

Normal disk verification includes checking the validity of the MDR records (sequence number/pointer checking), the ERP records (sequence number/pointer checking), the FSM, and VLR-2 counts. Optionally, under control of a CMEM value, extended initialization may be initiated. Extended initialization includes normal initialization plus checking of each FDR record for valid directory and data record pointers. The data records are not actually read during this operation.

Sector recovery is also part of initialization. If verification detects a logical inconsistency (possibly as the result of the deletion of a logically or physically impaired disk file), a report is issued to the report node, and, if CMEM so specifies, the error will be corrected by software. Correction takes the form of rewriting one or more sectors, based on information compiled during the initialization process. If CMEM does not indicate that errors should be corrected as detected, no action, other than the report, will be taken.

Note that if a drive is not readied when the port starts disk verification/initialization, initialization of that drive will await its being readied. An alarm is issued to the Report node, in this case.

Entry Point - IFDP$INIT:START

Function

Port initialization

Entry Condition

None

Exit Conditions

All registers unset.
Disk verification/initialization forked.

Entry Point - IFDP$INIT:DISK_VER

Function

Normal disk verification
Optional (CMEM) extended disk verification

Entry Condition

None

Exit Conditions

All registers unset.
All mounted and enabled drives initialized.
Initial drive and disk status reported to Report node.

6.7.4 Protocol Management

Every command received by the I/FDP contains a sequence number field which is used to manage the ARQ and flow control. Certain commands (OPEN, CREATE, DELETE) use this field to establish flow control parameters. Other commands (ADD, LOG), ignore this field because their nature implies a one-time mode of accessing the I/FDP.

OPEN, CREATE, and DELETE commands, when received from a requestor not already active, set the starting sequence number. Commands from this requestor are expected to sequentially increment this field (module 16). If a sequence number is found to be invalid (nonsequential to the last command received from this requestor), a NAK bit and the last correctly received sequence number are used to set the return code of the command, and the response returned to the requestor. All commands received for a requestor which has the NAK outstanding indication set, except a command with the missing sequence number (lost, rejected, or ignored) are discarded. The NAK outstanding indication is unset when the correct sequence number is finally received and validated.

Flow control is established by discarding commands, independent of their sequence number, if eight outstanding commands already exist (are validated and queued) for that requestor. This procedure will cause the next command accepted from that requestor to have an invalid sequence field, and thus be NAK'd.

An additional protocol function involves command which are valid but which produce error results (e.g. a RESET beyond the end-of-file). If the error does not cause the severance of the FCB/requestor connection (implicit CLOSE), an indicator is set, and the response returned to the requestor with the command error code. When the indicator is set, commands from this requestor, other than a command with the same sequence number as the error command, are discarded.

To insure that no requestor hangs (possibly as the result of a single path node failure or a port crash of the requestor), the I/FDP runs a command timer. Expiration of this timer causes the file being accessed to be CLOSEd (or the drive to be UNLOCKed) and the requestor/FCB linkage to be sundered. Ports expecting long delays between commands (other than LOG or ADD commands) may hold the connection by sending NOP commands, or the timer, which is CMEM defined, may be increased. The former option allows specialization for a single port without global extention of the timer. The command timer is reset and disabled when a valid command is received from a requestor, whether or not that command is actually executed. The timer is enabled (started when the command begins actual processing.

Entry Point - IFDP$RCV:START (batch task)

Function

Receive requestor commands
Protocol verification
Satisfaction of commands not requiring I/O

Entry Conditions

One or more commands queued by IPOS

Exit Conditions

No commands remaining on batch queue

6.7.5 File Management

I/FDP file management is a function of both the requestor and the I/FDP software. Files are handled in one of 3 modes, depending on the security code specified in the OPEN or CREATE:

Low security - the current subdirectory record is written to disk only when a data record write fills the current subdirectory record, or the file is closed.

High security - the current subdirectory is written to disk after every data record write.

Maximum security - high security, plus every data record write is followed by a read for that record and a verification of that record before declaring the command successful.

The first and current subdirectory records are always resident in memory. When the current subdirectory is reset, if the access mode is read, the record is checked for inconsistent or invalid pointers and/or counter. If an error is detected, the file availability byte is set to indicate a logically impaired disk file. Impaired files can be deleted, however, some records may be temporarily lost until the next I/FDP IPL.

The file management function is responsible for processing protocol validated commands, and issuing multiple I/O requests to perform the command specified action(s). It is also responsible for creating and maintenance of the FRTs. Once I/O has completed, the file management function is also responsible for error detection and correction.

I/O errors to write commands cause the bad sector to be added to the ERP, a new sector obtained from the FSM, and the operation to be retried until either the operation is successful, or no more free sectors are available (at which time the command is handled under disk-full processing). I/O error to read commands cause the operation to be retried "n" times, where "n" is CMEM defined, before the operation is terminated with an error response the record pointer and type indictor added to the ERP, and the file availability indicator set to show the file as physically disabled. If the record was a MDR, or FDR, the file is implicitly closed. If the read operation completes successfully after one or more (but less than "n") retried, and the record is a directory type record, LOG record, or the file open security mode is maximum, the error record is copied to another location and the record added to the ERP.

Satisfied (or physically unsatisfiable) commands are then returned to the requestor. If the command is a CLOSE or UNLOCK, the CHQ is checked for possible entries.

Entry Point - IFDP$FLOPPY:START (batch task)

Function

Command processor (I/O related commands)
File management

Entry Conditions

One or more commands queued

Exit Conditions

No commands queued to batch queue

Entry Point - IFDP$XMT:START (batch task)

Function

Response return routine

Entry Conditions

One or more commands queued

Exit Conditions

No commands queued

6.7.6 Device Control

The I/FDP manages from 1 to 4 floppy disks (single or dual density, single or dual sided), interleaving I/O commands. Requestor commands are broken down to I/O commands by the file management function, and used to set the track, and sector registers, and the drive selector. Each I/O request is single streamed with the next command not being processed until the prior command completes, and the data/status from that command has been saved/analyzed.

Data bytes are transferred to and from the disk via a 1771 (or 1791) using a 128 byte FIFO with CRC verification under hardware control. For write operations, if the data block is less than a sector size, the remainder of a record is zero filled. One interrupt is given for every sector of data bytes transferred.

Data read is stored in memory and a pointer to the memory block saved.

Entry Point - IFDP$FLOPPY:IRQ

Function

Floppy interrupt handler

Entry Conditions

Floppy IRQ enabled and outstanding

Exit Conditions

Floppy IRQ disabled
IFDP$FLOPPY:IO forked

Called By

IPOS interrupt handler

Entry Point - IFDP$FLOPPY:IO

Function

Floppy data/status management

Entry Conditions

Floppy IRQ disabled
Floppy data and or status set
I/O return address stored in memory

Exit Conditions

Floppy IRQ enabled
Control passed to I/O return address

Called By

IFDP$FLOPPY:IRQ

6.7.7 Line Control

When using a floppy disk emulator instead of a real floppy disk, processing is essentially equivalent to that specified in device control (6.7.6), except that an I/O control block is built from the I/O command registers and transmitted, using a Signetics 2651, over a communications line. Data and status are returned as another I/O control block which is then converted into a form equivalent to that from real disk. The I/O control blocks have an end-around checksum which is used to verify both commands (at the remote end) and responses (at the I/FDP). When the checksum does not validate at the I/FDP, the command is resent "n" times before returning an I/O error indication. The communications line may run from speeds of 50 baud to 19.2 K baud.

The I/O control blocks are of the general form:

CC	NODE	PORT	PARMS	CHECKSUM

where "CC" is the command code, "NODE" is the I/FDP node designation, "PORT" is the I/FDP port designation, "PARMS" is a command dependent parameter string (usually including a track and sector designation), and "CHECKSUM" is a two byte end-around checksum.

Responses from the emulator have the form:

RC	NODE	PORT	REPLY DATA	CHECKSUM

where "RC" is the response code (same value as CC), "NODE" and "PORT" are as defined for command blocks, "REPLY DATA" is a response field containing either data bytes, or command completion status, and "CHECKSUM" is a two byte end-around checksum generated by the emulator.

Entry Point - IFDP$FLOPPY:IRQ

Function

I/FDP 2651 interrupt handler

Entry Conditions

2651 IRQ enabled

Exit Conditions

2651 IRQ disabled
IFDP$FLOPPY:IO forked (on complete block)

Entry Point - IFDP$FLOPPY:IO

Function

Floppy data/status handler

Entry Conditions

I/O control block stored
2651 disabled

Exit Conditions

2651 enabled
Control passed to I/O return address

Called BY

IFDP$FLOPPY:IRQ

6.8 Intelligent Network Port (I/NP)

The I/NP module is organized as a set of submodules which are active in areas of unique functionality.

1) Initialization
2) Protocol Management
3) Device Management
4) Mainframe Interface
5) Statistics
6) Exceptions Monitoring

6.8.1 I/NP Data Structures

The I/NP uses frames, enveloped by protocol, to transfer information between D814 mainframe modules (MNL). The frames are of two formats, one for system level communications between connecting I/NP's (Figure 1) and one for customer data transferral (Figure 2). The frames are of the following format:

FLAG	NODE	CC	PARMS	ARQ'	FCS

Figure 1

FLAG	NODE	FTI/SEQ	DATA	ARQ'	FCS

Figure 2

The fields are:

FLAG - One byte (X'7E') start of frame marker and close of prior frame (if any) marker.

NODE - One byte initiating I/NP node number.

CC - One byte command code of the form B'1xxxxxxx'. The command codes are:

1) Link Init (LI)
2) Link Down (LD)
3) Assume Master (AMSTR)
4) Assume Secondary (ASEC)
5) Delay Determine (DD)
6) Delay Determine Response (DDR)

PARMS - "N" byte parameter field.

<u>ARQ</u> - One byte ARQ control field composed of two segments:

1) 1 bit (high order) ACK/NAK indicator

2) 7 bit sequence number (only six least significant bits are used).

<u>FTI/SEQ</u> - <u>F</u>rame <u>T</u>ype <u>I</u>ndicator: One byte field:

1) 2 bit FTI (see section 9, Mainframe Network Link (MNL) for FTI values).

2) 6 bit Link frame sequence number.

<u>DATA</u> - "N" bytes of network data (internal frames)

<u>FCS</u> - 16 bit Frame Check Sequence

<u>NOTE</u>: For system level frames, the software protocol fields (not generated under hardware control) are:

NODE CC ARQ'

For customer data transferral (also referred to as link data transferral):

NODE FTI/SEQ ARQ'

6.8.1.1 ARQ Buffers

The I/NP uses tables of ARQ buffer pointers (the Retransmit Queue) to track point-to-point link data transferral. System level frames are not kept in the Retransmission queue. Maintenance of the table is performed by the protocol submodule. The table is of the form:

BUF 1 PTR	BUF 2 PTR	...	BUF N PTR

Each entry in the table is a pointer to an IPOS byte queue in which the customer data (in MNL Internal Frame format) and the protocol fields have been stored, or zero if that slot has been acknowledged and not refilled with new data. At least one entry (for the current frame under transmission) is always found in the table. Note that the ARQ' protocol field is <u>not</u> stored in the retransmission queue but is fetched from memory when required. This insures that the most up-to-date ARQ information is always sent.

6.8.1.2 Buffer Pools

The I/NP uses a transmitter and receiver pool of buffers to avoid the overhead of constant allocation and reallocation of byte queue headers. The entries in both pools are basic byte queue headers (2 buffers). The pool buffers are daisy chained, with the first entry pointer stored in system memory.

6.8.2 Initialization

I/NP initialization is provided as two components: (1) port initialization, and (2) protocol initialization.

Port initialization includes clearing accumulators and counters for statistics variables, setting of state variables, creation of the ARQ table (Retransmit Queue), creation of the buffer pools, creation of the TCB's and BTCB's for IPOS interfacing, configuration memory retrieval and storage, and device initialization. Selection of NRZ/NRZI encoding, modem signal timing, and line speed are determined, and sent to the 6854, as a part of I/NP device initialization. Port initialization is called by the operating system as part of IPOS user initialization.

Protocol initialization includes synchronization with the remote I/NP over the communications line, and initial notification to the mainframe of protocol initialization completion. Refer to the chart in section 6.8.8, and the D814 Product Functional Specification for further information on I/NP initialization and synchronization sequences. Protocol initialization is invoked by port initialization to initially synchronize I/NP's, and by the protocol manager when resynchronization is required.

Entry Point - INP$INIT:START

Function

Port initialization

Entry Conditions

None

Exit Conditions

All registers unset
CC:I = 0
Mainframe (MNL) notified (I/NP active)

Entry Point - INP$INIT:COMMON_BOOT_RESTART

Function

Protocol initialization

Entry Conditions

I/NP active or I/NP fail must have been sent previously to the mainframe MNL module.

Exit Conditions

All registers unset
Connecting I/NP's synchronized for data transferral.

6.8.3 Protocol Management

Every frame sent or received by an I/NP contains an ARQ field composed of a one bit NAK indicator and seven bits of sequence number (with only the 6 least significant bits used. This field, along with the sequence number field is used to support the I/NP's Go-Back-N ARQ.

6.8.3.1 Protocol Management - Transmission

The I/NP transmitter protocol function (TPF) supplies protocol fields for new network data frames, and tracks available protocol sequence values. This task is forked by the device handling function when a complete link frame has been transmitted, and the 6854 transmitter interrupt is disabled to start the 6854 in idle synchronization (flag fill mode).

The protocol fields (see Figure 2) are preset by the transmission function for the device handling function. The sequence values (SEQ of FTI/SEQ in Figure 2) cycle through the range from 0 to 63, inclusive, with the FTI bits also being passed in this field.

Frame selection is a function of the current I/NP state. In the normal state (LINKUP), when all available values for sequence numbers has been used, TPF initiates automatic retransmission until values become available. A value becomes available as the result of the reception of an ACK. Retransmitted frames, and I/NP synchronization frames are prebuilt, i.e., all bytes required for the frame are already stored in a byte queue. New data frames (sent when not in retransmission mode, or tracking delay timing) have only the link frame protocol fields stored in the byte queue; the data for the frame is taken from the BIC (BIC-1, refer section 6.8.4.1).

Retransmission is also activated when no ACK or NAK is received over a measured period of time, independent of the number of ARQ buffers currently in use. The period of time (in seconds) is specified as a CMEM optional parameter. The I/NP automatically tracks the time required for frames to be transmitted across the communications line, received and processed by the remote I/NP, and for that frame to be returned and processed by the local I/NP. This time (roundtrip delay time) is updated automatically once every minute.

If the I/NP is not in the LINKUP state, frames, including protocol fields, are generated as defined in Appendix H of the D814 Product Specification.

6.8.3.2 Protocol Management - Reception

The I/NP Receiver Protocol Function (RPF) is a batch task which analyzes frames received and queued by the device handler function, verifies sequential integrity, and passes frames to the mainframe interface module (MNL). Received ARQ fields are stored for use by the TPF, as required. The parsed frames are passed in byte queues.

Validation, and responses to received frames, as a function of the I/NP state, are defined in Appendix H of the D814 Product Specification.

6.8.4 Device Management Function

The I/NP uses the Motorola 6854 ADLC communications Chip to send and receive link frames. Interrupts are separated into receive and transmit interrupts by the 6854 interrupt handler, and passed to one of two following submodules.

6.8.4.1 Device Management - Transmission

Frame transmission is performed, a byte at a time, by placing bytes into the 6854 transmission FIFO under interrupt control. Protocol fields are preselected by the TPF (refer to section 6.8.3.1). If the frame itself is not prebuilt by the TPF, as defined by a memory indicator, then the data for that frame is taken from the BIC (BIC-1) and copied to a byte queue (also preset by TPF) as it is transmitted.

Underrun conditions detected by the 6854 cause the frame under transmission to abort (hardware controlled), and the aborted frame to be resent (from the starting flag). When the condition occurs, the entire byte queue and protocol fields sent prior to the condition are sent, as a new frame, before continuing to transmit those bytes (if any) that would have been sent had the underrun not occurred. Thus, any frame passed to the transmitter device handler is guaranteed to be transmitted.

When all data for a frame has been sent, the ACK/NAK ARQ field is retrieved from memory and appended to the frame. The FCS bytes are generated by the hardware, as is the closing flag.

Clear-to-send (CTS) is tracked. When lost a timer (defined in CMEM) is started. If CTS does not return before the timer expiration, the I/NP transitions to the LINK KILL state and transmission of frames is aborted.

When transmission of a frame has completed, the 6854 transmitter interrupt is disabled and the TPF invoked via an IPOS fork to determine the source of the next frame to transmit.

Entry Point - INP$XMT:START

Function

Manage 6854 transmitter interrupts

Entry Conditions

Interrupt Level
Byte queue and next frame type preset
6854 transmitter interrupt enabled

Exit Conditions

All registers unset
TPF forked

6.8.4.2 Device Management - Reception

Frame reception is performed, a byte at a time, from the 6854 receiver FIFO under interrupt control. The fields (bytes) are parsed into a byte queue obtained from the I/NP buffer pool. If no buffers are available, the frame is discarded. Recovery of the frame is a function of the transmission protocol manager. If the node field has the most significant bit set, the frame is discarded. If an overrun condition occurs while receiving a frame, that frame is discarded.

When a complete frame has been received, if the FCS indicator from the 6854 specifies that the frame is invalid, the frame is discarded. If valid, the frame is batch queued to the RPF.

Data Carrier Detect (DCD) is tracked. If lost, a timer (defined in CMEM) is started. If the timer expires prior to the time DCD returns, the I/NP transitions to the LINK KILL state.

Entry Point - INP$RCV:START

Function

6854 receiver interrupt handler

Entry Conditions

6854 receiver enabled
Data in 6854 receiver FIFO

Exit Conditions

All registers unset
Parsed frame batched to RPF

6.8.5 Mainframe Interface Function (MIF)

The I/NP passes and receives mainframe Internal frames from MNL using the BIC (BIC-1). When the TPF decides to cause a data frame to be constructed (refer sec. 6.8.3.1), BIC-1 is set to cause an interrupt to the mainframe to notify MNL that data is required. Outbound interrupts (mainframe to I/NP) do not occur, and are not enabled.

On the I/NP receiver side, when a frame has been verified by RPF, it is passed to the mainframe (MNL) over the BIC. Data is placed in the Inbound FIFO as long as a not-full condition exists and data is available. When the full condition occurs, BIC interrupts (Inbound only) are enabled with the interrupt condition set for half-empty notification. When no more data is available, the MIF terminates with the BIC interrupt disabled. The interrupt is reenabled by the RPF when more data becomes available, at which time that data will be transferred to the mainframe.

Whenever data is being placed into the BIC Inbound FIFO, and until the end of data, or full condition occurs, the BIC Inbound FIFO interrupts are disabled.

Entry Point - INP$BTF:START

Function

Transfer data from RPF to mainframe

Entry Conditions

BIC-1 interrupts disabled
Byte queue with data queued

Exit Conditions

All registers unset
BIC-1 full or no more data

6.8.6 Statistics

Frame counts (initial and retransmitted), overhead and user character counts, NAK counts, delay counters, and average retransmission queue size (nonzero entry counts) are maintained. The statistics values are sampled every 4 seconds, and that snapshot passed to a separate internal routine for evaluation. Evaluation is performed by adding the latest statistics snapshot into a running accumulator, and dividing the resultant figure in half, giving a 6 second average of performance. The statistics maintained by the I/NP are:

1) Count of NAK's received.

2) Count of frames retransmitted.

3) Count of overhead and non-overhead bytes transmitted. An overhead byte is any byte generated by the D814 network. Non-overhead bytes are those bytes generated directly by the customer terminal.

4) Average ARQ size. Number of entries in the Retransmission queue.

5) Average roundtrip delay time. The roundtrip delay time is the time required to transmit a frame to a remote I/NP, have that frame received and reflected back by the remote I/NP, and for that frame to be received.

6) Buffer utilization. The ratio of buffers in use to the number of available buffers, expressed as a percentage.

Requests for the average accumulator values are received (and returned) in IPOS addressed packets. Requests received during link initiation, or during link recovery receive a not-ready indication.

The I/NP also sends a specialized statistics packet to the mainframe (MCC) every 30 seconds. This packet contains overhead and user character counts over the 30 second period and is used as part of D814 congestion control.

6.8.7 Exceptions Monitoring Function

The I/NP performs periodic evaluations of exceptions parameters (as specified in CMEM), and hardware failures in the 6854. The CMEM parameters are:

1) Error Density - Number of retransmitted frames (per second, averaged).

2) Processor Loading - Ratio of the number of processor cycles used, to the cycles available, expressed as a percentage.

3) Buffer Utilization - Ratio of the number of buffers in use, to the number of possible buffers available, expressed as a percentage.

4) Data Rate - The number of bytes of user data transmitted (divided by 100) per second, averaged.

If the value of the periodic evaluation exceeds the CMEM parameter value, an alarm (addressed packet) is sent to the Report node, specifying the error and evaluated value.

CTS and DCD are also tracked by the I/NP. If either signal is lost, an alarm is sent to the Report node. If the lost signal remains down "x" seconds (see sec. 6.8.4), an alarm is sent.

Transitions to the LINK UP and LINK DOWN state cause alarm packets to be sent to the Report node with the node and port of both I/NP's and the CMEM defined line speed.

6.8.8 I/NP Initialization Sequences

	Mainframe Network Link	I/NP'		I/NP	Mainframe Network
1	<--I/NP ACTIVE			I/NP ACTIVE-->	
2		LINK DOWN-->		<--LINK DOWN	
3		LINK DOWN-->		<--LINK DOWN	<--ACT I/NP
4		LINK DOWN-->		<--LINK INIT	
5	ACT I/NP->	LD-->		<--LINK INIT	
6		LINK INIT-->		<--LINK INIT	
7		LINK INIT-->		<--ASSUME SECONDARY	
8		ASSUME MASTER-->		<--ASSUME SECONDARY	
9		ASSUME MASTER-->		<--DF(-1,0)	-->LINK UP <--IF(0)
10	LINK UP<-- IF(0) <-- DF'(0)--->	DF'(0,0)-->		<--DF(-1,1)	

11	IF'(1)--->
	IF(1)<----
		DF'(1,1)-->		<--DF(0,2)	-->IF'(0)

12	IF'(2)--->
	IF(2)<----
		DF'(2,2)-->		<--DF(1,3)	-->IF'(1)

			* * *		
13			FRAME LOST		

14	IF'(X)--->
	IF(Y)<----
		DF'(X,X)-->	<--DF(X-1,Y+2)		<---IF(Y+2)
					--->IF'(X-1)

15	IF'(X+1)->	DF'(NY,X+1)-->	<--DF(X,Y+3)		<---IF(Y+3) --->IF'(X)
16	IF'(X+2)-->	DF'(Y,X+2)--->	<--DF(X+1,Y+1) <--DF(X+1,Y+2)		--->IF'(X+1)

```
17  IF'(X+3)-->                                            --->IF'(X+2)
    IF(Y+1)<--  DF'(Y+2,X+3)->   <--DF(X+2,Y+3)

18  IF'(X+4)-->                                            --->IF'(X+3)
    DF(Y+2)<--  DF'(Y+3,X+4)->   <--DF(X+3,Y+4)      <---IF(Y+4)

                                   *
                                   *
                                   *
(18*)                  MNL DECIDES TO ABORT LINK

19  ABORT LINK-->                                          --->IF'(N)
    IF(M)  <-- DF'(M,N+1)-->     <--DF(N,M+1)        <--IF(M+1)

20                                                         --->IF'(N+1)
               LINK DOWN-->      <--DF(N+1,M+2)      <---IF(M+2)

21             LINK DOWN-->      <--LINK DOWN        --->LINK FAIL

                                   *
                                   *
                                   *
          STEPS 3 THROUGH 12 ARE REPEATED TO RESYNCHRONIZE


                                   *
                                   *
                                   *
               CTS OR DCD DRPS OR 6854 CLOCK FAILS

22  LINK FAIL<--                                       -->IF'(A)
    IF'(A+1)--->      LINK DOWN-->    <--DF(A,B) <--IF(B)

23                    LINK DOWN-->    <--LINK DOWN   --->LINK FAIL
                                   *
                                   *
                                   *
         STEPS 3 THROUGH 12 ARE REPEATED TO RESYNCHRONIZE
```

Note: Although this chart implies simultaneity, the timing is independent, and asynchronous.

Abbreviations:

1) Underlined names are IPOS addressed packets.

2) DF(AM,N) is the representation of a data frame (I-frame) whose sequence number is "N" and whose ACK field is "M"; if "A" is blank, the field is an ACK, if "A" has the value "N", the field is a NAK.

3) IF(x) is the graphical representation of the data from (or for) link data frame DF(AM,x).

Explanation:

Step 1 - Both I/NPs have just been IPL'd. An I/NP ACTIVE address packet is sent from the I/NP to the mainframe.

Step 2 - The I/NPs transmit LINK DOWN (LD) link frames. I/NP states are LINK DOWN.

Step 3 - The second mainframe has decided to bring up its I/NP. To do so, it sends the I/NP an ACTIVATE I/NP (ACT I/NP) addressed packet.

Step 4 -The second I/NP transitions state to LINK INIT. The first I/NP, not having received an ACTIVATE I/NP remains in the LINK DOWN state.

Step 5 - The first mainframe decided to bring up its I/NP (see Step 4).

Step 6 - Both I/NP's are transmitting LINK INIT frames.

Step 7 - The second I/NP has assumed the secondary role in the synchronization. It transitions state to LINK SECONDARY INIT. The first I/NP, using the same algorithm, has decided it will assume the primary role. Note, however, that it does not change state until Step 8.

Step 8 - The first I/NP, having received an ASSUME SECONDARY frame now transitions state to LINK MASTER INIT and transmits ASSUME MASTER frames. The second I/NP is still in the LINK SECONDARY INIT state.

Step 9 - The second I/NP, having received the first I/NP's ASSUME MASTER has now verified full duplex communications capability and is ready to transmit data. It notifies the mainframe that it is ready using a LINK UP addressed packet and changes state to LINK UP. The mainframe provides data for the first I-frame (data frame). The first I/NP is still in the LINK MASTER INIT state.

Step 10 - The first I/NP, having received the second I/NP's data frame has established full duplex communications capability and is ready to transmit. It notifies the mainframe (see Step 9) and passes the data to the mainframe. It takes data from the mainframe to send its first data frame. The state transitions to LINK UP.

Step 11 - The I/NPs are in normal communications mode. The states are LINK UP. Data is being taken from the respective mainframe, enveloped with protocol fields (sequence and ACK/NAK) and transmitted to the remote I/NP.

Note: Despite the timing implied by the chart, both I/NP's are transmitting independently. This means that received sequence numbers are sequential, but that the ACK/NAK fields may or may not be sequential. When two frames are received by an I/NP before it completes transmitting one frame, the ACK/NAK number returned when the frame being transmitted completes is two higher than the previous ACK/NAK field transmitted by that same I/NP.

Step 12 - Transmission and reception continues asynchronously and independently, as long as received sequence fields are sequential.

Step 13 - A frame is lost. The receiving I/NP (the first I/NP) does not see sequence number "Y+1", but instead sees "Y+2". The frame could be lost through line noise garbling, or have been discarded by the receiving I/NP as the result of a buffer shortage.

Step 14 - Sequence number "Y+1" is missed by I/NP's, the second I/NP continues normally.

Step 15 - INP' sends a NAK. The last valid sequence number received ("Y") is returned. The second I/NP is still transmitting normally.

Step 16 - The second I/NP has received the NAK of I/NP'. It begins retransmitting from sequence number "Y+1", and passes the data received to the mainframe. ARQ buffers through "Y" are freed by the second I/NP.

Step 17 - I/NP' now receives frame "Y+1" and "Y+2". "Y+1" readies I/NP' to receive additional data frames. Note that the ACK/NAK field is doubly incremented. Asynchronous normal communications is restored. I/NP' continues retransmitting frames received prior to the NAK.

Step 18 - All retransmitted frames from the second I/NP have been sent, new data is now requested from the mainframe for transmission.

Step (18*) - The mainframe decides to bring the link down. (Possibly to adjust inconsistent software).

Step 19 - The mainframe (MNL) has decided to shutdown its I/NP. It does so by sending an ABORT LINK addressed packet.

Step 20 - The first I/NP transitions state to LINK KILL (effectively LINK DOWN). The second I/NP, unaware of the change of state, continues sending normally.

Step 21 - The second I/NP receives the LINK DOWN frame and transitions state to LINK KILL. It notifies the mainframe of the transition and transmits LINK DOWN frames. Both I/NP's will remain down until an ACTIVATE I/NP is received from their respective mainframes.

Step 22 - Having detected a hardware error (or no frames from the remote I/NP for an extended period of time) the first I/NP declares itself dead. It notifies the mainframe and transitions state to LINK KILL. The second I/NP remains in the LINK UP state.

Step 23 - The second I/NP sees the LINK DOWN frame and transitions state to LINK KILL. Both I/NP's are now shutdown.

6.9 Intelligent Group Band Network Port (I/GBNP)

The I/GBNP contains two microprocessors, one on the I Engine 2 (Engine) card, and one on the V.35 Bit-oriented Protocol (V-Bit) card. Although both processors share access to memory and the MC6854 Advanced Data Link Controller (ADLC), they are used to perform separate functions. The Engine processor receives data to be transmitted from the Mainframe Network Link (MNL) software via the Bus Interface Chips (BICs), and places it into memory buffers. The V-Bit processor acts as a Direct Memory Access (DMA) controller between the Engine's memory and the ADLC transmitter registers. It also performs a DMA function from the ADLC's receiver to the Engine's memory. The Engine processor moves characters from memory via the BICs to MNL. The remainder of the port functions are carried out in the Engine processor.

The I/GBNP software subsystem is organized as groups of modules which are responsible for unique functional subsets of the I/GBNP. The groups of modules are:

1. Intelligent Port Operating System
2. I/GBNP Initialization
3. Configuration Management
4. Statistics Management
5. Monitoring Module
6. Protocol Management and Data Movement
7. V-Bit Diagnostic Program

6.9.1 Design Considerations

The list which follows explains some decisions which were made in the design of the I/GBNP and the factors which influenced the choices.

Frame Size - The size of the frames sent over the group band network link will be 128 bytes maximum plus 1 flag and 2 CRC bytes for a total of 131 bytes or 1048 bits. (As a future enhancement, the frame size could be made a parameter which could be configured during port initialization with any fixed value which is 128 or less. This parameter would have to be set to a value which would allow the worst case number of unacknowledged frames expected for the link to be stored for retransmission.) At 64K bits per second (8000 bytes per second) the time to transmit this block is 131/8000 second or 16.375 milliseconds. This number directly affects the total worst case delay experienced by a character in passing through a node. Larger blocks increase the delay, but also improve the efficiency by allowing transmission of a higher percentage of MNL data bytes and a lower percentage of overhead (protocol and control) bytes. A full frame of data has 125 data bytes and 6 overhead bytes (only 3 of which are stored in memory) for a total efficiency of 125/131 or 95.4 percent. Since there are 128 possible Frame Sequence Numbers, 16K of memory will be required to store 128 frames of 128 bytes each. This represents two seconds' worth of data in storage for possible retransmission. This figure is reasonable in that two satellite hops introduce the maximum reasonably expected delay in the transmission medium, and each such hop accounts for less than 1 second in round-trip delay. Hence, if the entire retransmission queue fills before an ACK is received, automatically initiating a series of retransmissions is a reasonable course of action. Note that if frames of less than the maximum size are transmitted, the retransmission queue may fill in less than two seconds.

Frame Format - Several differences exist between the frame format chosen for the I/GBNP and the format used in the I/NP. Since the two network ports have no common data rate, this is not seen as a cause for concern. The sequence numbers are seven bits long in the I/GBNP. The extra sequence numbers are necessary to allow a larger number of outstanding unacknowledged frames due to the higher data rate of the port. The Frame Type Indicator (FTI) cannot share a byte with the ACK/NAK field. The FTI will be transmitted between I/GBNP's as a full byte of data. Neither the FTI nor any other MNL data need receive any special handling by the I/GBNP. Link frame boundaries are insensitive to MNL frame boundaries. The ACK/NAK field has been placed near the beginning of the frame in a fixed location. There is no advantage to placing the ACK/NAK field at the end of the frame, since the DMA scheme prohibits changing a frame in memory after it is scheduled for transmission, and the retransmission queue is completely allocated during initialization for all time, hence, no buffer space is saved by having the ACK/NAK field be as current as possible.

6.9.2 I/GBNP Data Structures

Much of the random access memory in the I/GBNP may be accessed (physically) by either of the processors, but access to various locations is restricted by software convention. The programs executed by the two processors are kept separate, as are various local data structures. Certain locations are shared, such as the Transmit Queue and various inter-processor communications bytes, but each of these is written by only one of the processors. The hardware physically prevents the V-Bit processor from accessing the lowest 16K bytes of engine memory space, so this area is used to hold the engine's program.

6.9.2.1 Frame Structure

The I/GBNP uses frames, enveloped by CRC protected protocols, to transfer information between D814 I/GBNPs. Two types of frames are used. The first type of frame is called a supervisory frame (see Figure 6.9-1A). This type of frame only contains data exchanged between I/GBNPs. The second type of frame is called a data frame (see Figure 6.9-1B). This type of frame only contains data exchanged between two MNL modules in remote D814 mainframes. Only data frames are protected by a Go-Back-N ARQ error protection scheme. Supervisory frames are protected against loss (when necessary) by higher levels in the I/GBNP software.

6.9.2.1.1 Supervisory Frames Types

There are six different Supervisory Frames transmitted/received by an I/GBNP (Figure 6.9-1A).

"ccccccc" Code	Frame Type
1	Link/Intialization (LI)
2	Link Down (LD)
3	Assume Master (AMSTR)
4	Assume Secondary (ASEC)
5	Delay Determine (DD)
6	Delay Determine Response (DDR)

The format of each of these frames is given in Figure 6.9-2. Some common notation used frame parameter naming is as follows:

```
S(xxx) = Frame Sender's parameter xxx
R(xxx) = Frame Receiver's parameter xxx          (Validation)
N,P    = Node #, Port #
CFG    = Boot Configuration Number
SW REV = Software Revision Level
SW REL = Software Release
T      = Time (LSB of 24 bit 10 msec clock)
```

(NOTE: S(N) must equal the A field in any frame)

6.9.2.1.2 Data Frame Types

There is only one type of Data Frame (see Figure 6.9-1B). The contents of data frames are passed transparently between MNL modules in D814 mainframes. The only interpretation done by the I/GBNP of MNL data frame information is for statistical purposes; this will be explained later.

I/GBNP Link Frame Formats

Figure 6.9-1A

Supervisory Frame

. . .	FLAG	CCITT CRC	PARAMETERS	CS	A/N	A	FLAG	. . .
			128 Bytes Max!					

$$\text{CS} = \text{B '1}\underbrace{\text{ccccccc}}_{\text{Control Code}}\text{'}$$

Figure 6.9-1B

Data Frame

. . .	FLAG	CCITT CRC	MNL DATA	CD	A/N	A	FLAG	. . .
			128 Bytes Max!					

CD = B 'Ønnnnnnn'

Frame Seq #

For All Frames: A = Initiating Node #

```
A/N = B 'Xnnnnnnn'
         |   V
         |   Seq # of last good frame revd
         |
         |   0 = ACK
         |-->
             1 = NAK
```

Supervisory Frame Formats

Figure 6.9-2

FLAG	CRC	CRC	FILL	HELP TYPE	RT	RP	RN	SSW REL	SSW REV	SCFG	ST	SP	SN	$81	A/N	A	FLAG

LINK INIT

FLAG	CRC	FILLER	SP	SN	$82	A/N	A	FLAG

LINK DOWN

FLAG	CRC	CRC	FILL	HELP TYPE	RT	RP	RN	SSW REL	SSW REV	SCFG	ST	SP	SN	$83	A/N	A	FLAG

ASSUME MASTER

FLAG	CRC	CRC	FILL	HELP TYPE	RT	RP	RN	SSW REL	SSW REV	SCFG	ST	SP	SN	$84	A/N	A	FLAG

ASSUME SECONDARY

FLAG	CRC	FILLER	ST	SP	SN	$85	A/N	A	FLAG

DELAY DETERMINE

FLAG	CRC	CRC	FILL	HELP TYPE	RT	RP	RN	UN-USED	UN-USED	SCFG	ST	SP	SN	$86	A/N	A	FLAG

DELAY DETERMINE

6.9.2.2 Frame Queues

There are two structures for storing frames in the I/GBNP. The first structure is the Outbound Frame Retransmit Queue (RXMTQ). This queue holds copies of frames which have been transmitted but not yet acknowledged. The second structure is the Inbound BIC Data Frame Queue (IBDFQ) which contains data frames for delivery to the D814 MNL module via the BIC data FIFO.

Both frame queues are of fixed length and preallocated for the rapid access required for high speed lines. The RXMTQ contains 64 frame buffers of 128 bytes each, and the IBDFQ contains 16 frame buffers of 128 bytes each. The total storage required for frame queues is thus 10,240 bytes.

6.9.2.3 Statistics and Monitoring Table

The I/GBNP uses the standard statistics routines to collect statistics on the parameters listed below. These parameters are inserted into the standard statistics request packet response in addition to the ones reported by the common software. The number of bytes listed is the number used by the parameter in the statistics packet.

RETRANSMITTED FRAMES	(2 Bytes)
NEW DATA FRAMES TRANSMITTED	(2 Bytes)
MNL BYTES SENT	(2 Bytes)
ROUND-TRIP LINK DELAY (MSEC)	(2 Bytes)
NAKS RECEIVED	(2 Bytes)
UNACKNOWLEDGED FRAMES OUTSTANDING	(1 Byte)
AVAILABLE (UNUSED) BANDWIDTH (CHARS)	(2 Bytes)

6.9.2.4 State Table

The I/GBNP state table really controls the operation of the I/GBNP since it is a state driven subsystem. The I/GBNP can be in the following master states:

1. Link Up (LU) - Normal state when frames are being exchanged between I/GBNPs.

2. Link Down (LD) - I/GBNP not ready or not authorized to bring up link.

3. Link Kill (LK) - The I/GBNP is aborting the link and cleaning up internal data structures. When all internal recovery is complete, the state transitions to the LD state.

4. Link Initiate (LI) - This is the link initialization state used to synchronize bringing up the link between two I/GBNPs.

5. Assume Master (AMSTR)	- In this state the I/GBNP has assumed the role of link master for the link initialization sequence.
6. Assume Secondary (ASEC)	- In this state the I/GBNP has assumed the secondary role during link initialization.

In addition to the master I/GBNP state parameter, the following state parameters and flags are necessary for I/GBNP operation.

1. Last ACK Received	(LAST_ACK_RCVD)
2. Last Frame Transmitted	(LAST_SEQN_XMTD)
3. Nak Request Flag	(NAK_FLG)
4. Last Frame Received	(LAST_BLOC_RCVD)
5. Retransmit Mode Flag	(RXMT_FLG)
6. Next Frame Expected	(NXT_FRM_EXP)

6.9.2.5 Configuration Tables

This table contains the configuration related parameters. These parameters are:

CMEM_GTYP	Generic Port Type
CMEM_STYP	Port Subtype
CMEM_TIMC	Statistics Time Constant
CMEM_SPDH	Configured speed. No relation to actual speed. Not used.
CMEM_SPDL	Not used.
CMEM_MODE	Not used.
CMEM_RTTH	Threshold for retransmit rate system report.
CMEM_BUTH	Threshold for buffer utilization system report (should be CMEM_BUIP).
CMEM_PLTH	Threshold for processor loading system port (should be CMEM_PLIP).
CMEM_NRZI	NRZ/NRZI Encoding Selector
CMEM_TACK	No ACK timeout threshold.

6.9.3 I/GBNP Main Modules

6.9.3.1 I/GBNP IP Operating System

The I/GBNP will use the standard 6809 Intelligent Port Operating System (IPOS/Ø9) described in Section 6.1 of this manual. It is assumed that IPOS/09 IRQ dispatching works using SWI3 for termination, RTI to resume and start tasks.

6.9.3.2 Initialization

The I/GBNP initialization routine executes in the Engine processor with the V-Bit processor held in a reset state. It creates and initializes all the TCBs required for its operation and copies the V-Bit program into the V-Bit card's memory.

It then requests and receives CMEM parameters from the mainframe and sends another packet to the mainframe to report that it is active and ready to try to bring the link up. When it receives a packet instructing it to do so, it enters "Link Initialization" state, initializes the ADLC chip, and releases the V-Bit processor's RESET line. The port then begins sending supervisory frames in an effort to achieve synchronization with the remote GBNP.

6.9.3.3 Configuration Management

The port presently uses its own configuration module, but could be modified to use the system standard routine.

6.9.3.4 Statistics Management

The port uses the standard statistics modules to report statistics upon request and to send system reports when alarm conditions occur.

6.9.3.5 Monitoring Module

This module will bring the link down if any of the following conditions exist for longer than the configured (CMEM_NOAK) threshold period:

CTS Off
DCD Off
No new acknowledgements received.

6.9.3.6 Protocol Management and Data Movement

The Protocol Management and Data Movement functions are executed in both processors and consists of six submodules:

1.	BIC IRQ Control	(BICIRQ)
2.	Outbound BIC Driver	(OBBIC)
3.	Outbound Protocol/DMA Initiation	(OBPDMA)
4.	ADLC DMA Driver (Executed in V-Bit proc.)	(ADMA)
5.	Inbound Protocol/DMA Initiation	(IBPDMA)
6.	Inbound BIC Driver	(IBBIC)

The general structure is diagrammed in Figure 6.9-4. The submodules operate asynchronously from one another. Communication and coordination between the processors and among submodules in the engine processor is accomplished by using RAM flags. The functions of each submodule are described below. All Engine submodules execute at the IRQ level.

6.9.3.6.1 BIC IRQ Control (BICIRQ)

This submodule is entered upon receipt of an interrupt from the BIC to determine which of the BIC Driver submodules to execute.

6.9.3.6.2 Outbound BIC Driver (OBBIC)

This submodule receives control from BICIRQ when the BIC FIFO is half full or completely full (choice to be based on actual experience). It removes bytes of data from the BIC and places them into a buffer in the retransmit queue (RXMTQ). When this buffer becomes full the BIC IRQ is disabled to prevent further characters from being read from the BIC until OBPDMA has scheduled the full buffer for transmission. Since OBBIC and OBPDMA have different functions involving the same buffer in RXMTQ, both of them run with IRQ masked. Since OBBIC only runs upon receipt of a BIC interrupt, it is expected that MNL will keep filling the BIC FIFO with nonzero data frequently enough to prevent a small number of characters placed into the FIFO from experiencing an unacceptable delay while waiting for enough characters to cause an interrupt. Zero characters may not be placed into the FIFO because a zero is used to mark the start of pad characters in a partially full frame.

6.9.3.6.3 Outbound Protocol/DMA Initiation (OBPDMA)

This submodule receives control when an IRQ is received from the V-Bit card. This IRQ occurs when the ADMA submodule has completed transmitting the previous buffer. If a supervisory frame has been built, it is selected for transmission instead of the data frame built by OBBIC. If there is no supervisory frame, OBPDMA terminates the RXMTQ buffer being filled by OBBIC (unless it has already been completely filled) by placing a zero byte in the next character position. For either kind of frame, the current ACK/NAK byte is placed into the buffer. The buffer address is then passed to ADMA for transmission and a new buffer (if available) is set up for OBBIC to fill. BIC outbound FIFO interrupts are then enabled so that OBBIC will start filling the new buffer. If OBPDMA receives control when OBBIC has not started to fill a RXMTQ buffer because of lack of BIC FIFO data, a full-sized frame with no data bytes (first data byte position zero) will be sent. If all 127 RXMTQ data frame entries are full of unacknowledged but already-transmitted frames, OBPDMA switches to retransmit mode.

6.9.3.6.4 ADLC DMA Driver (ADMA)

The ADMA submodule is in constant control of the processor on the V-Bit card. It receives a pointer to a 128-byte buffer in the Engine's memory. It transmits the data contained in the buffer as a single link frame through the M68B54 Advanced Data Link Controller chip. When the last character has been passed to the ADLC chip, an interrupt is sent to the Engine to request another buffer address from OBPDMA.

Along with the transmitter code, ADMA also contains receiver code which accepts received characters from the ADLC chip and places them into a 128-byte buffer in IBDFQ, the address of which is received from IBPDMA. When the buffer has been filled, ADMA interrupts the Engine to cause IBPDMA to run.

6.9.3.6.5 Inbound Protocol/DMA Initiation (IBPDMA)

The IBPDMA module is run when an interrupt is received from the V-Bit processor, to notify the Engine that a buffer in the Inbound BIC Data FIFO Queue has been filled with a received link frame. The ACK/NAK field is read and processed, by releasing RXMTQ buffers as necessary (if it is an ACK), or by changing to retransmit mode (and possibly also releasing some buffers) if it is a NAK. If the frame is a supervisory frame, a task is forked to process it. If it is a data frame, the sequence number is checked and is converted to an ACK/NAK byte to be transmitted in the next outbound frame. The data frame is then linked to a queue to be passed through the BIC FIFO to MNL. If necessary, the BIC FIFO interrupt is enabled.

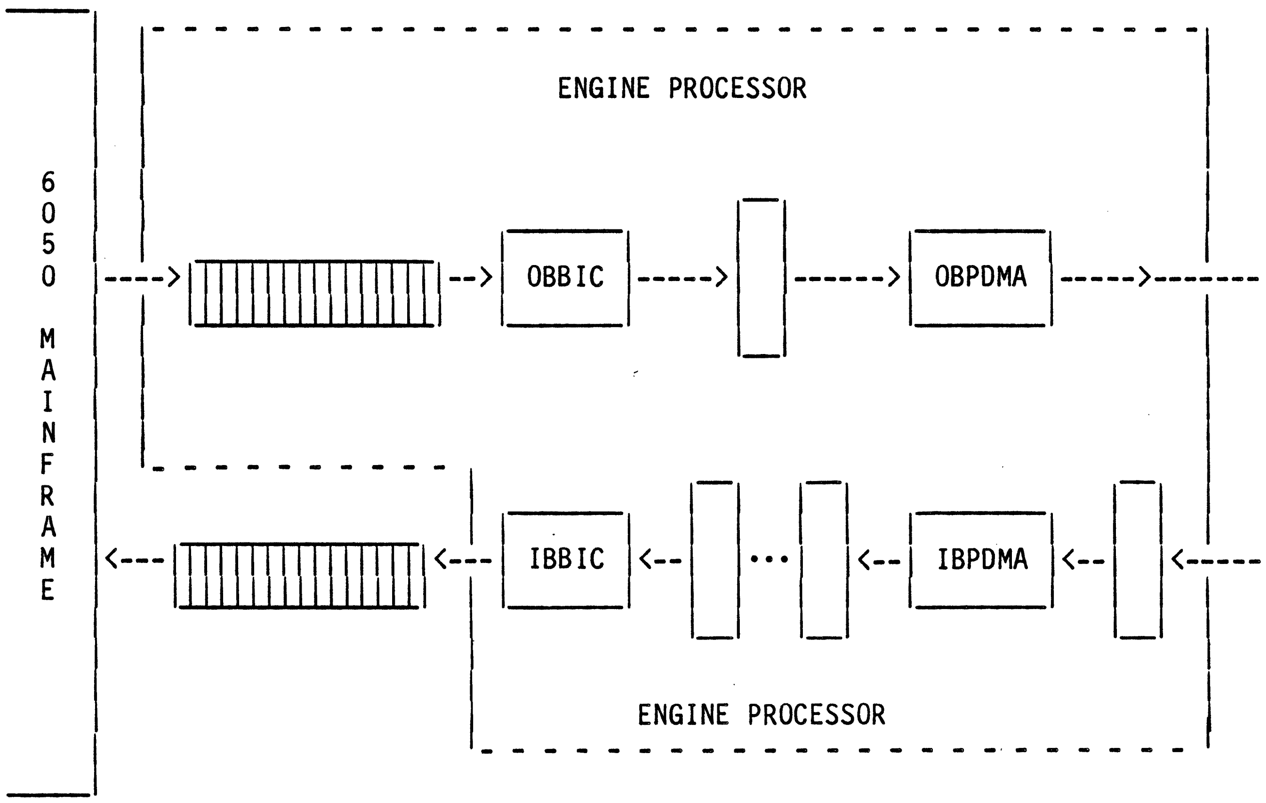
6.9.3.6.6 Inbound BIC Driver (IBBIC)

This routine is executed upon receipt of an interrupt which signifies that the BIC FIFO is ready to accept more characters to be passed to MNL. It removes characters from IBDFQ and places them into the FIFO. If a zero byte is read from IBDFQ, it signifies that the link frame contains no more actual data bytes. If IBBIC finishes processing the last IBDFQ buffer, it disables further BIC Inbound FIFO interrupts.

Group Band Network Port

Data Movement Software Structure

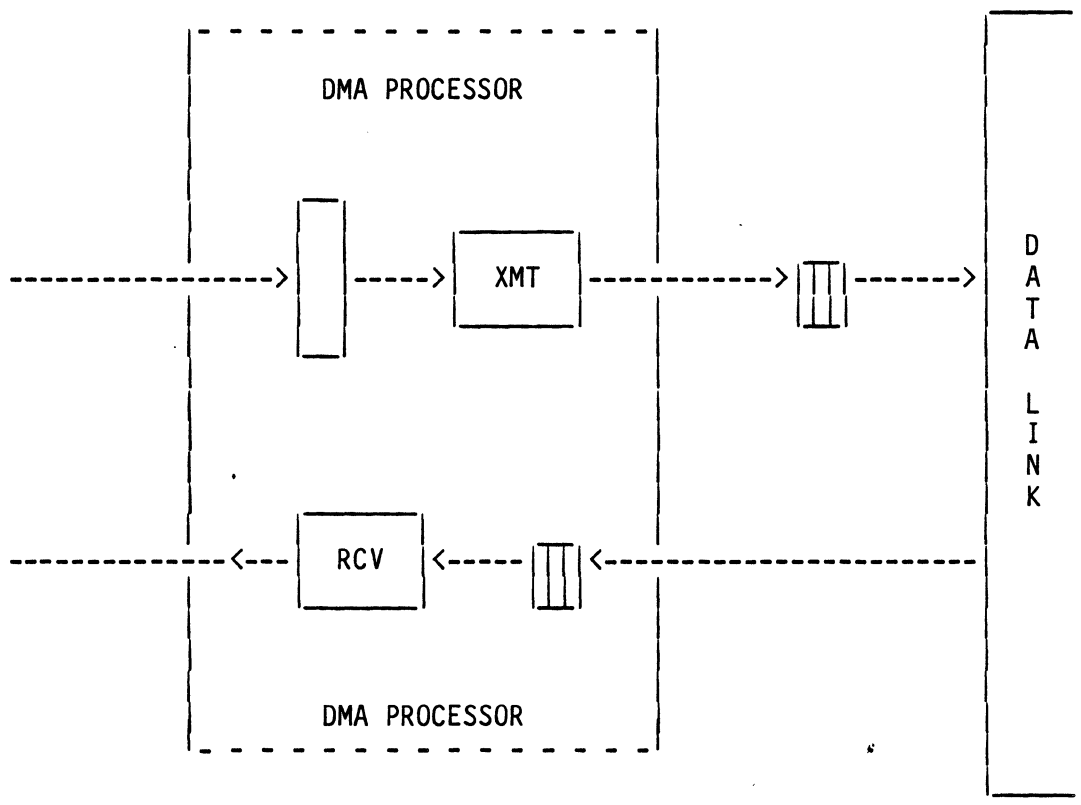
Figure 6.9-4A



Group Band Network Port

Data Movement Software Structure

Figure 6.9-4B



6.9.3.6.7 V-Bit Diagnostic Program (DIAG)

Since the memory on the V-Bit card is not parity-checked by hardware, this module is executed periodically by the background diagnostic task as a port-dependent diagnostic subroutine. It compares the "copy" of the V-Bit program stored in the V-Bit card's memory with the "original" stored in the Engine's memory.

6.10 Intelligent Datagram Port (I/DGP)

6.10.1 Overview

6.10.1.1 Introduction

Any even numbered port except 0 in the D814 network can be configured to be an I/DGP port. In addition, each Control Terminal Port (I/CTP) will have all the functions of the I/DGP. All references to I/DGP in this document are also applicable to the portion of I/CTP that it is in common with.

The I/DGP subsystem consist of the following modules:

1. I/POS (see Section 6.1)
2. User Interface (see PFS)
3. Message Manager
4. Statistic Gathering
5. Special I/O Routines

6.10.1.2 Design Philosophy

The basic goal in the design of the I/DGP is to:

1. Have the user interface as flexible and simple to use as possible so as to aid the unfamiliar without overburdening the sophisticated users.

2. Provide powerful features, but taking the necessary measures to avoid the datagram traffic from significantly impacting the network's performance.

6.10.2 Message Manager

6.10.2.1 Introduction

The set of routines that make up the Message Manager can be divided functionally into two parts:

1. Datagram Transmitter (see Section 6.10.2.5)
2. Datagram Receiver (see Section 6.10.2.6)

Datagrams in the D814 network are transmitted in addressed packets (see Section 6.5), which can be up to 256 bytes of data and control information. The I/DGP allows up to 15 addressed packets per datagram, and the actual number and size of these addressed packets can be easily changed to optimize network efficiency.

The addressed packets (AP's) are sent from the source I/DGP to its mainframe. From the source I/DGP's mainframe the AP's are then sent to the destination mainframe. There all of the AP's that belong to a datagram are first accounted for before any AP is retransmitted to the destination I/DGP.

6.10.2.2 Design Considerations

The major problems in the design of the I/DGP are:

1. Network congestion due to the high priority of addressed packet traffic and the capability of the user in sending a datagram to many destinations simultaneously.

2. Buffer under-run from generating too many addressed packets at once.

The problems are dealt with by:

1. Reducing the number of transmissions required.

2. Generate the addressed packets from a low priority task.

3. Limiting the replies to cases where the number of destinations is small.

6.10.2.3 Datagram Format

The format of the addressed packet for datagrams follows the same format as other addressed packets for the first seven (0 to 6) and last bytes. Additionally, the following information is required:

1. Message number - To uniquely identify each datagram.

2. Sequence and numbering of each address packet in the datagram.

3. Indicator to show if reply is required.

4. Addresses of the I/DGP ports within the node that is to receive the datagram.

Items 1, 2 and 3 are in each AP, and item 4 is only in the first AP. Thus, for each datagram all AP have the first 10 bytes of identical information, except for the sequence number. The last byte of each addressed packet will carry the error code, same as other addressed packets, if the error code bit is '1'.

6.10.2.4 Acknowledgement Format

When requested, the receiving I/DGP port will return an acknowledgement, in the form of a short datagram, to the sending I/DGP. The information in the acknowledgement is to include:

1. Address of receiving port and node.
2. Address of sending port and node.
3. Message number.
4. Time that the datagram was received.

6.10.2.5 Datagram Transmitter

This set of routines performs the following functions:

1. Collect data from the User Interface and time stamp the datagram.
2. Organize the destination addresses for possible grouping of them to reduce the number of transmissions.
3. Set the acknowledgement flag, if required.
4. Generate the addressed packets and send each set of them at specific time intervals, short enough to allow for reasonable response time and long enough to avoid adversely impacting the mainframe's resources.
5. List any expected acknowledgements for datagrams sent, if any, set timer for those ack's to return. If not received within the time allowed, then assume lost and queue a message to inform the user.

6.10.2.6 Datagram Receiver

This set of routines performs the following functions:

1. Assemble received addressed packets into datagrams.
2. Time stamp the datagram.
3. Arrange the datagram into a format suitable to send to the user.
4. Return ack's if required.

6.10.3 Error Messages

The following is a list of possible error messages:

1. Node NN not in the network.
2. Port PP at node NN is not a Datagram or Control Terminal Port.
3. Message MMMM to node NN port PP lost.
4. No acknowledgement received for message MMMM to node NN port PP.
5. Receive Queue full at node NN.
6. Buffers full, cannot accept any message.
7. Invalid node address.
8. Invalid port address.
9. No destination entered.
10. No message entered.
11. Message too long, 512 characters maximum.
12. Message aborted by user, return to output mode.
13. Idle at input mode too long, return to output mode.

6.10.4 Statistic Collection

The following statistics are monitored:

1. Processor loading.
2. Buffer utilization.
3. Maximum buffer size.
4. Line hit (errors on 2651).

6.10.5 Mainframe Interface

After the addressed packets are setup for transmission, they will be sent to the mainframe one at a time. There will be two mainframe submodules to handle these datagram AP's. One module to handle datagrams with specific destination node address and the other to handle datagrams that are to be sent to all nodes in the network, including one to the receiving datagram module in the same node.

Since the specific port address information is contained in the first addressed packet only, the submodule in the destination mainframe has to wait for a complete set of addressed packets before transmitting them to its I/DGP ports. This submodule is expected to return addressed packets in case of transmission errors or missing addressed packets. A time interval will be set, after the arrival of the first addressed packet, for the arrival of a complete datagram; after which it is assumed that the remaining addressed packets are lost.

In the case of transmission to all I/DGP ports in a node, it is assumed that the mainframe submodule that handles this task will determine where the I/DGP ports are in its node.

6.11 Intelligent Asynchronous Terminal Port Protocol Software

This subsystem has been deleted from D814 system software.

6.12 Intelligent Synchronous Terminal Port Protocol Software

This subsystem has been deleted from D814 system software.

6.13 Intelligent Spoofed Synchronous Terminal Port Protocol Software (BSC Version)

The Intelligent Spoofed Synchronous Terminal Port Protocol Module (2780/3780 BSC Version) (I/SSTP-BSC, herein referred to as IS/BSC) is organized as a set of submodules, each of which provides a related set of functions. These functions include:

1) System Initialization
2) Communications Interrupt Handling
3) Network Spoofing Control
4) Inbound Protocol Handling
5) Outbound Protocol Handling
6) Call Manager Interface
7) Statistics and Exception Monitoring

Descriptions of the submodules which perform these functions are found throughout Sections 6.13.1 through 6.13.7.

Because spoofing utilizes protocol intervention to increase throughput, several of the ISBSC modules have intelligence far beyond that of their counterparts in non-spoofed ports. Data is not merely moved - it is verified and generated as well.

The terminals involved in a BSC "conversation" each have a specific role: one is sender of data (MASTER), the other receiver/acknowledger of data (SLAVE). In order for spoofing to be exercised, it is necessary that the spoofing controller be able to identify the role of its user and emulate the functions of the opposing role. Additionally, since this determination is not made until a conversation is begun, the controller must be capable of temporarily suspending protocol intervention when desirable.

The specific tasks associated with each role are identified below:

Master SPOC

Spoofed Data Transmission -

* receives data from master
* checks BCC value, discarding message if incorrect
* generates and sends FACK's (false ACK's) to master for data correctly received.
* NAK's incorrectly received blocks
* generates and sends WACK's to master when outstanding ACK's exceeds calculated maximum
* sends RVI's and EOT's received from slave to master
* transmits data across the network

Normal Data Transmission -

* receives and transmits data in both directions across the network
* generates and sends WACK's to master when required to maintain orderly transmission

Slave SPOC

Spoofed Data Transmission -

* receives and buffers data from the network
* transmits data to slave
* receives slave responses and transmits them across the network (excluding NAK's and WACK's)
* resends NAK'ed data
* generates and sends TTD's to slave when lacking data from master
* sends ENQ when 3-second timeout occurs

Normal Data Transmission -

* receives and transmits data in both directions across the network
* generates and sends TTD's to slave when required to maintain orderly transmission

6.13.1 System Initialization (Submodule ISBSC$INIT)

The function of this submodule is to start IP common modules (i.e., ADCM$, CMM$, FIFO$, FLOW$) and to initialize the IS/BSC data structures. The submodule is composed of three routines. The first routine is called by IPOS to begin initialization of the protocol tasks, the second is forked by the first, and the third routine is a batch task with an entry in the Module Dispatch Table that is scheduled subsequently.

The first routine initializes all protocol state variables and data structures for statistics accumulation, and obtains buffers to be used as permanent TCB's for all other IS/BSC submodules. In addition, it obtains the Holding Buffers used by the 2651 interrupt routines as temporary storage of data to be sent or data that has been received, and forks the second routine (ISBSC$INIT:REQ) before returning to IPOS.

ISBSC$INIT:REQ builds a configuration-request addressed packet and routes it to the mainframe MCM$CMEM module (see Section 5.7). The return of an addressed packet containing the requested information causes scheduling of the third routine (ISBSC$INIT:CONF), that which initializes the remaining data structures and completes the initialization process for the IS/BSC.

Entry Point - ISBSC$INIT:ENTRY

Function

Initialize IS/BSC data structures, start IP common modules, and obtain 2651 Holding Buffers.

Entry Conditions

* None

Exit Conditions

All registers destroyed

Entry Point - ISBSC$INIT:REQ

Function

Build a configuration-request packet

Entry Conditions

* Forked

Exit Conditions

* Terminates

Entry Point - ISBSC$INIT:CONF

Function

Complete IS/BSC data structure initialization based on the information contained in returned configuration-request packet.

Entry Conditions

* Batch Task

Exit Conditions

* Terminates

6.13.2 Communications Interrupt Handling (Submodule ISBSC$COMM)

This submodule is invoked by IPOS to service any interrupt requests from the 2651 Communications Interface Chip and the Auxiliary Control Signal Register. ISBSC$COMM:IRQ scans the contents of the ACS Status Register to identify the requestor, and routes the IRQ to the appropriate IRQ processor.

Interrupts from the 2651 are of three types: Receiver Ready, Transmitter Ready, and Data Set Change. ACS interrupts are modem signal changes, hence are handled in the same way as 2651 Data Set Changes.

Entry Point - ISBSC$COMM:IRQ

Function

Dispatch 2651 and ACS IRQ's to the appropriate interrupt processing routine.

Entry Conditions

* ACS or 2651 IRQ

Exit Conditions

* Terminates

Receiver Ready (IRQ) Routine (RECEIVE)

This routine performs processing necessitated by a 2651 Receiver Ready interrupt. It has 6 basic functions to perform:

1) Move data from the 2651 to the Receive Holding Buffer, encoding when necessary (see Section 6.13.2.1).
2) Detect and record parity and BCC errors by means of the 2653 Polynomial Generator/Checker
3) Update spoofing controller variables
4) Discard blocks received in error
5) Fork spoofing controller modules
6) Fork IBP module

After processing a data byte, the routine returns control to ISBSC$COMM:IRQ which checks for other pending interrupts.

Transmitter Ready (IRQ) Routine (TRANSMIT)

This routine responds to 2651 transmitter ready IRQ's. It has 5 basic functions:

1) Move data from the Transmit Buffer to the 2651
2) Decode data and modem control signals (see Section 6.13.2.1)
3) Insert between-block syn and idle characters
4) Process modem signals received from the network.
5) Verify data accuracy by means of the 2653

After processing a data byte, control is returned to ISBSC$COMM:IRQ which checks for other pending interrupts.

Note: Modem signal changes are processed by setting the appropriate bits in the Auxiliary Control Signal Register and the 2651 Command Register. The 2651 Command Register is detailed in the Signetics 2651 PCI document. Information concerning the ACSR structure and use is found in the Hardware System Spec. It is useful to note that the only bits of the ACSR which are utilized in the IS/BSC (other than IRQ) are BUSY (ACS_IN), CTS and RNG (ACS_OUT). CTS is used only when Clear to Send Delay is activated.

Data Set Change (IRQ) Routine (ISBSC$COMM:DSC)

This routine is responsible for handling Data Set (Modem) changes. It is initiated by the occurrence of a data set change interrupt from the 2651 or ACSR. The data set signals that can cause this interrupt (at the local 2651 or ACSR) are:

1) DCD - data carrier detect
2) DSR - data set ready
3) RI - ring in

When this routine is entered, the 2651's Status register and the ACSR are read. The above mentioned signals are formed into a Modem Cntl. Signal byte and the byte is encoded as described in Section 6.13.2.1. The encoded data is placed in the Receive Holding Buffer and the routine exits. If Clear-to-Send Delay is activated, the routine to generate this signal is delay-forked.

6.13.2.1 Data Encoding/Decoding

Data encoding is performed as follows:

```
         Data (D) ==>            (D)| if 0 < (D) < X'FF'
                         (X'FF',X'81')| if (D) = 0
                         (X'FF',X'FF')| if (D) = X'FF'

   Modem Signals(S) ==>      (X'FF',S)| if (S) > 0
                         (X'FF',X'80')| if (S) = 0

       Call signals ==>  (X'FF',X'82')| Request Call Termination
```

Note: S≤X'0F' (i.e., high order nibble must be zero)

Data decoding is the inverse of the encoding procedure, except that Call Signals are not decoded. The Call Signals described above are special signals used by the CMM Interface to terminate a call.

6.13.3 Network Spoofing Control (Submodule ISBSC$SPOC)

This submodule controls the movement of data between the Transmit Holding Buffer and the Transmit Buffer, and the generation of all spoofed messages. Additionally, it handles the discarding of previous outbound blocks and enabling of the 2651 transmitter.

The submodule is composed of 3 routines:

ISBSC$SPOC:IMMED
ISBSC$SPOC:DELAYED
ISBSC$SPOC:NEGATIVE

Entry Point

ISBSC$SPOC:IMMED

Function

Generate positive spoofed responses (ACK's) or transfer actual data to the transmit buffer.

Entry Conditions

* Forked

Exit Conditions

* Terminates

Entry Point

ISBSC$SPOC:DELAYED

Function

Generate WACK's, TTD's or ENQ's, depending on the value of OF$IP$TCB:XSAVE3, and spoof EOT when TTD or ENQ limits are exceeded.

Entry Conditions

* Forked
* OF$IP$TCB:XSAVE3 = flag indicating desired message

Exit Conditions

* Terminates

Entry Point

ISBSC$SPOC:NEGATIVE

Function

Generate NAK or ENQ outbound messages based on the value of OF$ISBSC:PORT_STAT.

Entry Conditions

* Forked

Exit Conditions

* Terminates

6.13.4 Inbound Protocol Handling (Submodule ISBSC$IBP)

This protocol routine is forked by the Receiver Ready (IRQ) routine when data has been moved to the Receive Buffer.

As long as a call is in progress, the contents of the Receive Buffer are sent to the Adaptive Data Compression Encoder where they are encoded and stored in the Inbound Data Buffer. If no call is in progress all data, excluding Data Set Change Signals, are ignored; these may be used to initiate a call.

Entry Point

ISBSC$IBP:GETCHAR

Entry Conditions

* Forked

Exit Conditions

* Terminates

6.13.5 Outbound Protocol Handling (ISBSC$OBP)

This protocol routine is forked by the PRE-ARQ/BIC Receiver or the Spoofing Controller when data is to be moved to the Transmit Holding Buffer. When no bytes are available, it sets a flag (ARQ fork flag) indicating to the PRE-ARQ/BIC Receiver to fork this routine when new data is placed in the Outbound Data Buffer.

The submodule 1) moves bytes to the Transmit Holding Buffer, 2) inserts Internal Signals (IS's) when spoofing to cause the 2653 to do BCC checking, 3) updates the ACT count, discarding ACK blocks when spoofing, and 4) flags the Spoofing Controller module when data is ready for movement to the Transmit Buffer.

The routine terminates without setting the fork flag when a complete BSC data block is in the Transmit Holding Buffer.

Entry Point

ISBSC$OBP:ENTRY

Entry Conditions

* Forked

Exit Conditions

* Terminates

6.13.6 Call Manager Interface (ISBSC$CMI)

This submodule is responsible for handling communications between the protocol modules and the Call Manager. It has 3 entry points:

ISBSC$CMI:ENTRY
ISBSC$CMI:SEND_CRECALL
ISBSC$CMI:SEND_HANGUP

and performs 5 distinct tasks:

1)	Call End Processing	(CMM --> Protocol)
2)	Call Request Processing	(CMM --> Protocol)
3)	Call Created Processing	(CMM --> Protocol)
4)	Hangup Request	(Protocol --> CMM)
5)	Create Call Request	(Protocol --> CMM)

Entry Point - ISBSC$CMI:ENTRY

Function

Dequeue an addressed packet from the Call Manager, activate the appropriate routine to process it by identifying the command code contained in the addressed packet message field.

Entry Conditions

* None

Exit Conditions

* All registers destroyed

Call End Processing

This routine is invoked by ISBSC$CMI:ENTRY upon receipt of a call end AP from the Call Manager. It initiates reinitialization of IP common routines, and sends a create call AP to the Call Manager when a new call is indicated.

Call Request Processing

This routine is invoked by ISBSC$CMI:ENTRY when a "Call Request" AP is received from the Call Manager. It is responsible for rejecting calls when the port is already busy or there is no answer, for raising "RING" 1-5 times when the terminal DSR is down, and for sending "Call Accepted" AP's to the Call Manager.

Call Created Processing

This routine is invoked by ISBSC$CMI:ENTRY upon receipt of a "Call Created" AP from the Call Manager. The routine enables the BIC IB and OB FIFO's and the 2651 receiver so the newly active call can proceed with data transmission.

Entry Point - ISBSC$CMI:SEND_HANGUP

Function

This routine is called by the Spoofing Controller or the Call Manager Interface. Its function is to construct and route to the Call Manager an hangup addressed packet.

Entry Conditions

* None

Exit Conditions

* A-Reg destroyed
* X-Reg destroyed

Entry Point - ISBSC$CMI:SEND_CRECALL

Function

This routine is called by the Protocol modules or the Call Manager Interface. It handles the construction and routing of a create call addressed packet to the Call Manager.

Entry Conditions

* None

Exit Conditions

* A-Reg destroyed
* X-Reg destroyed

6.13.7 Statistics and Monitoring (Submodule ISBSC$STAT)

This submodule is responsible for monitoring the performance of the ISBSC, reporting error and exception conditions to the network report port, and responding to requests for port statistics. It has three external entry points:

1. ISBSC$STAT:COLLECT_STATS
2. ISBSC$STAT:STAT
3. ISBSC$STAT:MONITOR

Entry Point - ISBSC$STAT:COLLECT_STATS

Function

Collects instantaneous values for processor loading, encoder nibbles in, encoder nibbles out, number of buffers in use, number of characters received from the 2651, number of blocks received from the 2651, number of blocks received in error, and current character error count. Forks the monitoring routine (ISBSC$STAT:MONITOR). (This routine is initiated by IPOS every 6 seconds.)

Entry Conditions

* None

Exit Conditions

* All registers destroyed

Entry Point - ISBSC$STAT:MONITOR

Function

Updates the weighted-average value for each statistic using the current instantaneous value. Calculates instantaneous processor load, buffer utilization, compression efficiency, character error rate and block error rate, comparing percentages to threshold values and reporting exceptions to Network Report Port.

Entry Conditions

* None

Exit Conditions

* None

Entry Point - ISBSC$STAT:STAT

Function

Creates and sends a statistics addressed packet in response to statistics request from the CTP. Statistics included:

1. Processing Loading - As calculated by IPOS (see Section 6.1.10).

2. Buffer Utilization - The ratio of the number of buffers currently in use to the total number of buffers in the free buffer pool, times 100%.

3. Compression Efficiency - the ratio of the total number of bits (including parity bit) from the terminal to the number of bits resulting from code compression, times 100%.

4. Character Error rate - the ratio of the number of characters with bad parity received from the terminal to the total number of characters received, times 100%.

5. Statistical Loading - the ratio of the number of characters from the terminal to the maximum number of characters which the terminal could have sent in the elapsed time, times 100%.

6. Compressed Loading - The ratio of the number of bits resulting from code compression to the maximum number of bits (including parity bit) the terminal could have sent in the elapsed time, times 100%.

7. Block Error Rate - The ratio of the number of blocks received with either parity or BCC errors from the terminal to the total number of blocks received, times 100%.

Entry Conditions

* None

Exit Conditions

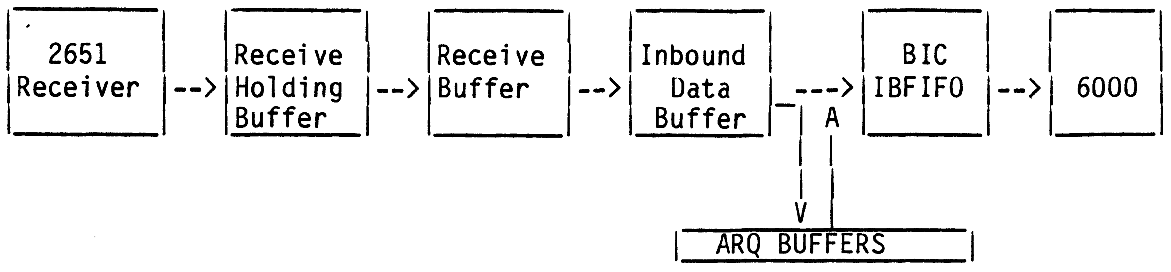
* None

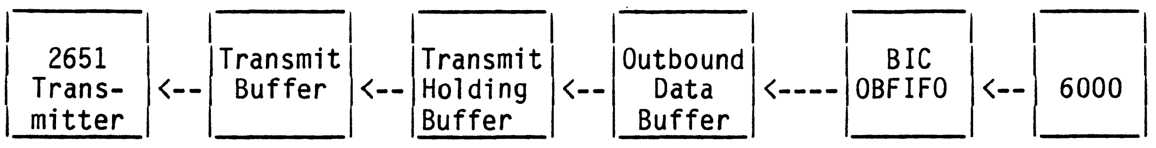
At this point it seems desirable to deviate from the module functional specifications in order to illustrate IS/BSC data flow and the interaction of IS/BSC submodules and IP Common routines.

In transmitting data from a local to a remote terminal port, a number of data and hardware structures are used to temporarily hold data. These transfers of data allow IS/BSC modules to determine the type of processing required and to perform the necessary and/or desirable protocol functions (i.e., sync filling, data compression, etc.).

The following diagrams indicate the structures used (by name only) and the direction of data flow in the IS/BSC.

INBOUND DATA MOVEMENT

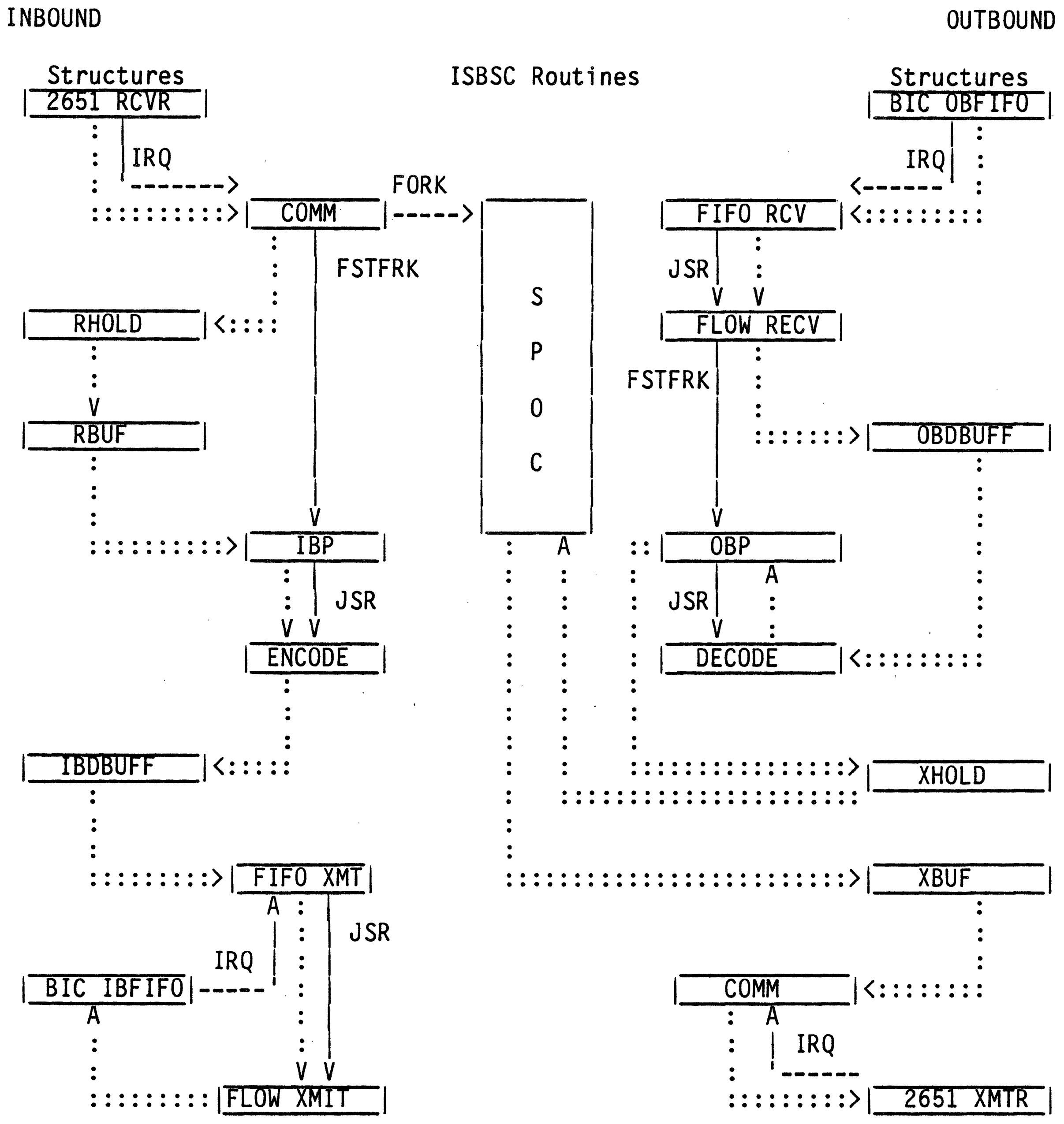




OUTBOUND DATA MOVEMENT

(Data in the Receive and Transmit Holding Buffers may never go any further, but may be discarded - e.g., inbound error blocks, outbound ACK's; during spoofing.)

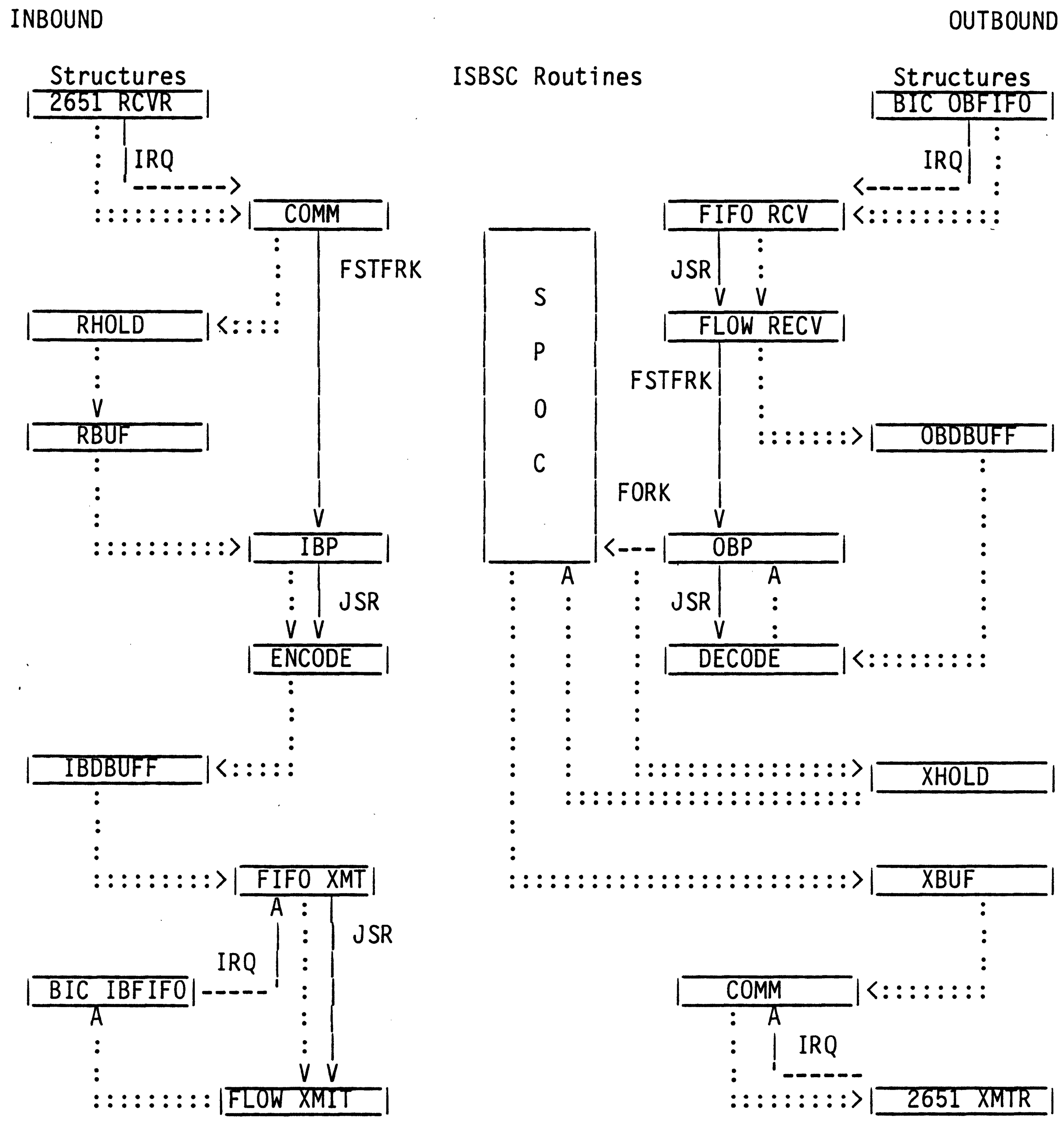
The following diagrams graphically illustrate combined logic and data flow in the ISBSC. Separate diagrams are given for spoofed and non-spoofed operation to emphasize the difference in transfer of control to SPOC. Note that even when not spoofing, the Spoofing Controller is active in data movement.



:::: ==> DATA FLOW
___ --> LOGIC FLOW

ISBSC - Spoofed Operation

Note: This diagram ignores all structures and routines associated with ARQ/FLOW control except those which directly interface with ISBSC routines.



:::: ==> DATA FLOW
___ --> LOGIC FLOW

ISBSC - Non-Spoofed Operation

Note: This diagram ignores all structures and routines associated with ARQ/FLOW control except those which directly interface with ISBSC routines.

6.14 Intelligent Spoofed Synchronous Terminal Port Protocol Software (HASP Version)

The Intelligent Spoofed Synchronous Terminal Port Protocol Module (HASP version) (I/SSTP-HASP, herein referred to as IS/HSP) is organized as a set of submodules, each of which provides a related set of functions. These functions include:

1) System Initialization
2) Communications Interrupt Handling
3) Network Spoofing Control
4) Call Manager Interface
5) Statistics and Exception Monitoring

Descriptions of the submodules which perform these functions are found throughout Sections 6.14.1 through 6.14.5.

6.14.1 System Initialization (Submodule ISHSP$INIT)

The function of this submodule is to start IP common modules (i.e., ADCM$, CMM$, FIFO$, FLOW$) and to initialize the IS/HSP data structures. The submodule is composed of three routines. The first routine is called by IPOS to begin initialization of the protocol tasks, the second is forked by the first, and the third routine is a batch task with an entry in the Module Dispatch Table that is scheduled subsequently.

The first routine initializes all protocol state variables and data structures for statistics accumulation, and obtains buffers to be used as permanent TCB's for all other IS/HSP submodules. In addition, it obtains the Spoofing Buffer and Holding Buffers used by the 2651 interrupt routines as temporary storage of data to be sent or data that has been received, and forks the second routine (ISHSP$INIT:REQ) before returning to IPOS.

ISHSP$INIT:REQ builds a configuration-request addressed packet and routes it to the mainframe MCM$CMEM module (see Section 5.7). The return of an addressed packet containing the requested information causes scheduling of the third routine (ISHSP$INIT:CONF), that which initializes the remaining data structures and completes and the initialization process for the IS/HSP. Included in this initialization stage is the calculation of the maximum number of outstanding blocks permitted.

Of particular importance to IS/HSP is the correspondence of FCS bits to RCB's, information of which is received from CMEM at this time.

Entry Point - ISHSP$INIT:ENTRY

Function

Initialize IS/HSP data structures, start IP common modules, and obtain 2651 Holding Buffers and Spoofing Buffer.

Entry Conditions

* None

Exit Conditions

* All registers destroyed

Entry Point - ISHSP$INIT:REQ

Function

Build a configuration-request packet

Entry Conditions

* None

Exit Conditions

* None

Entry Point - ISHSP$INIT:CONF

Function

Complete IS/HSP data structure initialization based on the information contained in returned configuration-request packet.

Entry Conditions

* None

Exit Conditions

* None

6.14.2 Communications Interrupt Handling (Submodule ISHSP$COMM)

This submodule is responsible for handling interrupts caused by the 2651 communications circuit. Interrupt types include:

1) Receiver Ready
2) Transmitter Ready
3) Data Set (MODEM) Change

Whenever a 2651 interrupt is received, it is routed to a routine (ISHSP$COMM:IRQ) which determines what type of interrupt has occurred and dispatches it to the appropriate interrupt processor.

Entry Point - ISHSP$COMM:IRQ

Function

Dispatch 2651 IRQ's to their corresponding interrupt routines.

Receiver Ready (IRQ) Routine (ISHSP$COMM:RCV)

This routine is utilized to handle the 2651's receiver section, being initiated by the occurrence of a receiver ready interrupt. It performs 3 functions:

1) Moves data from the 2651 to the Receive Holding Buffer.
2) Performs data encoding (as detailed in Section 6.14.2.1).
3) Detects and records line errors.

The routine only relinquishes control when a data set change occurs or when no more data is available.

Transmitter Ready (IRQ) Routine (ISHSP$COMM:XMT)

This routine is utilized to handle the 2651's transmitter section, being initiated by the occurrence of a transmitter ready interrupt. It has 4 basic functions to perform:

1) Move data from the Transmit Holding Buffer to the 2651.
2) Decode data and modem control signals (as detailed in Section 6.14.2.1).
3) Line fill (either syn or pad characters) when unable to transmit data.
4) Process modem signal changes received from the network.

The routine only relinquishes control when a data set change occurs or when no data remains in the Transmit Holding Buffer.

Note: Modem signal changes are processed by setting the appropriate bits in the Auxiliary Control Signal Register and the 2651 Command Register. The 2651 Command Register is detailed in the Signetics 2651 PCI document. Information concerning the ACSR structure and use is found in the Hardware System Specification. It is useful to note that the only bits of the ACSR which are utilized in the IS/HSP (other than IRQ) are BUSY (ACS_IN), CTS and RNG (ACS_OUT). CTS is used only when Clear to Send Delay is activated.

Data Set Change (IRQ) Routine (ISHSP$COMM:DSC)

This routine is responsible for handling Data Set (Modem) changes. It is initiated by the occurrence of a data set change interrupt from the 2651 or ACSR. The data set signals that can cause this interrupt (at the local 2651 or ACSR) are:

1) DCD - data carrier detect
2) DSR - data set ready
3) RI - ring in

When this routine is entered, the 2651's Status Register and the ACSR are read. The above mentioned signals are formed into a Modem Cntl. Signal byte and the byte is encoded as described in Section 6.14.2.1. The encoded data is placed in the Receive Holding Buffer and the routine exits. However, if the buffer is full, the routine retries until successful.

6.14.2.1 Data Encoding/Decoding

Data encoding is performed as follows:

```
          Data (D) ==>            (D)| if 0 < (D) < X'FF'
                       (X'FF',X'81')| if (D) = 0
                       (X'FF',X'FF')| if (D) = X'FF'

  Modem Signals(S) ==>       (X'FF',S)| if (S) > 0
                       (X'FF',X'80')| if (S) = 0

      Call signals ==> (X'FF',X'82')| Request Call Termination
                       (X'FF',X'83')| Call Termination Granted
```

Note: $S \leq X'7F'$ (i.e., high order bit must be zero)

Data decoding is the inverse of the encoding procedure, except that Call Signals are not decoded. The Call Signals described above are special signals used by the CMM Interface to terminate a call.

6.14.3 Network Spoofing Control (Submodule ISHSP$SPOC)

This submodule is responsible for handling the protocol operations required in a HASP spoofing environment as detailed in the D814 Product Functional Specification, Appendix F. In short, the Spoofing Controller regulates the flow of data so that communications appear to proceed as in a non-spoofed HASP network, sometimes by generating responses like those of the remote CPU (i.e., ACKO, WAB) in the absence of real outbound data.

The controller is physically divided into Inbound and Outbound Protocol Processors which are described in the following subsections. Familiarity with multileaved data block structure is assumed throughout.

Note: The structure of a typical HASP transmission block can be found in Appendix F of the D814 FunctionaL Specification.

Protocol In Processor

This protocol routine is forked by the Receiver Ready (IRQ) routine when data has been placed in the Receive Holding Buffer. The functions performed by this routine include:

1) BCB verification

2) FCS monitoring and reflective updating of metering indicators

3) BCC calculation and verification

4) Appending of "abort" ICS's to incorrectly received data blocks

5) Appending of "in response to data" flags to data blocks (when applicable)

6) Timing for, and generation of, ACKO's and WAB's in the absence of outbound data to maintain synchronization

7) Updating of outstanding block count

If a call is in progress, the data in the Receive Holding Buffer is given to the Adaptive Data Compression Encoder after appropriate processing; otherwise, all data is ignored until a call is established. In either case, recognition of changes in the terminal's DSR signal are transmitted to the Call Manager for processing.

Entry Point - ISHSP$SPOC:IBP

Entry Conditions

* None

Exit Conditions

* None

Protocol Out Processor

This protocol routine is forked by the Transmitter Ready (IRQ) routine, the pre-ARQ/BIC Receiver, or the Call Manager when data is to be moved to the Transmit Holding Buffer. Its responsibilities include:

1) Comparison of RCB's to metering indicators, performing metering off when appropriate

2) Calculation of new BCC's for data blocks which have had records metered off

3) Resetting of BCB's when metering on

4) Formulation of metered data into normal transmission blocks

5) Retransmittal of NAKed data blocks

6) Error recovery

7) Updating of outstanding block count

8) Regulation of data block transmittal (i.e., one response received ===> one block transmitted)

9) Generation of remote modem control signals (in particular RTS and CTS).

The routine calls the Adaptive Data Compression Decoder in order to obtain data bytes from the OB Data Buffer. When a "request call termination" or "call terminate" signal is received (see Section 6.14.2.1), the signal is passed to the Call Manager for processing and is not placed in the Holding Buffer. When no bytes are available, it sets a flag (ARQ Fork flag) indicating to the Pre-ARQ/BIC Receiver to fork this routine when data becomes available, and it resets a flag (Transmitter Fork flag) indicating to the Transmitter Ready (IRQ) routine not to fork this routine when the Xmit holding buffer falls below half full.

As bytes are transferred to the Transmit Holding Buffer, if the communications transmitter is not running it is started by enabling the 2651 transmitter. This causes a transmitter ready IRQ, which causes the Transmitter Ready (IRQ) routine to start sending the data contained in the Holding Buffer. When the Transmit Buffer fills, the routine sets the Transmitter Fork flag and terminates.

Entry Point - ISHSP$SPOC:OBP

Entry Conditions

* None

Exit Conditions

* None

6.14.4 Call Manager Interface (ISHSP$CMI)

This submodule is responsible for handling communications between the Protocol modules and the Call Manager. It has one external entry point (ISHSP$CMI:ENTRY), and performs 5 distinct tasks:

1)	Call End Processing	(CMM --> Protocol)
2)	Call Request Processing	(CMM --> Protocol)
3)	Call Created Processing	(CMM --> Protocol)
4)	Hangup Request	(Protocol --> CMM)
5)	Create Call Request	(Protocol --> CMM)

The routines to process the first three are activated by ISHSP$CMI:ENTRY upon receipt of an addressed packet from the Call Manager. The latter two are subroutines used by the Spoofing Controller and the other CMI routines, and are invoked by a JSR.

Entry Point - ISHSP$CMI:ENTRY

Function

Dequeue an addressed packet from the Call Manager, activate the appropriate routine to process it by identifying the command code contained in the addressed packet message field.

Entry Conditions

* None

Exit Conditions

* All registers destroyed

Call End Processing (ISHSP$CMI:CALLEND)

This routine is invoked by receipt of a call end AP from the Call Manager. It initiates reinitialization of IP common routines, and sends a create call AP to the Call Manager when a new call is indicated.

Call Request Processing (ISHSP$CMI:CALLCRE)

This routine is invoked by the receipt of a call request AP from the Call Manager. It is responsible for setting an error flag when the port is already busy or when the call has been improperly routed, and sends a call accepted AP to Call Manager.

Call Created Processing (ISHSP$CMI:CALLCRE)

This routine is invoked by the receipt of a call created addressed packet from the Call Manager. It enables the 2651 receiver and BIC Inbound and Outbound FIFO's so a call can be established.

Hangup Request (ISHSP$CMI:SEND_HANGUP)

This routine is called by the Spoofing Controller or the Call Manager Interface. Its function is to construct and route to the Call Manager an hangup address packet.

Create Call Request (ISHSP$CMI:SEND_CRECALL)

This routine is called by the Protocol modules or the Call Manager Interface. It handles the construction and routing of a create call addressed packet to the Call Manager.

==

At this point it seems desirable to take an aside from module functional specifications in order to illustrate IS/HSP data flow and the interaction of IS/HSP submodules and IP Common routines.

In transmitting data from a local to a remote terminal port, a number of data and hardware structures are used to temporarily hold data. These transfers of data allow IS/HSP modules to determine the type of processing required and to perform the necessary and/or desirable protocol functions (i.e., sync filling, data compression, etc.).

The following diagrams indicate the structures used (by name only) and the direction of data flow in the IS/HSP.

INBOUND DATA MOVEMENT

```
 __________      ___________      _________      _________      ______
|          |    |           |    |   IB    |    |         |    |      |
|   2651   | -->| OF$ISHSP: | -->|  DATA   | -->|   BIC   | -->| 6000 |
| RECEIVER |    |   RRING   |    | BUFFER  |    | IBFIFO  |    |      |
|__________|    |___________|    |_________|    |_________|    |______|
```

OUTBOUND DATA MOVEMENT

```
 _____________       ___________          ________       _________       ______
|             |     | OF$ISHSP: |        |        |     |         |     |      |
|    2651     | <-- |___XRING___|  <--   |   OB   | <-- |   BIC   | <-- | 6000 |
| TRANSMITTER |           A              |  DATA  |     | OBFIFO  |     |      |
|_____________|           |        <--   | BUFFER |     |_________|     |______|
                     _____|_____         |________|
                    | SPOOFING  |
                    | BUFFER    |
                    |___________|
```

In this schemata, data is concurrently moved from the OB Data Buffer into the Spoofing Buffer and XRING. The Spoofing Buffer saves a copy of an entire block transmitted; the OB Data Buffer is ready to accept a new block. Data transfer from the Spoofing Buffer to XRING occurs only when a NAK response is encountered, indicating that the block previously transmitted was not correctly received, hence requires retransmittal.

(In pursuit of optimization, direct data transfer between the OB Data Buffer and XRING may be eliminated. All data to XRING may be required to first pass through the Spoofing Buffer.)

In order to gain a full understanding of how the IS/HSP submodules interact to accomplish their task, it is helpful to study the diagrams in Section 6.12 which briefly trace the path of data and logic through the I/STP. The only difference in program control flow for the IS/HSP is that SPOC replaces both IBP and OBP of I/STP.

6.14.5 Statistics and Monitoring (ISHSP$STAT)

This routine is responsible for monitoring the performance of the IS/HSP and reporting the information as a system alarm to the network Report Port. In addition, any system module can request current information from this module.

The individual modules in the IS/HSP are responsible for updating monitoring information as follows:

1) Receiver Ready Interrupt Routine

   a) Number of characters received
   b) Number of errors that occurred
   c) Number of characters after encoding

2) Adaptive Data Compression Encoder

   a) Number of nibbles encoded
   b) Number of resulting nibbles

The statistics routine is started by IP initialization and runs every 6 seconds. Each time it executes, it calculates the following statistics:

1) Compression Efficiency - The ratio of the total number of bits (including parity bit) from the terminal to the number of bits resulting from code compression times 100%.

2) Character Error Rate - The ratio of the number of characters with bad parity received from the terminal to the total number of characters received, times 100%.

3) Buffer Utilization - The ratio of the number of buffers currently in use to the total number of buffers in the free buffer pool, times 100%.

4) Processor Loading - As calculated by IPOS (see Section 6.1.10).

The results of these calculations are compared to their respective thresholds, which are obtained from the mainframe during initialization. If any threshold has been exceeded, an alarm packet is constructed and sent to the network Report Port.

If a packet is received from a system module requesting statistics, an addressed packet containing the above information, as well as:

1) Statistic Loading - The ratio of the number of characters from the terminal to the maximum number of characters which the terminal could have sent in the elapsed time, times 100%.

2) Compressed Loading - The ratio of the number of bits resulting from code compression to the maximum number of bits (including parity bit) the terminal could have sent in the elapsed time, times 100%.

3) Memory Utilization - The ratio of the size of the port software (including page zero and system areas) to the total memory size of the physical port on which the software is running, times 100%.

is constructed and returned to the requesting module.

Entry Point - ISHSP$STAT:MONITOR

Function

Calculate statistics and send exception report addressed packets to the network Report Port.

Entry Conditions:

* None

Exit Conditions:

* None

Entry Point - ISHSP$STAT:AP

Function

Calculate statistics and send a statistics addressed packet to the requesting module.

Entry Conditions:

* None

Exit Conditions:

* None

6.15 Intelligent Bit-Oriented-Protocol Terminal Port (I/BOP) Protocol Module

6.15.1 Introduction

An I/BOP is an intelligent terminal port designed to interface HDLC (High Level Data Link Control) type protocols so that two DTE's supporting such protocols can communicate through the D814 network.

The I/BOP requires a communications section based on a Motorola MC6854 ADLC chip and ACS (Auxiliary Control Signal) register.

Running under the 6800 IP Operating System (IPOS, Section 6.1), the I/BOP-specific module that is described here is divided into the following submodules:

1) Initialization
2) Communications Control Processing
3) Protocol Handling
4) Call Manager Interfacing
5) Statistics and Monitoring

6.15.2 Functional Submodule Description

In this section, the functions of an I/BOP are described as belonging to 5 submodules as follows.

6.15.2.1 Initialization Submodule (IBOP$INIT:)

The function of this submodule is to make the I/BOP ready for a user.

Called by IPOS during the port initialization time, IBOP$INIT:START initializes common single-threaded I/TP modules, and allocates storage blocks for I/BOP data structures and initializes them. Finally, it forks IBOP$INIT:SETUP and terminates. IBOP$INIT:SETUP then, started by the IPOS scheduler, sends an Addressed Packet to MCM$CMEM of the Mainframe Configuration Control Module (see Section 5.7) to request for configuration parameters which are eventually received by IBOP$INIT:CONF (module number EQ$IP$MDT:IPP_INIT) for the port configuration. When the port is configured accordingly, the port initialization is complete. Then, a call is set up if a leased line is in effect.

After a table of configuration parameters has been built by this submodule, it is used by IPCC$XMTRCV of the IP Configuration Control Module (see Section 6.2) for configuration reads or reconfigurations.

The configurable parameters for an I/BOP are specified in the Section 3.2.7 of the D814 Product Functional Specification.

6.15.2.2 Communications Control Submodule (IBOP$COMM:)

This submodule has the complete control over the communications section of the I/BOP. It processes interrupts from the local communications section, and detects Circuit Control Signals (CCS's) received from the remote port to reflect the signaled conditions at the local communications section.

In order to exchange virtual circuit control information with the remote port, an internal encoding/decoding scheme is used as follows.

```
           Data  (D) ==>            (D) | if 0 < (D) < X'FF'
                         (X'FF', X'81') | if (D) = 0
                         (X'FF', X'FF') | if (D) = X'FF'

           Frame CCS ==> (X'FF', X'84') | good frame (good FCS)
                         (X'FF', X'85') | bad frame (bad FCS)
                         (X'FF', X'86') | aborted frame
                         (X'FF', X'87') | mark idle

       Modem CCS (S) ==>     (X'FF', S) | if (S) > 0
                         (X'FF', X'80') | if (S) = 0

Call Termination CCS ==> (X'FF', X'83') | call termination
```

Note: S≤X'7F' (i.e., high order bit must be zero)

All the interrupts are entered at IBOP$COMM:IRQ by the IPOS.

Entry Point - IBOP$COMM:IRQ

After determining the cause of the interrupt, this routine calls one of the following three routines whose functions are described in the subsequent subsections.

1) Transmitter Controller

2) Receiver Controller

3) Modem Signal Change Handler

1) Transmitter Controller (IBOP$COMM:XMT)

This routine is called by IBOP$COMM:IRQ when the interrupt was originated from the 6854 transmitter.

Data bytes are retrieved from a transmit frame byte queue (see Section 7.1 for byte queue description) only if it contains a complete frame. Therefore, a transmitter underrun is prevented. However, it will result in a delay of one frame.

If the data from a transmit frame byte queue is user data, it is transmitted and then the routine returns. If it is a frame CCS, the MC6854 is instructed to send either a closing or abort flag as appropriate. If it is a modem CCS, the appropriate change is made to the communications section. After the control function has been performed, the transmit frame byte queue is deleted and the routine goes back to its beginning to process another data from the next transmit frame byte queue. If there is no such a transmit frame byte queue completely assembled yet, the 6854 transmitter is disabled.

The modem signal changes made by this routine are the opposite (RS 232-C protocol wise) to those sensed at the remote port. The following relationship exists between the signals sensed at the remote port and those changed at the local port:

Data Set Ready (DSR)	<---->	Data Terminal Ready (DTR)
Data Carrier Detect (DCD)	<---->	Request to Send (RTS)
Make Busy (MB)	<---->	Ring In (RI)

If CTS went down, the 6854 transmitter is reset and the active transmit frame byte queue is reset so that the frame can be retransmitted when CTS comes up.

2) Receiver Controller (IBOP$COMM:RCV)

Called by IBOP$COMM:IRQ when the interrupt was originated from the receiver, this routine first checks the cause of the interrupt.

If it is a Receiver Data Available interrupt, a byte from the 6854 receiver is passed to IP$FLOW$PXMT routine. If the interrupt is due to an end of a frame, an appropriate frame CCS is passed.

If a mark idle condition is detected in the input stream, it is translated into the corresponding frame CCS to reflect this condition at the remote port.

3) Modem Signal Change Handler (IBOP$COMM:MODEM)

There are four modem signals whose changes cause the transfer of control to this routine:

1. Data Set Ready (DSR)
2. Data Carrier Detect (DCD)
3. Make Busy (MB)
4. Clear To Send (CTS)

If the interrupt is due to any changes in the first three signals, the Protocol Inbound routine (IBOP$PROT:INB) is called.

If CTS came up, the 6854 transmitter is enabled as long as there is a complete transmit frame byte queue to transmit.

If DCD came up, the SPARE_CTS is raised after configured CTS-delay time period. If DCD went down, it is dropped.

6.15.2.3 Protocol Handler Submodule (IBOP$PROT:)

This submodule is responsible for moving data between the Communications Control Submodule and the ARQ/Flow control (IP$FLOW$) Module. Additionally, it calls the Call Manager Interface Submodule in order to establish or terminate a call.

The actions performed by this submodule depend on the call state which may be one of the following:

1. idle: There is no call being established, terminated, or active.
2. calling: A call is being established.
3. active: A call has been established.
4. terminating: A call is being terminated.

There are two protocol handling routines:

1) Outbound Protocol Handler
2) Inbound Protocol Handler

The functions of these routines are described below.

1) Outbound Protocol Handler (IBOP$PROT:OUTB)

This routine is called by the IP$FLOW$RECV when data is available to be moved to the transmitter.

Entry Point - IBOP$PROT:OUTB

If the data received by this routine is user data, it is put into the last transmit frame byte queue.

If it is an abort frame CCS, it is ut into the last transmit frame byte queue. However, if it is a bad frame CCS, the routine processes it depending on the configurable bad frame option as follows. If the option is set for abort, a frame CCS for an abort is put into the last transmit frame byte queue instead. But, if the option is set for discard, the entire transmit frame byte queue is discarded. When the routine finishes building a transmit frame byte queue by appending a frame CCS (none for a good frame CCS), it enables the 6854 transmitter. Then it creates a new transmit frame byte queue to be ready for subsequent data.

When the routine receives a 'call termination' CCS, the ensuing action depends on the call state. If it is 'idle', the CCS is ignored. If it is 'active', the call state is changed to 'terminating', the call light is made blinking, and the same 'call termination' CCS is passed back to IP$FLOW$PXMT. If the call state is 'calling' or 'terminating', the routine calls IBOP$CMI:HANGUP to clear the call.

2) Inbound Protocol Handler (IBOP$PROT:INB)

This routine is called by IBO$COMM:MODEM when a modem signal has been changed.

If DSR came up and the auto-dial call option is in effect, the routine calls IBOP$CMI:CRECALL in the Call Manager Interface Submodule. If DSR went down and the call state is 'active' under either auto-dial or contention call option, the routine changes the call state to 'terminating' and passes a 'call termination' CCS to IP$FLOW$PXMT. If the call state is 'calling' when the 'DSR down' is detected, the call state is just changed to 'terminating'.

If the call state is 'active', the modem signal is encoded into a modem CCS and passed to the remote port by calling IP$FLOW$PXMT.

6.15.2.4 Call Manager Interface Submodule (IBOP$CMI:)

Being called by the Protocol Handling Submodule when a call is to be established or terminated, this submodule interfaces with the IP Call Manager Module (CMM, see Section 6.3) to communicate with a remote port regarding call maintenance procedures. Being called by the Protocol Handling Submodule when a call is to be established or terminated.

The routines in this submodule belong to one of the two categories: the first is for sending Addressed Packets to CMM, and the second is for handling Addressed Packets received from CMM.

1) Call Addressed Packet Sender

   There are two routines which send Addressed Packets to CMM: IBOP$CMI:CRECALL for changing the call state to 'calling', making the call light blinking, and sending a 'create call' Addressed Packet; and IBOP$CMI:HANGUP for making the call state 'terminating' and sending a 'hang up' Addressed Packet. They are called when a call is to be established and terminated respectively.

2) Call Addressed Packet Receiver

   The routine IBOP$CMI:AP_RCV (module number EQ$IP$MDT:IPP_CMM), running as a batch task, retrieves a call Addressed Packet from its input job queue and calls one of the following three routines depending on the type of the Addressed Packet. There are three types of call Addressed Packets acknowledged by this submodule.

   a. Received a 'call requested' Addressed Packet: (IBOP$CMI:CALLREQ)

      The incoming call is accepted if the call state is 'idle' under the contention call option or if the call state is 'idle' or 'calling' under the auto-dial or leased line call option and the calling node/port is correct. For a contention or auto-dial call to be accepted, DSR must be up. Otherwise, the user is notified by raising RI modem signal 5 times.

      If the call is accepted, the call state is made 'calling', the call light is made blinking, and a 'call accepted' Addressed Packet is sent back to CMM. If the call is rejected, the 'call accepted' Addressed Packet is sent back to CMM with an error code.

b. Received a 'call created' Addressed Packet: (IBO$CMI:CALLCRE)

If the call state is 'calling', the routine sets it to 'active', reinitializes the transmit frame data structures, passes a local modem CCS to the remote port, enables the 6854 receiver and BIC FIFO's, and changes the call light from blinking to on. If the call state is 'terminating', it sends a 'hang up' Addressed Packet to CMM.

c. Received a 'call ended' Addressed Packet: (IBOP$CMI:CALLEND)

This routine first checks the error code in the 'call ended' Addressed Packet. If it indicates a busy condition at the remote port, the routine tries again to establish a call.

If there is no error in the Addressed Packet, it checks the call state. If it is 'idle', the routine does nothing. If it is 'calling', it tries again to establish the call. Otherwise, the routine changes the call state to 'idle', reinitializes the port, and turns off the call light.

Then, it tries to establish a call again if the call option is either leased-line or auto-dial with DSR up.

6.15.2.5 Statistics and Monitoring Submodule (IBOP$SM:)

This submodule is responsible for monitoring the performance of the I/BOP and reporting it to the network report port as any exception condition arises. It works with the Statistics and Monitoring Module (IP$SM$) by providing I/BOP-specific monitoring and statistics-gathering functions.

Toward that end, it is required that the Adaptive Data Compression Encoder (IP$ADCM:ENCODE, see Section 6.4) calculates the following.

1. Number of bytes encoded
2. Number of resulting nibbles

Also, the following statistics-gathering operations are imbedded in the Communications Control Submodule within the I/BOP Protocol Module.

1. In the Receiver Controller (IBOP$COMM:RCV)

   a) Number of frames received
   b) Number of frames received with bad FCS

2. In the Transmitter Controller (IBOP$COMM:XMT)

   a) Number of frames transmitted

When called by IP$SM$UPDATE:TASK, IBOP$SM:MON computes the following based on the last monitoring period.

1. Compression Efficiency - The ratio of the number of bits received to the number of bits resulting from adaptive data compression for a specific time period, multiplied by 100%.

2. Receiver Frame Error Rate - The ratio of the number of frames received with bad FCS to the total number of frames received for a specific time period, multiplied by 100%.

The results of the above calculations are compared to their respective thresholds obtained as configuration parameters during the port initialization. If any threshold has been exceeded, a monitoring report Addressed Packet is constructed and sent to the network report port.

When called by IP$SM$STAT:TASK, IBOP$SM$STAT appends the above statistics as well as the following in the current statistics report addressed packet.

1) Statistical Loading - The ratio of the number of bits received to the maximum number of bits which could have been received for a specific time period, multiplied by 100%.

2) Compressed Loading - The ratio of the number of bits resulting from adaptive data compression to the maximum number of bits which could have been received for a specific time period, multiplied by 100%.

3) Transmit Frame Rate - The rate of frames being transmitted measured for a specific time period, in frames/sec.

4) Receive Frame Rate - The rate of frames being received measured for a specific time period, in frames/sec.

6.15.3 Data Flow and Program Control Flow

Finally, a pictorial overview of the data flow and program control flow in an I/BOP is presented below.

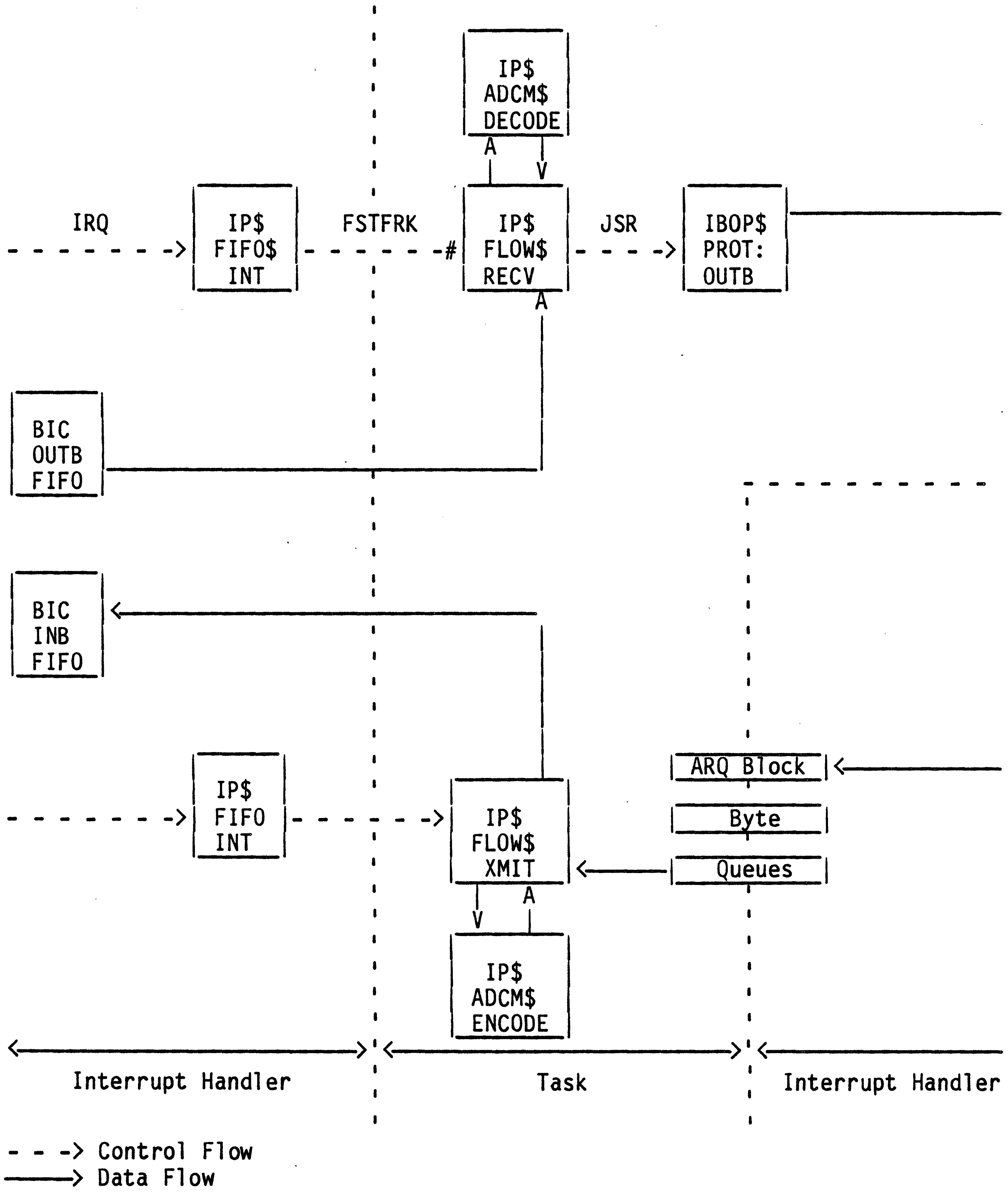
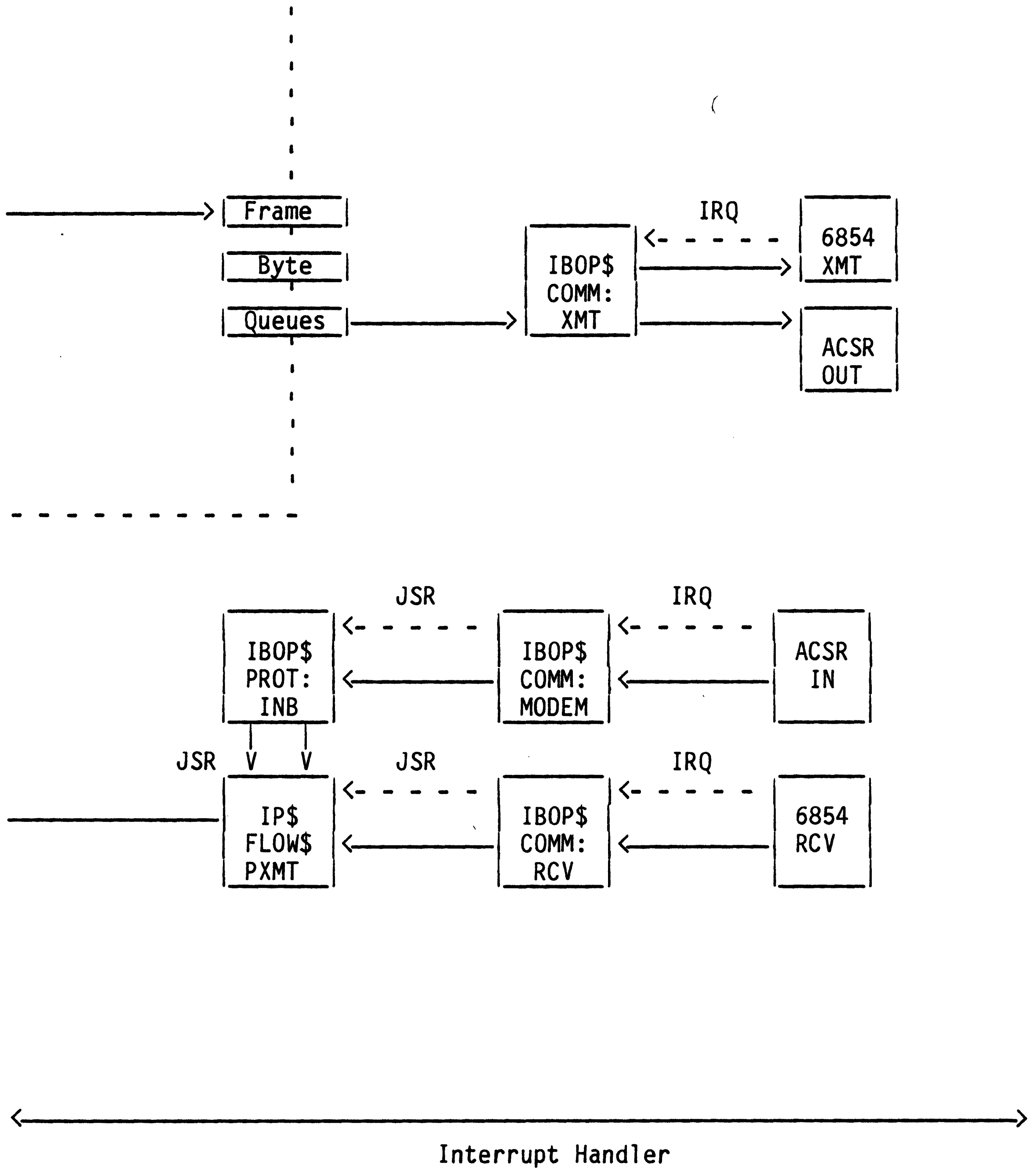


Figure 6.15-A

I/BOP SW Structure



- - -> Control Flow
———> Data Flow

Figure 6.15-B

I/BOP SW Structure

6.16 MODULE ITP

6.16.1 Overview and Definition of Terms

An explanation of the concepts of 'thread' and 'virtual port' is necessary for an understanding of the function of module ITP$.

A thread is an interface seen by the user as a distinct terminal capable of calling other terminals. ITP's may be single-threaded or multi-threaded. A single-threaded port, at its simplest, provides one RS232 connector for a user terminal while a multi-threaded port might provide more than one such connector. Each thread establishes and disconnects calls independently of other threads in the port. Threads are numbered within a port roughly sequentially from 0 to 63. Single-threaded ports are considered to have one thread whose number is 0 (0 is an invalid thread number for a multi-threaded port) and non-terminal ports are considered to have no threads.

Each thread within a multi-threaded port is considered to be a Virtual Port (VP) and has a VP ID from 2 to 255 by which it may be addressed from other ports in the network. VP ID's are, therefore, unique within the node while thread numbers are only unique within a particular port.

Module ITP$ is responsible for translating beween the VP ID's used outside of the port and the thread numbers used within the port software and for providing data areas associated with each of the threads in the port.

6.16.2 Data Structures

ITPS's most important function is the support of these thread-related data structures:

1. VP Directory: This structure is allocated for all multi-threaded ports. It is a 256-byte area of contiguous memory aligned on a page boundary. The n'th byte of the VP directory contains the thread number associated with VP number n. If there is no thread in this port associated with that VP number, then the n'th byte is 0. OF$IP$COMM:VP_DIRECTORY is a one-byte pointer to the page on which this structure resides. If the VP directory is not allocated (if, in other words, this is not a multi-threaded port), then OF$IP$COMM:VP_DIRECTORY contains a 0.

2. Thread Directory: This structure is allocated for all multi-threaded ports. It resides in contiguous memory and begins on a page boundary. Byte n of the Thread Directory contains the VP number associated with thread number n, or 0 if there is no VP for that type of number. The number of bytes in the Thread Directory is equal to the maximum configured thread number plus 1. OF$IP$COMM:THREAD_DIRECTORY is a one-byte pointer to the page on which the Thread Directory resides. If this is not a multi-threaded port, this pointer is cleared.

3. Thread Structures: These are areas of contiguous memory, one for each thread in the port. Each thread structure contains as its first byte the VP number for the thread. In addition, the Thread Structure has the following data areas:

   Call Manager Area - This area contains fields used by the Call Manager and prefixed by OF$IP$THREAD:CMM_. Only the Call Manager may write to these fields.

   Protocol Area - This area contains fields written by the I/P protocol modules and prefixed by OF$IP$THREAD:IPP_.

4. Thread Structure Index: This is an index to the Thread Structures discussed above. The size of the Thread Structure index is 2M + 2 where M is the highest thread number configured for the port, stored in OF$IP$COMM:MAX_THREAD. The n'th two-byte entry in the index is the address of the thread structure for thread number n. Location OF$IP$COMM:THREAD_INDEX points to the thread structure index. The thread structures and the thread structure index are allocated for all I/TP's.

6.16.3 ITP$ Entry Points

Module ITP$ is composed of the following submodules --

MTINIT -- Initialization code for multi-threaded ports.

STINIT -- Initialization code for single-threaded ports.

THREAD -- Utility for thread structures. Common to all I/TP's.

TRANSLATE -- VP ID to thread number translation utility. Common to all ports.

The external entry points into these submodules are listed below:

Subroutine ITP$MTINIT:USER

Function --

This subroutine is called from the user initialization code before any dynamic buffers have been deleted. It allocates a 2K contiguous area of memory which should be big enough to contain all the ITP$ data structures.

Calling Sequence --

Called with the thread structure size in X register. All registers and user-alterable TCB fields are destroyed on return to the caller.

Subroutine ITP$MTINIT:CMEM

Function --

Called from user initialization code in multi-threaded ports to complete initialization of ITP$ data structures. This routine sends an inquiry packet to the Mainframe Configuration Module requesting a list of VP's configured for the port. It then waits for ITP$MTINIT:AP to process the packet before returning to the caller.

Calling Sequence --

No calling arguments. All registers and user-alterable TCB fields are destroyed. On return, all ITP$ data structures are initialized and may be used by external modules. Also, the I/P port address is known and all thread structures have been cleared.

Routine ITP$MTINIT:AP

Function --

Completes the initialization begun by ITP$MTINIT:INIT. It is an addressed packet task activated by the response from MCM to the inquiry packet sent by ITP$MTINIT:USER.

Subroutine ITP$STINIT:USER

Function --

Called from user initialization in single-threaded ports to take the place of ITP$MTINIT:USER. It completes all ITP$ initialization. There is no MCM inquiry packet sent by ITP$ in single-threaded ports.

Calling Sequence --

No calling arguments. All registers and user-alterable TCB fields are destroyed on return to the caller.

Subroutine ITP$THREAD:LOCATE

Function --

Provides the user with the address of the thread structure for any desired thread.

Calling Sequence --

Entry -- Calling A register contains the thread number.

Exit -- On exit, X register contains the address of the thread structure (if any) for the thread in the calling A register. An invalid thread number is signaled by setting the CC:Z bit and returning 0 in the X register.

Subroutine ITP$TRANSLATE:VP_TO_THREAD

Function --

Looks up a VP number in the VP directory (if it is allocated) and returns the corresponding thread number, if there is a non-zero thread number in the VP directory.

Calling Sequence --

Entry -- Calling A register must contain the VP number. ITP initialization must be complete before calling this routine.

Exit -- On exit, A register contains the thread number for the calling A register, if it was valid. CC:Z is set if and only if there is no valid thread for the VP number passed in the A register. It should be noted that this subroutine never returns a thread number of 0 since the only way a thread 0 may exist in a port is if the port is single threaded. If that is the case, the thread is not associated with any VP.

6.17 Intelligent Multiple Asynchronous Terminal Port Protocol Software

The Intelligent Multiple Asynchronous Terminal Port Protocol Module (IMATP) passes data from 1 - 16 asynchronous terminals to the D814 Network without protocol intervention. It is organized as a collection of submodules, each responsible for performing a related set of functions. These functions include:

1) System Initialization
2) Communications Interrupt Handling
3) Protocol Handling
4) Call Manager Interface
5) Statistics and Exception Monitoring

Descriptions of the submodules which perform these tasks are found in Sections 6.16.1 through 6.16.5.

The IMATP user data structures, called thread structures, are data areas for each configured terminal. These thread structures are initialized by an accessed through module ITP$. They are discussed in Section 6.16.

6.17.1 System Initialization (submodule IMATP$INIT)

This submodule is responsible for starting multithread IP Common routines (i.e., Data Compression Module, Call Manager, FIFO Interrupt Handlers, ARQ and Flow Control) and allocating and initializing I/MATP data structures. It consists of three distinct routines. The first is called by IPOS during its own initialization phase to begin initialization of the protocol tasks; the second is forked by the first; and the third routine is a batch task with an entry in the Module Dispatch Table that is scheduled subsequently.

The first routine calls routine ITP$MTINIT:INIT to allocate sufficient contiguous storage for the maximum conceivable number of thread structures. The thread structure size is specified in the call to ITP$MTINIT:INIT. It then forks the second routine (IMATP$INIT:REQ) and returns to IPOS.

IMATP$INIT:REQ first calls ITP$MTINIT:CMEM to complete the initialization of the thread structures. ITP$MTINIT:CMEM gets from Mainframe Configuration Memory a list of all threads configured for this port and initializes a thread structure for each, placing the VP address in the first byte and clearing the rest of the structure. The thread structures contain all permanent data structures which must be maintained for each configured thread. The following sorts of MATP data items are in the thread structures:

Receive and Transmit Holding Buffers used by the 2651 interrupt routines for temporary stroage of incoming and outgoing data.

Protocol state variables.

Data structures for statistics accumulation.

After calling ITP$MTINIT:CMEM, the routine routes terminal configuration request addressed packets (one for each configured thread) to Mainframe Module MCM$CMEM (see Section 5.7).

The receipt of terminal configuration parameters (via addressed packets) causes scheduling of the third routine, IMATP$INIT:CONF. IMATP$INIT:CONF is responsible for storing CMEM parameters in the individual thread structures initializing the 2651's for asynchronous transmission, and initializing the remaining I/MATP data structures. Additionally, verification that the aggregate speeds of the terminals does not exceed the current limit of 9600 baud is done at this time. If the limit is exceeded, a message is sent to the operator and the system traps.

NOTE: It may be beneficial to store the thread number in the headers of the Receive and Transmit Holding Buffers and Echo Buffers. This allows us to proceed with one thread of communications by maintaining the address of the correct buffer in the X-register, without allocating additional space or tying up a register to keep track of the thread. In this way, repeated table look-ups during the Communications Interrupt and Protocol Handling routines are eliminated.

Entry Point: IMATP$INIT:ENTRY

Function:

Begin I/MATP data structure initialization.

Entry Conditions:

* None

Exit Conditons:

* All registers destroyed.

Entry Point: IMATP$INIT:REQ

Function:

Complete thread structure initialization and send configuration-request packets for all VP's to the Mainframe.

Entry Conditions:

* None

Exit Conditions:

* None

Entry Point: IMATP$INIT:CONF

Function:

Complete I/MATP data structure initialization based on the information contained in returned configuration-request packets.

Entry Conditions:

* None

Exit Conditions:

* None

6.17.2 Communications Interrupt Handling (submodule IMATP$COMM)

This submodule is responsible for servicing interrupts received from the 2651 communications circuits. Interrupt types include:

1) Receiver Ready
2) Transmitter Ready
3) Data Set (MODEM) Change

IPOS routes all 2651 IRQ's to a routine (IMATP$COMM:IRQ) that both determines interrupt type (by reading the ACS Register and 2651 Status Register) and does polling to locate the requesting 2651. The IRQ is dispatched to the processing routine corresponding to the interrupt type, with the X-register pointing to the Holding Buffer associated with the requesting 2651. (This address is found in the Correspondence Table mentioned in IMATP$INIT.)

After processing each data byte, the interrupt processors return control to IMATP$COMM:IRQ, which checks to see if there is more to do before terminating. To insure that no heavily loaded 2651 may monopolize processing time, IMATP$COMM:IRQ maintains a count of the number of data bytes processed for one thread during the current session. If the number exceeds the maximum permitted for one session, the routine will poll and service the other 2651's before granting the original requestor a second session.

NOTE: Polling of 2651's occurs in a pre-determined, fixed order which is based on the addresses of the D814 hardware. For polling purposes, low address ===> high priority. Thus, the 2651's on the QBYTE card with the lowest address have the highest priority and are polled first. Similarly, on any particular QBYTE card, the 2651 with the lowest address is polled first.

Entry Point: IMATP$COMM:IRQ

Function:

Dispatch 2651 IRQ's to their corresponding processing routines

Entry Conditions

* None

Exit Conditions

* None

Receiver Ready (IRQ) Routine (IMATP$COMM:RCV)

This routine is initiated by the occurrence of a Receiver Ready IRQ for a 2651. It is responsible for performing three functions:

1) Moving data from the 2651 to the corresponding Receiver Holding Buffer (and Echo Buffer, if auto echo).
2) Performing data encoding (as detailed in Section 6.11.2.1).
3) Detecting and recording line and parity errors.

When necessary, IMATP$COMM:RCV will fork the Protocol In Processor, first identifying the data stream by storing the address of the correct Receive Holding Buffer in the TCB for that Processor.

Transmitter Ready (IRQ) Routine (IMATP$COMM:XMT)

This routine is utilized to handle the 2651's transmitter section, and is initiated by the occurrence of a Transmitter Ready interrupt. It has four basic functions to perform:

1) Move data from a Transmit Holding Buffer (and Echo Buffer, if auto echo is enabled) to the corresponding 2651.
2) Decode data and modem control signals (as detailed in Section 6.11.2.1).
3) Process modem signal changes received from the network.
4) Execute software support of data parity (when enabled).

When necessary, IMATP$COMM:XMT will fork the Protocol Out Processor, first identifying the outgoing data stream by storing the address of the correct Transmit Holding Buffer in the TCB for that Processor:

NOTE: Modem signal changes are processed by setting the appropriate bits in the Auxiliary Control Signal Register and the 2651 Command Register. The 2651 Command register is detailed in the Signetics 2651 PCI document. Information concerning the ACSR structure and use is found in the Hardware System Specification. All equated bits of the ACSR are utilized in the I/MATP.

Data Set Change (IRQ) Routine (IMATP$COMM:DSC)

This routine is initiated by the occurrence of a Data Set Change interrupt from a Local 2651 or ACS Register. The data set signals that cause this type of interrupt include:

1) DCD - data carrier detect
2) DSR - data set ready
3) RI - ring in
4) Sec. TxD - secondary transmit data
5) Sec. RTS - secondary request to send

Modem changes are processed by isolating the bits of the 2651 Status Register and ACS Register which correspond to these signals and combining them to form a Modem Control Signal byte. This byte is encoded (as described in Section 6.11.2.1) and placed in the Receive Holding Buffer.

6.17.3 Protocol Handling (Submodules IMATP$IBP and IMATP$OBP)

The primary responsibility of the protocol handlers is the movement of data between the Encoder/Decoder routines and the Communications Transmit and Receive Holding Buffers. Additionally, they transfer modem signal changes to the Call Manager when required to initiate or terminate a call. There are two protocol processors:

1) Protocol In Processor
2) Protocol Out Processor

Protocol In processor (IMATP$IBP)

This protocol routine is forked by the Receiver Ready (IRQ) routine when data has been placed in a Receive Holding Buffer.

All data (excluding DSR signal changes) are ignored until a call has been established. While the call remains active, the contents of the appropriate Receive Holding Buffer are sent to the Data Compression Encoder with the thread number identified in the B-register.

Movement of data from a specific Receive Holding Buffer continues until the buffer is emptied OR the maximum number permitted during one session has been exceeded. At this point, the "next" Receive Holding Buffer is scanned for data, and the process is repeated until no data remains in any of the Receive Holding Buffers.

Receipt of ICS's indicating changes in a terminal's DSR signal are always transmitted to the Call Manager for processing. Additional responsibilities include the generation and sending of "calling" messages to remote terminals when attempting to establish dial calls.

Entry Point: IMATP$IBP:ENTRY

Entry Conditions:

* None

Exit Conditions:

* None

Protocol Out processor (IMATP$OBP)

This protocol routine is forked by Transmitter Ready (IRQ) routine or the Pre-ARQ/BIC Receiver routine when data is to be moved to a Transmit Holding Buffer. It calls the Data Compression Decoder to obtain data bytes from the corresponding Outbound Data Buffer, then selectively moves them into the Transmit Holding Buffer.

When the active Outbound Data Buffer is emptied OR the maximum number of bytes to be accepted from a terminal during one session has been exceeded, OR the Transmit Holding Buffer for that OB Data Buffer fills (whichever comes first), the "next" OB Data Buffer is scanned for data and the process is repeated.

When bytes are no longer available from any OB Data Buffer, the routine sets the ARQ Fork flag (indicating to the Pre-ARQ/BIC Receiver to fork this routine when data becomes available, and resets the Transmitter Fork flag (indicating to the Transmitter Ready (IRQ) routine not to fork this routine when any Transmit Holding Buffer falls below half full).

As bytes are transferred to a Holding Buffer, if its 2651 transmitter is not running, it is enabled. This creates a Transmitter Ready IRQ, which causes the Communications Interrupt Handler to start removing data from the Holding Buffers.

Any 'request call termination' or 'call terminate' signals received (see Section 6.11.2.1), are passed to the Call Manager for processing and are not placed in the Holding Buffers.

Entry Point: IMATP$OBP:ENTRY

Entry Conditions:

* None

Exit Conditions:

* None

6.17.4 Call Manager Interface (IMATP$CMI)

This submodule is responsible for handling communications between the Protocol modules and the Call Manager. It has one external entry point (IMATP$CMI:ENTRY), and performs five distinct tasks:

1)	Call End Processing	(CMM ---> Protocol)
2)	Call Request Processing	(CMM ---> Protocol)
3)	Call Created Processing	(CMM ---> Protocol)
4)	Hangup Request	(Protocol ---> CMM)
5)	Create Call Request	(Protocol ---> CMM)

The routines to process the first three are activated by IMATP$CMI:ENTRY upon receipt of an addressed packet from the Call Manager. The latter two are subroutines used by the Inbound and Outbound Protocol modules and the other CMI routines.

Entry Point: IMATP$CMI:ENTRY

Function:

Dequeue an addressed packet from the Call Manager and activate the appropriate routine to process it by identifying the command code contained in the addressed packet message field.

Entry Conditions:

* None

Exit Conditions:

* None

Call End Processing (IMATP$CMI:CALLEND)

This routine is activated by receipt of a "call end" AP from the Call Manager. At "normal" call end, it starts reinitialization of ARQ/FLOW control and resets call state variables. The routine also resends a "create call" AP to the Call Manager when the remote terminal has indicated it is "busy" OR the call was ended before it was fully established.

Call Request Processing (IMATP$CMI:CALLREQ)

This routine is invoked by the receipt of a "call request" AP from the Call Manager. It is responsible for rejecting calls when the port is already busy OR the "request" packet has been improperly routed. Otherwise the routine sends a "call accepted" AP to Call Manager.

Call Created Processing (IMATP$CMI:CALLCRE)

This routine is invoked following the receipt of a "call created" addressed packet from the Call Manager. It enables the 2651 receiver and BIC Inbound and Outbound FIFO's so a call can proceed.

Hangup Request (IMATP$CMI:SEND HANGUP)

This routine is called by the Protocol modules or the Call Manager Interface. Its function is to construct and route to the Call Manager an "hangup" addressed packet.

Create Call Request (IMATP$CMI:SEND CRECALL)

This routine is called by the Protocol modules or the Call Manager Interface. It handles the construction and routing of a "create call" addressed packet to the Call Manager.

===

At this point it seems desirable to diverge from module functional specifications in order to delineate and discuss those characteristics of the I/MATP which set it apart from the single-threaded I/ATP.

As indicated in the introductory paragraph of Section 6.17, the I/MATP passes data from 1 - 16 asynchronous terminals. Multiple terminals per set of port software create a number of complexities in establishing and maintaining communications. There are multiple 2651 communications chips; multiple Receive and Transmit Holding Buffers; multiple Inbound and Outbound Data Buffers; but only one BIC chip used to transmit the data from these across the network. Despite this, each individual terminal of the I/MATP must have the ability to converse with an I/ATP terminal, a terminal "belonging to" another I/MATP, or another terminal within the same I/MATP, <u>concurrent to active communications involving other terminals of the same I/MATP</u>. In addition to this, <u>multiplicity must be transparent across the link</u>, i.e., communication between an I/ATP and I/MATP must in no way differ from that betwen two I/ATP's from the I/ATP's perspective. In order to address this problem, the concepts of "thread" number and "Virtual Port" (VP) came into existence.

When a singly-threaded port requests call establishment, it identifies itself by means of node and port number. This is not sufficient routing information when dealing with multiply-threaded ports. In the case of an I/MATP, we know that each terminal sends and receives data via a specific 2651, and there will be no more than 16 terminals per port. For this reason (based on QBYTE-card number and 2651 hardware address) each terminal is uniquely identifiable by permanently associating with it a number ranging 0 - F. This number, the "thread" number, is mentioned throughout Section 6.16 and effectively creates 16 fixed data paths through the I/MATP.

It would appear that the addition of thread number to addressed packets for call establishment would be a quick and simple solution to the routing problem. However, this violates our second requirement, that of transparency. In order to maintain this transparency, Virtual Port numbers may be assigned to each I/MATP by the mainframe during port initialization. The Call Manager maintains a table which associates with each Virtual Port number a specific thread number. Communications between the Call Manager and a terminal are identified by thread number; addressed packets travelling across the network contain Virtual Port numbers. The IPOS addressed packet system has the responsibility of changing the AP field containing destination port so that it is meaningful to the receiver of the addressed packet. (More detailed information on this can be found in Sections 6.1, 6.3 and 6.5 of this document.)

The following is a summary which may help to clarify the points brought up in the preceeding discussion.

Assume throughout that we are dealing with communications internal to a single node. The node consists of three ports:

Port 12 -- IMATP with one QBYTE
Port 20 -- IATP
Port 56 -- IMATP with two QBYTEs

At initialization time, the mainframe allocates:

VP numbers 3,5,7,9 to Port 12
VP number 35,37,39,41,43,45,47,49 to Port 56

The relationship between thread numbers and VP numbers within the two IMATP's is as follows:

Port 12

Thread #	VP#
0	3
1	5
2	7
3	9

Port 56

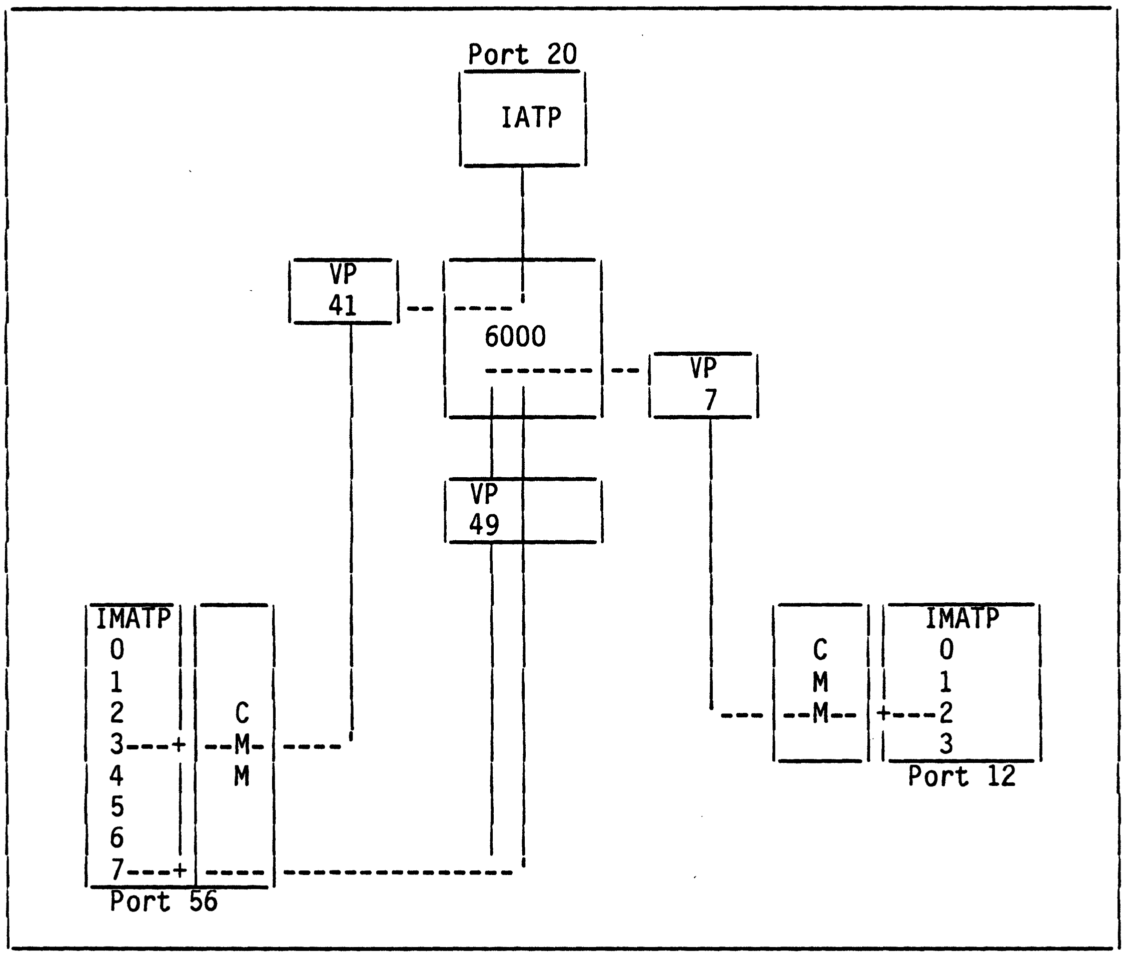
Thread #	VP#
0	35
1	37
2	39
3	41
4	43
5	45
6	47
7	49

Note: These are permanent assignments in the IMATP.

No matter what two terminals are communicating (even when two IMATPs), each Call Manager believes that the remote terminal is an IATP. For instance, when Port 20 communicates with Port 56, Thread 3, its Call Manager believes it is talking to Port 41. Similarly, when Port 56, Thread 7 communicates with Port 12, Thread 2 the Call Manager for Port 12 believes the communication is with Port 49, the Call Manager for Port 56 believes the communication is with Port 7.

When the IPOS addressed packet router for Port 12 receives an AP for Port 7, it recognizes it as referring to Thread 2.

The following diagram graphically illustrates the data paths described above.



NODE 6

Figure 6.17.1

In order to gain a full understanding of how the I/MATP submodules interact to accomplish their task, it is helpful to study the diagrams in Section 6.6.11 which briefly trace the path of data and logic through the I/ATP.

The only differences in program control flow for the I/MATP are: AP is replaced by CMI, the Data Compression module is fixed, not adaptive, and FLOW$ represents the multithread ARQ/Flow control submodules (MFLOW$).

What must also be kept in mind is that there are multiple terminals involved, hence a number of calls may be in progress at any given time.

===

6.17.5 Statistics and Monitoring (IMATP$STAT)

This routine is responsible for monitoring the performance of the I/MATP and reporting the information as a system alarm to the network Report Port. In addition, any system module can request current information from this module.

The individual modules in the I/MATP are responsible for updating monitoring information as follows:

1) Receiver Ready interrupt routine

   a) Number of characters received
   b) Number of errors that occurred
   c) Number of characters after encoding

2) Data Compression Encoder

   a) Number of nibbles encoded
   b) Number of resulting nibbles

The statistics routine is started by IP initialization and runs every 6 seconds. Each time it executes, it calculates the following port level statistics:

1) Buffer Utilization - The ratio of the number of buffers currently in use to the total number of buffers in the free buffer pool, times 100%.

2) Processor Loading - As calculated by IPOS (see Section 6.1.10).

The results of these calculations are compared to their respective thresholds, which are obtained from the mainframe during initialization. If either threshold has been exceeded, an alarm packet is constructed and sent to the network Report Port.

If a packet is received from system module requesting terminal statistics, an addressed packet containing the above information, as well as:

1) Statistical Loading - The ratio of the number of characters from the terminal to the maximum number of characters which the terminal could have sent in the elapsed time, times 100%.

2) Compressed Loading - The ratio of the number of bits resulting from code compression to the maximum number of bits (including parity bit) the terminal could have sent in the elapsed time, times 100%.

3) Compression Efficiency - the ratio of the total number of bits (including parity bit) from the terminal to the number of bits resulting from code compression, times 100%.

4) Character Error Rate - The ratio of the number of characters with bad parity received from the terminal to the total number of characters received, times 100%.

is constructed and returned to the requesting module.

Note: Statistics must be requested by node and VIRTUAL PORT number in the I/MATP.

Entry Point: IMATP$STAT:MONITOR

Function:

Calculate statistics and send exception report addressed packets to the network Report Port

Entry Conditons:

*   None

Exit Conditions:

*   None

Entry Point: IMATP$STAT:AP

Function:

Calculate statistics and send a statistics addressed packet to the requesting module.

Entry Conditons:

*   None

Exit Conditions:

*   None

6.18 Intelligent Multiple Synchronous Terminal Port Protocol Software

The Intelligent Multiple Synchronous Terminal Port Protocol Module (IMSTP) passes data from 1-16 synchronous terminals to the D814 Network without protocol intervention. It is organized as a collection of submodules, each responsible for performing a related set of functions. These functions include:

1. System Initialization
2. Communications Interrupt Handling
3. Protocol Handling
4. Call Manager Interface
5. Statistics and Exception Monitoring

Descriptions of the submodules which perform these tasks are found in Sections 6.18.1 through 6.18.5.

The I/MSTP user data structures, called thread structures, are data areas reserved for each configured terminal. The thread structures are initialized and accessed by calls to module ITP$ and are discussed in detail in Section 6.16.

6.18.1 System Initialization (Submodule IMSTP$INIT)

This submodule consists of three distinct routines. The first (IMSTP$INIT:ENTRY) is called by IPOS during port initialization to begin protocol-specific initialization tasks; the second (IMSTP$INIT:REQ) is forked by IMSTP$INIT:ENTRY; and the third routine is a batch task that is scheduled on receipt of an addressed packet containing configuration information.

IMSTP$INIT:ENTRY is responsible for starting common multithread IP initialization routines (i.e., Data Compression, Call Manager, FIFO Interrupt Handler and ARQ/Flow Control) and calling ITP$MTINIT:USER to allocate a block of contiguous memory to be later divided into thread structures for use by assigned VP's. It additionally obtains and initializes buffers to be used as permanent TCB's by IMSTP modules, sets up the port Module Dispatch Table, creates a temporary byte file containing addresses of ring buffers allocated for data storage by 2651 interrupt routines, and forks IMSTP$INIT:REQ before terminating.

IMSTP$INIT:REQ first calls ITP$MTINIT:CMEM to request VP numbers configured for the port and await 1) the division and formatting of the previously allocated block of contiguous memory into thread structures (one for each VP), and 2) the release of all unused memory within this block. Upon return from this routine, the thread directory created by ITP$MTINIT:AP is scanned. For each valid thread, a configuration-request packet is constructed and routed, the thread structure variables are initialized, and previously allocated ring buffers are assigned as receive and transmit holding buffers. When thread initialization is complete, all unassigned ring buffers are released and the routine terminates.

IMSTP$INIT:CONF dequeues configuration addressed packets, storing CMEM parameters in the individual thread structures. In addition to this, it initializes the 2651's for synchronous transmission, calculates parameters used in statistics and monitoring tasks and verifies that the aggregate speeds of the terminals does not exceed the current 9600 baud limit. Before terminating, the routine sets the bits in OF$IP$COMM:INIT_FLAGS indicating to the Call Manager that protocol initialization is complete.

NOTE: It may be beneficial to store the thread number in the headers of the Receive and Transmit Holding Buffers. This allows us to proceed with one thread of communications by maintaining the address of the correct buffer in the index register without allocating additional space or tying up a register to keep track of the thread. In this way, repeated table look-ups during the Communications Interrupt and Protocol handling routines are eliminated.

Entry Point - IMSTP$INIT:ENTRY

Function

Initialize I/MSTP data structures, start IP common modules and obtain 2651 Holding Buffers.

Entry Conditions

*   None

Exit Conditions

*   All registers destroyed

Entry Point - IMSTP$INIT:REQ

Function

Build a configuration-request packet

Entry Conditions

*   None

Exit Conditions

*   None

Entry Point - IMSTP$INIT:CONF

Function

Complete I/MSTP data structure initialization based on the information contained in returned configuration-request packet.

Entry Conditions

* None

Exit Conditions

* None

6.18.2 Communications Interrupt Handling (Submodule IMSTP$COMM)

This submodule is responsible for servicing interrupts tracked by the ACS Status Register. IMSTP IRQ's are generated by the following:

1. 2651 Communications Chip
2. ACS Register

All such interrupt requests are routed to IMSTP$COMM:IRQ that both determines interrupt type (by reading the ACS Register and 2651 Status Register) and does polling to locate the requesting 2651. The IRQ is dispatched to the corresponding interrupt processor with the index register pointing to the holding buffer associated with the active thread.

2651 interrupts are of three types: receiver ready, transmitter ready, and data set change. ACS interrupts are processed identically to 2651 data set changes.

After processing each data byte the interrupt processors return control to IMSTP$COMM:IRQ, which checks for outstanding interrupts. To insure that no heavily loaded 2651 may monopolize processing time, IMSTP$COMM:IRQ maintains a count of the number of data bytes processed for one thread during the current session. If the number exceeds the maximum permitted for one session, the routine will poll and service the other 2651's before granting the original requestor a second session.

NOTE: Polling of 2651's occurs in a predetermined, fixed order which is based on the addresses of the D814 hardware. For polling purposes, low address ===> high priority. Thus, the 2651's on the QBYTE card with the lowest address have the highest priority and are polled first. Similarly, on any particular QBYTE card, the 2651 with the lowest address is polled first.

Entry Point - IMSTP$COMM:IRQ

Function

Poll threads to locate requestor and dispatch ACS Status Register IRQ's to their corresponding interrupt routines.

2651 Receiver Ready (IRQ) Routine (RECEIVE)

This routine is utilized to handle the 2651's receiver section, being initiated by the occurrence of a receiver ready interrupt. It performs 5 functions:

1. Moves data from the 2651 to the Receive Holding Buffer.
2. Performs data encoding (as detailed in Section 6.18.2.1).
3. Detects and records line and parity errors.
4. Inserts Parity Check ICS's.
5. Disables 2651 receiver between blocks to force sync hunt.

When necessary, RECEIVE will fork the Protocol In Processor, first identifying the data stream by storing the address of the correct Receive Holding Buffer in the TCB for that processor.

Transmitter Ready (IRQ) Routine (TRANSMIT)

This routine is utilized to handle the 2651's transmitter section, being initiated by the occurrence of a transmitter ready interrupt. It has 4 basic functions to perform:

1. Move data from the Transmit Holding Buffer to the 2651.
2. Decode data and modem control signals (as detailed in Section 6.18.2.1).
3. Line fill (either syn or pad characters) when unable to transmit data.
4. Process modem signal changes received from the network.

When necessary, TRANSMIT will fork the Protocol Out Processor, first identifying the outgoing data stream by storing the address of the correct Transmit Holding Buffer in the TCB for that Processor.

NOTE: Modem signal changes are processed by setting the appropriate bits in the Auxiliary Control Signal Register and the 2651 Command Register. The 2651 Command Register is detailed in the Signetics 2651 PCI document. Information concerning the ACSR structure and use is found in the Hardware System Spec. It is useful to note that the only bits of the ACSR which are utilized in the I/MSTP (other than IRQ) are BUSY (ACS_IN), CTS and RNG (ACS_OUT). CTS is used only when Clear to Send Delay is activated.

2651 or ACS Register DATA Set Change (IRQ) Routine (DSCHG)

This routine is responsible for handling Data Set (Modem) changes. It is initiated by the occurrence of a data set change interrupt from the 2651 or ACSR. The data set signals that can cause this interrupt (at the local 2651 or ACSR) are:

1. DCD - Data Carrier Detect
2. DSR - Data Set Ready
3. RI - Ring In

Modem changes are processed by isolating the bits of the 2651 Status Register and ACS Register which correspond to these signals and combining them to form a Modem Control Signal byte. This byte is encoded (as described in Section 6.18.2.1) and placed in the Receive Holding Buffer.

6.18.2.1 Data Encoding/Decoding

Data encoding is performed as follows:

```
        Data (D) ==>           (D)| if 0 < (D) < X'FF'
                       (X'FF',X'81')| if (D) = 0
                       (X'FF',X'FF')| if (D) = X'FF'

Modem Signals (S) ==>     (X'FF',S)| if (S) > 0
                       (X'FF',X'80')| if (S) = 0

    Call Signals  ==>(X'FF',X'82')| Request Call Termination
                       (X'FF',X'83')| Call Termination Granted

        Note:  S≤X'7F' (i.e, high order bit must be zero)
```

Data decoding is the inverse of the encoding procedure, except that Call Signals are not decoded. The Call Signals described above are special signals used by the CMM Interface to terminate a call.

6.18.3 Protocol Handling (Submodules IMSTP$IBP and IMSTP$OBP)

The primary responsibility of the protocol handlers is the movement of data between the Encoder/Decoder routines and the Communications Transmit and Receive Holding Buffers. Additionally, they transfer modem signal changes to the Call Manager when required to initiate or terminate a call. There are two protocol processors:

1. Protocol In processor
2. Protocol Out processor

Protocol In Processor (IMSTP$IBP)

This protocol routine is forked by the Receiver Ready (IRQ) routine when data has been placed in a Receive Holding Buffer.

All data (excluding DSR signal changes) are ignored until a call has been established. While the call remains active, the contents of the appropriate Receive Holding Buffer are sent to the Data Compression Encoder with the thread number identified in the B-register.

Movement of data from a specific Receive Holding Buffer continues until the buffer is emptied OR the maximum number permitted during one session has been exceeded. At this point, the "next" Receive Holding Buffer is scanned for data, and the process is repeated until no data remains in any of the Receive Holding Buffers.

Receipt of ICS's indicating changes in a terminal's DSR signal are always transmitted to the Call Manager for processing.

Entry Point - IMSTP$IBP:ENTRY

Entry Conditions

* None

Exit Conditions

* None

Protocol Out Processor (IMSTP$OBP)

This protocol routine is forked by Transmitter Ready (IRQ) routine or the Pre-ARQ/BIC Receiver routine when data is to be moved to a Transmit Holding Buffer. It calls the Data Compression Decoder to obtain data bytes from the corresponding Outbound Data Buffer and "tracks" them by means of a finite state machine to insure that the appropriate mode of sync-filling (transparent or normal) will occur only where permissible within the BSC message block.

When the active Outbound Data Buffer is emptied OR the maximum number of bytes to be accepted from a terminal during one session has been exceeded, OR the Transmit Holding Buffer for thè OB Data Buffer fills (whichever comes first), the "next" OB Data Buffer is scanned for data and the process is repeated.

When bytes are no longer available from any OB Data Buffer, the routine sets the ARQ Fork flag (indicating to the Pre-ARQ/BIC Receiver to fork this routine when data becomes available, and resets the Transmitter Fork flag (indicating to the Transmitter Ready (IRQ) routine not to fork this routine when any Transmit Holding Buffer falls below half full).

As bytes are transferred to a Holding Buffer, if its 2651 transmitter is not running, it is enabled. This creates a Transmitter Ready IRQ, which causes the Communications Interrupt Handler to start removing data from the Holding Buffers.

Any 'request call termination' or 'call terminate' signals received (see Section 6.18.2.1) are passed to the Call Manager for processing and are not placed in the Holding Buffers.

Entry Point - IMSTP$OBP:ENTRY

Entry Conditions

*   None

Exit Conditions

*   None

6.18.4 Call Manager Interface (IMSTP$CMI)

This submodule is responsible for handling communications between the Protocol modules and the Call Manager. It has one external entry point (IMSTP$CMI:ENTRY), and performs five distinct tasks:

1.	Call End Processing	(CMM ---> Protocol)
2.	Call Request Processing	(CMM ---> Protocol)
3.	Call Created Processing	(CMM ---> Protocol)
4.	Hangup Request	(Protocol ---> CMM)
5.	Create Call Request	(Protocol ---> CMM)

The routines to process the first three are activated by IMSTP$CMI:ENTRY upon receipt of an addressed packet from the Call Manager. The latter two are subroutines used by the Inbound and Outbound Protocol modules and the other CMI routines.

Entry Point - IMSTP$CMI:ENTRY

Function

Dequeue an addressed packet from the Call Manager and activate the appropriate routine to process it by identifying the command code contained in the addressed packet message field.

Entry Conditions

*   None

Exit Conditions

*   None

Call End Processing (IMSTP$CMI:CALLEND)

This routine is activated by receipt of a "call end" AP from the Call Manager, At "normal" call end, it starts reinitialization of ARQ/FLOW control and resets call state variables. The routine also resends a "create call" AP to the Call Manager when the remote terminal has indicated it is "busy" OR the call was ended before it was fully established.

Call Request Processing (IMSTP$CMI:CALLREQ)

This routine is invoked by the receipt of a "call request" AP from the Call Manager. It is responsible for rejecting calls when the port is already busy OR the "request" packet has been improperly routed. Otherwise the routine sends a "call accepted" AP to Call Manager.

Call Created Processing (IMSTP$CMI:CALLCRE)

This routine is invoked following the receipt of a "call created" addressed packet from the Call Manager. It enables the 2651 receiver and BIC Inbound and Outbound FIFO's so a call can proceed.

Hangup Request (IMSTP$CMI:SEND HANGUP)

This routine is called by the Protocol modules or the Call Manager Interface. Its function is to construct and route to the Call Manager a "hangup" addressed packet.

Create Call Request (IMSTP$CMI:SEND CRECALL)

This routine is called by the Protocol modules or the Call Manager Interface. It handles the construction and routing of a "create call" addressed packet to the Call Manager.

= =

At this point it seems desirable to diverge from module functional specifications in order to delineate and discuss those characteristics of the I/MSTP which set it apart from the singly-threaded I/STP.

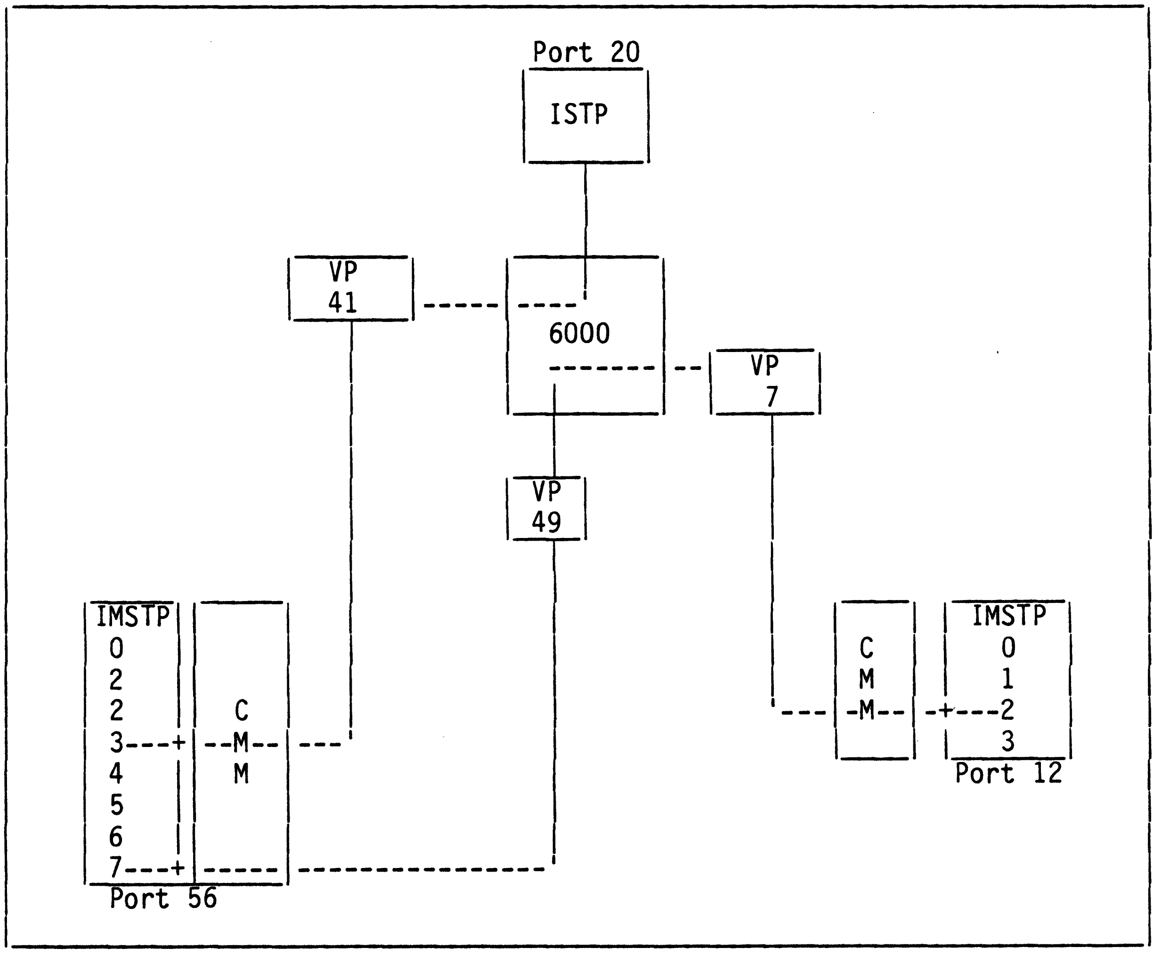
As indicated in the introductory paragraph of Section 6.18, the I/MSTP passes data from 1-16 synchronous terminals. Multiple terminals per set of port software create a number of complexities in establishing and maintaining communications. There are multiple 2651 communications chips; multiple Receive and Transmit Holding Buffers; multiple Inbound and Outbound Data Buffers; but only one BIC chip used to transmit the data from these across the network. Despite this, each individual terminal of the I/MSTP must have the ability to converse with an I/STP terminal, a terminal "belonging to" another I/MSTP, or another terminal within the same I/MSTP, concurrent to active communications involving other terminals of the same I/MSTP. In addition to this, multiplicity must be transparent across the link, i.e., communications between an I/STP and I/MSTP must in no way differ from that between two I/STP's from the I/STP's perspective. In order to address this problem, the concepts of "thread" number and "Virtual Port" (VP) came into existence.

NOTE: These are permanent assignments in the IMSTP.

No matter what two terminals are communicating (even when two IMSTPs), each Call Manager believes that the remote terminal is an ISTP. For instance when Port 20 communicates with Port 56, Thread 3, its Call Manager believes it is talking to Port 41. Similarly, when Port 56, Thread 7 communicates with Port 12, Thread 2 the Call Manager for Port 12 believes the communication is with Port 49, the Call Manager for Port 56 believes the communication is with Port 7.

When the IPOS addressed packet router for Port 12 receives an AP for Port 7, it recognizes it as referring to Thread 2.

The following diagram graphically illustrates the data paths described above.



NODE 6

Figure 6.18.1

In order to gain a full understanding of how the I/MSTP submodules interact to accomplish their task, it is helpful to study the diagrams in Section 6.12 which briefly trace the path of data and logic through the I/STP.

The only differences in program control flow for the I/MSTP are: the Data Compression module is fixed, not adaptive, and FLOW$ represents the multithread ARQ/Flow control submodules (MFLOW$).

What must also be kept in mind is that there are multiple terminals involved, hence a number of calls may be in progress at any given time.

6.18.5 Statistics and Monitoring (Submodule IMSTP$STAT)

This submodule is responsible for monitoring the performance of the IMSTP, reporting error and exception conditions to the network report port, and responding to requests for port statistics. It has three external entry points:

1. IMSTP$STAT:COLLECT_STATS
2. IMSTP$STAT:STAT
3. IMSTP$STAT:MONITOR

Entry Point - IMSTP$STAT:COLLECT_STATS

Function

Collects instantaneous values for processor loading, encoder nibbles in, encoder nibbles out, number of buffers in use, number of characters received from the 2651, and current error count. Forks the monitoring routine (IMSTP$STAT:MONITOR). (This routine is initiated by IPOS every 6 seconds.)

Entry Conditions

* None

Exit Conditions

* All registers destroyed

Entry Point - IMSTP$STAT:MONITOR

Function

Updates the weighted-average value for each statistic using the current instantaneous value. Calculates instantaneous processor load, buffer utilization, compression efficiency and character error rate, comparing percentages to threshold values and reporting exceptions to Network Report Port.

6.19 Intelligent Multiplex Port (I/MXP) Protocol Module

6.19.1 Introduction

An Intelligent Multiplex Port (I/MXP) is an intelligent terminal port designed to interface with a Multiplex Port of another 6000 series Intelligent Network Processor or a Front-End Processor over a high-speed link in order to establish multiplexed data paths according to the Codex Multiplex Protocol (see Codex Multiplex Protocol Specification).

The I/MXP uses a similar hardware and the same line layer (HDLC) protocol as the intelligent Bit-Oriented Protocol terminal port (I/BOP, see Section 6.15). The hardware consists of an I/ENG2 card and an I/BIT daughter card based on a Motorola MC6854 ADLC chip and an Auxiliary Control Signal Register (ACSR).

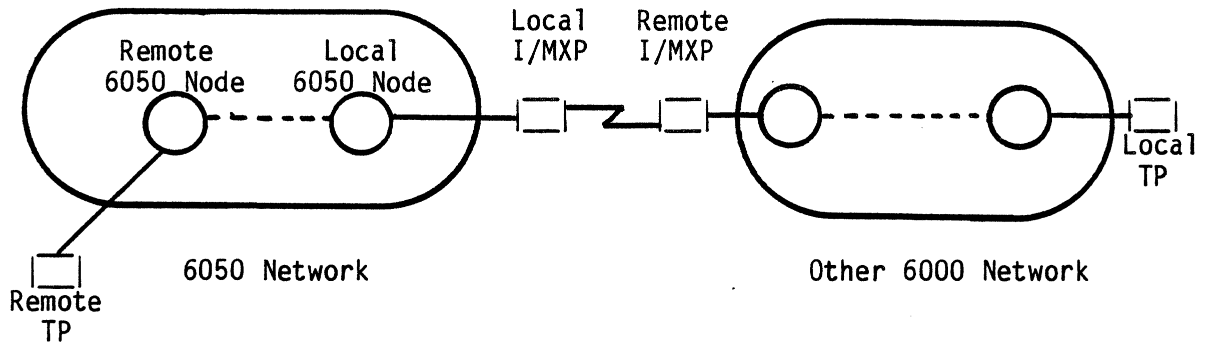
The I/MXP supports all types of threads; asynchronous, binary synchronous (BSC), and bit-oriented protocol threads.

Running under the 6809 IP Operating System (IPOS Ø9), and interfacing with the Multi-threaded Data Movement Module (MTDM, Section 6.5), Call Manager Module (CMM, Section 6.3) and IP Configuration Control Module (CCM, Section 6.2), the I/MXP-specific module that is described here is divided into the following submodules:

1. Initialization
2. Line Layer Protocol Handling
3. ARQ Layer Protocol Handling
4. MUX Layer Protocol Handling
5. Connection Layer Protocol Handling
6. Call Manager Interfacing
7. Statistics Gathering and Monitoring

Before going into the description of the I/MXP Protocol Module, a few naming conventions need to be defined here.

Each multiplexed data path which is a logical construct in software within the local I/MXP is called a thread. The remote terminal port at the end of a 6050 subnetwork, be it an I/TP or a thread in another I/MXP, is called a remote TP. Away from the 6050 subnetwork, the I/MXP is connected to another Multiplex Port, which is called a remote MXP, via a high-speed link called MUX link. The local TP is the TP in the other subnetwork which is eventually connected to the remote TP for the thread through the MUX link.



6.19.2 <u>Functional Submodule Description</u>

In this section the functions of an I/MXP are described as belonging to one of the 7 submodules listed above.

6.19.2.1 Initialization Submodule (IMXP$INIT:)

The function of this submodule is to make the I/MXP ready for users. Called by IPOS Ø9 during the port initialization, IMXP$INIT:START initializes common multi-threaded I/TP modules and allocates storage blocks for I/MXP data structures and initializes them. Finally, it forks IMXP$INIT:REQ_PORT_CONF and returns to IPOS Ø9. IMXP$INIT:REQ_PORT_CONF then, started by the IPOS Ø scheduler, calls ITP$MTINIT:CMEM and sends addressed packets to the Mainframe Configuration Control Module (see Section 5.7) to request for thread configuration parameters which are eventually received by IMXP$INIT: CONF (module number EQ$IP$MDT:IPP_INIT). These addressed packet formats are described in Section 5.7.3. When a thread is configured, call is set up for the thread if it is specified as a leased-line connection.

After tables of configuration parameters are built by this submodule, it is used by the IP Configuration Control Module for configuration reads or updates (see Section 6.2).

The configurable parameters for an I/MXP as a whole are the following:

- Port generic type
- Port sub-type
- Speed
- Mode (normal/local loopback/remote loopback)
- Control field extension (Extended ARQ sequencing)
- Processor loading threshold
- Buffer utilization threshold
- Frame receive error rate threshold
- Retransmit frame rate threshold
- Statistics averaging time constant factor

The configurable parameters for each thread are the same as the VP configuration parameters for the asynchronous, BSC, and BOP threads except that a new parameter 'Slot Weight' is added.

6.19.2.2 Line Layer Protocol Handler Submodule (IMXP$LINE:)

This submodule has the complete control over the line communications section hardware of the I/MXP. Entry to this submodule is via an interrupt for which the IPOS Ø9 gives control to IMXP$LINE:IRQ.

Entry Point - IMXP$LINE:IRQ

After determining the cause of the interrupt, this routine calls one of the following three routines whose functions are described in the subsequent subsections.

1. Line Transmitter Controller
2. Line Receiver Controller
3. Modem Signal Change Handler

1. Line Transmitter Controller (IMXP$LINE:XMT)

   This routine is called by IMXP$LINE:IRQ if the interrupt was originated by the 6854 transmitter.

   For each transmitter interrupt, a character is read and transmitted from the current byte queue. If the current byte queue is empty, the processing depends on whether there is a slot byte queue enqueued in a chain. If there is one, the routine makes it the current slot byte queue, and reads and transmits its first character. If there is none, the frame is terminated. As a character is transmitted it is also put into the frame byte queue if it came from a chained byte queue in order to save the frame byte queue in the most compact form.

   The Line Transmitter Controller transmits all user data collected up to the time when it gets to the thread; therefore, delay is minimized.

   When there is no more character in the frame, the routine terminates the frame and disables further transmitter interrupts by controlling the 6854 transmitter. Then the pointer to the slot byte queue chain is saved in the transmit frame byte queue header, and the ARQ Transmitter task is fast-forked.

2. Line Receiver Controller (IMXP$LINE:RCV)

   Called by IMXP$LINE:IRQ if the interrupt was originated by the 6854 receiver, this routine first checks the cause of the interrupt.

   If it is a Receiver Data Available interrupt, a byte is retrieved from the 6854 receiver and placed in a receive frame byte queue which is being assembled. If the interrupt is due to an end of a frame, the receive frame byte queue is enqueued to the input job queue of the ARQ Layer Receiver task and it is forked if it is not already active.

   However, if the frame is aborted or received with a bad FCS, the receive frame byte queue is just returned to the free buffer pool. Thus, the Line Receiver Controller passes only good frames to the ARQ Layer Receiver task.

   The reception of a Remote Reset command, which is a special abort frame (see Codex Multiplex Protocol Specification), causes an interrupt with the postamble command byte saved in the Remote Loopback Register. This is used to recognize any change of normal/lopback mode in the MUX link.

3. Modem Signal Change Handler (IMXP$LINE:MODEM)

   All the input modem signals, DSR, DCD, and CTS, are to be strapped high so that none of those signals would cause an interrupt. The recovery from a temporary link-down condition can be done under the Multiplex Protocol.

   Therefore, the Auxiliary Control Signal (ACS) register interrupt will be trapped.

6.19.2.3 ARQ Layer Protocol Handler Submodule (IMXP$ARQ:)

This submodule is responsible not only for maintaining information transfer but also for setting up the MUX link according to the Multiplex Protocol.

There are two ARQ layer protocol handling tasks, one for each direction:

1. ARQ Layer Transmitter Task
2. ARQ Layer Receiver Task

The functions of these are described below:

1. ARQ Layer Transmitter Task (IMXP$ARQ:XMT)

   This task is fast-forked either by the ARQ Layer Receiver task when a Supervisory frame (S-frame) or Unnumbered frame (U-frame) needs to be transmitted or by the Line Transmitter Controller routine when a frame has been transmitted.

   When the task is started, it takes the previous frame's byte queue from the transmit frame pointer and tests its control byte. If it is an S or U-frame, the byte queue is destroyed. If it is an I-frame in the normal information transmit state, it is enqueued to the retransmit queue and the chained slot byte queues are destroyed. However, if it is an I-frame in the information retransmit state nothing is done to the frame.

   If there is an S or U-frame to be transmitted, or an I-frame to be retransmitted, the ARQ transmitter task removes it from the frame queue, sets up correct control (C) field, and and places its pointer in the transmit frame pointer. Otherwise, it creates a new frame byte queue, puts an I-frame header, and calls MUX transmit. Then it chains the first slot byte queue to be transmitted to the transmit frame byte queue.

   After the pointer to the frame has been set up for the Line Transmitter Controller, the 6854 transmitter interrupt is enabled. Finally, the ARQ transmitter task is terminated.

2. ARQ Layer Receiver Task (IMXP$ARQ:RCV)

   This task is forked by the Line Receiver Controller when a receive frame byte queue is enqueued to its input job queue and the queue was previously empty.

   When started, it dequeues a receive frame byte queue from the top of its input job queue and processes the A and C fields by feeding the ARQ Layer Transmitter task with received ARQ information and also by controlling the MUX link as needed. A FRMR frame is fully processed by this task, resulting in a transfer to the link set-up state.

   If the frame is either an S or U-frame, it is completely processed by the task and the byte queue is returned to the free buffer pool. If an S or U-frame needs to be transmitted in response, an S or U-frame byte queue is generated and enqueued to the S and U-frame input job queue of the ARQ Layer Transmitter task. At this time the ARQ Layer Transmitter task is forked.

   If the received frame is an I-frame, however, it is passed to the MUX Layer Receiver task by enqueuing the receive frame byte queue to the input job queue of the MUX Layer Receiver task and forking it.

6.19.2.4 MUX Layer Protocol Handler Submodule (IMXP$MUX:)

There are 4 MUX Layer protocol handling tasks:

1. MUX Layer Transmitter Routine
2. MUX Layer Receiver Task
3. MUX Layer Control Transmitter Task
4. MUX Layer Control Receiver Task

The functions of these tasks are described below.

1. MUX Layer Transmitter Routine (IMXP$MUX:XMT)

   MUX Layer transmitter is called by the ARQ Layer Transmitter task whenever a frame may be sent. It first writes the MUX layer control byte, and copies a control slot command from the MUX Layer control slot byte queue if there is any to be transmitted, then returns to the caller.

2. MUX Layer Receiver Task (IMXP$MUX:RCV)

   This task is forked by the ARQ Layer Receiver task when a received I-frame has been already processed for the A and C fields.

   If a MUX layer control slot is present in the frame, it is enqueued to the receive MUX layer control slot byte queue and the MUX Layer Control Receiver task is forked unless it is already active.

   Then the task calls the Connection Layer Receiver routine to process all supervisory and user slot data in the frame.

   The task terminates when there is no more received I-frame byte queue to process.

3. MUX Layer Control Transmitter Task (IMXP$MUX:CTRL_XMT)

   When an I/CTP in a 6050 network needs to read or write information such as configuration parameters, statistics, or statistics thresholds maintained by a TP connected to the remote MXP, it may send an addressed packet to this batch task.

   The task retrieves an addressed packet from its input job queue, converts it to a MUX layer control slot command (see Appendix A of the Codex Multiplex Protocol Specification), and then enqueues it to the input job queue of the MUX Layer Transmitter routine.

The I/MXP is always a controlling unit since the MUX layer control interface is implemented only for a MXP of a lower-level 6000 series INP. Therefore, the task only needs to Transmit commands.

The task runs in a half-duplex mode in the sense that the response must be received by the MUX Layer Control Receiver task for a transmitted command before it transmits another command.

However, if the response is not received within 20 seconds, the task will effectively discard the previous command and send a new one.

4. MUX Layer Control Receiver Task (IMXP$MUX:CTRL_RCV)

Forked by the MUX Layer Receiver task, this task dequeues the next MUX layer control slot response from its input job queue.

The task checks whether the response in the MUX layer control slot is the one that is expected. If it is the correct response, the task routes a response AP to the source node/port of the original command, and then forks the MUX Layer Control Transmitter task to transmit another command, if there is any, before the task terminates itself. If the response is not the correct one, it is discarded, and the next MUX layer control slot response is dequeued from its input job queue. If there is none, the task terminates.

For the relationship between any two Multiplex Ports of different systems, refer to the Single Line Interface Functional Specification.

6.19.2.5 Connection Layer Protocol Handler Submodule (IMXP$CONN:)

This submodule performs multiplexing and demultiplexing of the individual data threads between the Multi-Threaded Data Movement module (MTDM) and the MUX layer. It is also responsible for using CMI to establish and terminate calls in response to state transitions in the individual threads.

Connection layer is implemented as a family of coupled finite state machines. State information is shared in a common table called the Thread Data structure. There is one thread data structure for each thread supported. Due to space restrictions and the added complexity that would be introduced by providing multiple slot groups, there may be no more than 31 slots in the MUX port. Therefore, all slots are assumed to be in slot group number Ø.

It is a functional requirement of the MUX port that it provide a mapping between the 6050 Circuit Control Signals (CCS) and the In-Stream Control Codes (ISCC) used in the Multiplex Protocol. This mapping is shown below.

ISCC		CCS
User Data (D)	<--->	(D) if X'Ø2' < (D) < X'FF'
(X'FF')	<--->	(X'FF', X'82')
Control Signal Update (CSU) (X'01',	<--->	Modem CCS (X'FF',

```
| 0 | 0 | 0 | 1 | * | MB | RTS | DTR |)  | 0 | 0 | 0 | 0 | 0 | RI | DCD | DSR |)
```

where * means that the bit is not used.

The bit correspondence between the two modem signals are:

```
DTR  <--->  DSR
RTS  <--->  DCD
MB   <--->  RI
```

But,

(X'01',	<--->	(X'FF', X'80')

```
| 0 | 0 | 0 | 1 | * | 0 | 0 | 0 |)
```

Data Path Initialization (DPI) (X'01', X'20')	<---	Call Termination CCS (X'FF', X'83')

When a DPI is received from the remote MXP, it is discarded and a DPIA is transmitted back to the remote MXP for the data path. (Call Termination CCS has been already generated within 6050 network due to the prior reception of CSU for DTR down for auto-dial, dial, or contention call option.)

Data Path Initialization Acknowledgement (DPIA) (X'01', X'30')	--->	Discarded
Control Signal Update Request (CSUR) (X'01', X'40')	--->	Processed and discarded
Start Break (X'01', X'70')	--->	Break for 500 msec (X'FF', X'93', X'FF', X'92')
Stop Break (X'01', X'80')	--->	Discarded

Break for n-character time (X'01', X'7n')	<--->	Break for a time period (X'FF', X'9n_1', X'FF', X'9n_2') (n_1, n_2) makes an integer in 10 msec units.
Autospeed Initialization (X'01', X'An')	<--->	Autospeed Initialization CCS (X'FF', X'An')
Escape Output (X'01', X'01')	<--->	X'01' as data (X'01')

There are some other ISCC's that are required to support BOP data traffic. They are not listed above since they are not specified yet.

There are two routines in this submodule that are described below.

1. Connection Layer Transmitter Routine (IMXP$CONN:XMT)

   As the MTDM outbound interface, the routine is responsible for receiving characters from MTDM, translating CS strings to ISCC strings where necessary and placing them onto either the thread slot hold byte queue or a thread slot-weight overflow byte queue.

   If additional characters may not be placed in the current slot byte queue because of flow control, or if a slot flow-control overflow byte queue already exists for the thread, the character or string is placed on the slot flow-control overflow byte queue, which will be created if it does not already exist. If the slot weight is overflowed, a new byte queue is created and chained to the previous slot byte queue. In a critical section it tests for the presence of a slot hold byte queue pointer in the thread data structure, creating one if none is present. It then updates the count field of the slot and places the character in the slot's byte queue. The protocol requirement that slots may have one or two byte headers is hanbdled by setting the get-byteand read bytepointers of the slot byte queue to the second byte of the byte queue when the byte queue is created, then moving them back to the first byte when the count reaches 8. To avoid prolonged interrupt-masked operation, the critical section will be broken wherever possible by "windows". When the CCs string 'FFFF' is received, it is immediately acknowledged by a call to IP$FLOW$XMT:ACK.

2. Connection Layer Receiver Routine (IMXP$CONN:RCV)

Connection Layer Receiver is called by MUX Layer Receiver whenever a frame is available to be demultiplexed. It is implemented as nested state machines. The outermost state machine breaks up the frame into slots and associates each slot with a thread. Long and short form slots are decomposed into one byte input tokens to one of the two inner state machines. Supervisory slots are similarly decomposed to feed the other state machine.

The data slot state machine is responsible for call management and for the ISCC to CCS mapping. It also authorizes flow from the remote MXP. For connections other than leased line, call management is actuated by the state of DTR in the Modem ISCC. For contention connections, this may be caused at the remote end by a DTE responding to ringing on the RI pin. For auto-dial connections, it will be caused by application of power to the remote modem or terminal. TP dial will be supported as per the I/MATP design specification.

Flow control for received data streams is based on an authorization mechanism. It is a goal to try to keep the number of characters authorized for each thread at any given time greater than the number of characters of delay imposed by the physical link, so as to prevent the loss of effective bandwidth caused by forcing the remote port to wait for additional authorization. On the other hand, it is necessary to prevent a single VP from degrading port performance by committing more buffers than it can reasonably consume. Therefore, flow control strategy calls for allowing each VP to authorize a part of the free pool proportional to its speed relative to the sum of the speeds of the virtual ports, but limited to second link delay (to cover the worst case of 2-hop satellite link).

When FCA's are received, if there are any characters on the thread's slot flow-control overflow byte queue, characters are moved to the thread's slot hold byte queue or a slot-weight overflow byte queue until flow control authorization is again exhausted.

6.19.2.6 Call Manager Interface Submodule (IMXP$CMI:)

The Call Manager Interface is responsible for communication between the Connection Layer and the Call Manager. It consists of utility subroutines for creating and terminating calls, and an addressed packet handler.

The utility subroutines, IMXP$CMI:CRECALL and IMXP$CMI:HANGUP respectively build and send Create Call and Hangup addressed packets to the AP routing module.

The Call Addressed Packet Receiver Task IMXP$CMI:AP_RCV is started by IPOS Ø when an addressed packet is received for the thread. It dispatches on the following addressed packets:

1. Call Created

   Call Created packets cause CMI to initialize the thread and send a modem ISCC to the remote MXP.

2. Call Request

   Call Request addressed packets result in call acceptance for available lines in leased line ports if the calling node is the same as the configured partner. The same holds for auto-dial; however, if DTR is down, RI is raised five times for 3 seconds, with 6 seconds between rings. For contention ports, it does not check calling address against a configured value.

3. Call End

   Call End addressed packets cause the thread to be reset, and the port set to a waiting condition.

6.19.2.7 Statistics and Monitoring Submodule (IMXP$SM:)

This submodule is responsible for monitoring the performance of the I/MXP and reporting it to the Network Report Port as any exception condition arises. It works with the Statistics and Monitoring module (IPØ9$SM$) by providing I/MXP-specific monitoring and statistics-gathering functions.

Toward that end, the following statistics-gathering operations are embedded in the Protocol Handling Submodules within the I/MXP Protocol Module.

1. In the Line Transmitter Controller (IMXP$LINE:XMT)

   1. Number of bytes transmitted.
   2. Number of frames transmitted.

2. In the Line Receiver Controller (IMXP$LINE:RCV)

    1. Number of bytes received.
    2. Number of frames received.
    3. Number of frames received with bad FCS.

3. In the ARQ Layer Transmitter Task (IMXP$ARQ:XMT)

    1. Number of frames retransmitted.

4. In the Connection Layer Transmitter Routine (IMXP$CONN:XMT)

    1. User data bytes transmitted.

5. In the Connection Layer Receiver Routine (IMXP$CONN:RCV)

    1. Number of bytes received for each thread.

When called by IPØ9$SM$UPDATE task, IMXP$SM$MON computes the following based on the last monitoring period.

1. Error density (Receive Frame Error Rate) - The ratio of the number of frames received with bad FCS to the total number of frames received, multiplied by 100%.

2. Retransmit Frame Rate - The rate of frames being retransmitted in frames/sec.

3. Thread Compression Efficiency - The ratio of the number of input nibbles to the number of output nibbles of the adaptive data compression, multiplied by 100%. This is calculated for each active thread.

4. Thread Statistical Loading - The ratio of the number of bits received to the maximum number of bits which could be received, multiplied by 100%. This is calculated for each active thread.

5. Thread Compressed Loading - The ratio of the number of bits resulting from the adaptive data compression to the maximum number of bits that could be received (derived from the configured VP speed), multiplied by 100%. This is calculated for each active thread.

The results of the above calculations are compared to their respective thresholds which have been obtained as configuration parameters during the port initialization. If any threshold has been exceeded, a monitoring report addressed packet is constructed and sent to the Network Report Port. The monitoring of the Port Processor Loading and Buffer Utilization is done by IPØ9$SM$UPDATE task.

When called by IPØ9$SM$STAT:TASK, IMXP$SM$STAT appends the above statistics (1 and 2 for the port, and 3, 4, and 5 for a thread) as well as the following (for the port only) in the current statistics report addressed packet.

1. Statistical Loading - The ratio of the number of bits received to the maximum number of bits which could have been received by the 6854 receiver, multiplied by 100%.

2. Transmit Frame Rate - The rate of frames being transmitted in frames/sec.

3. Receive Frame Rate - The rate of frames being received in frames/sec.

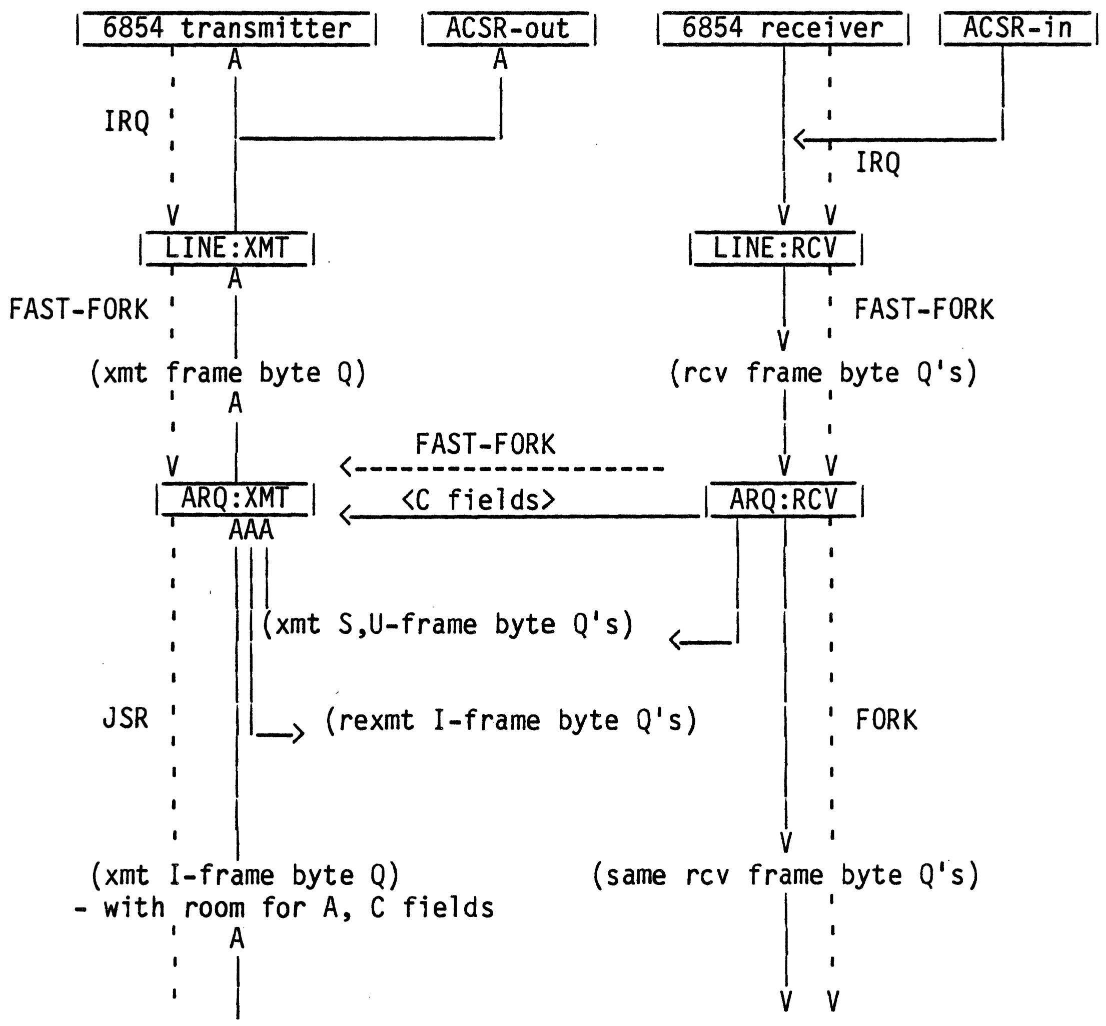
4. Traffic Density - The ratio of the number of user data bytes transmitted to the total number of bytes (user data bytes + overhead bytes) transmitted, multiplied by 100%.

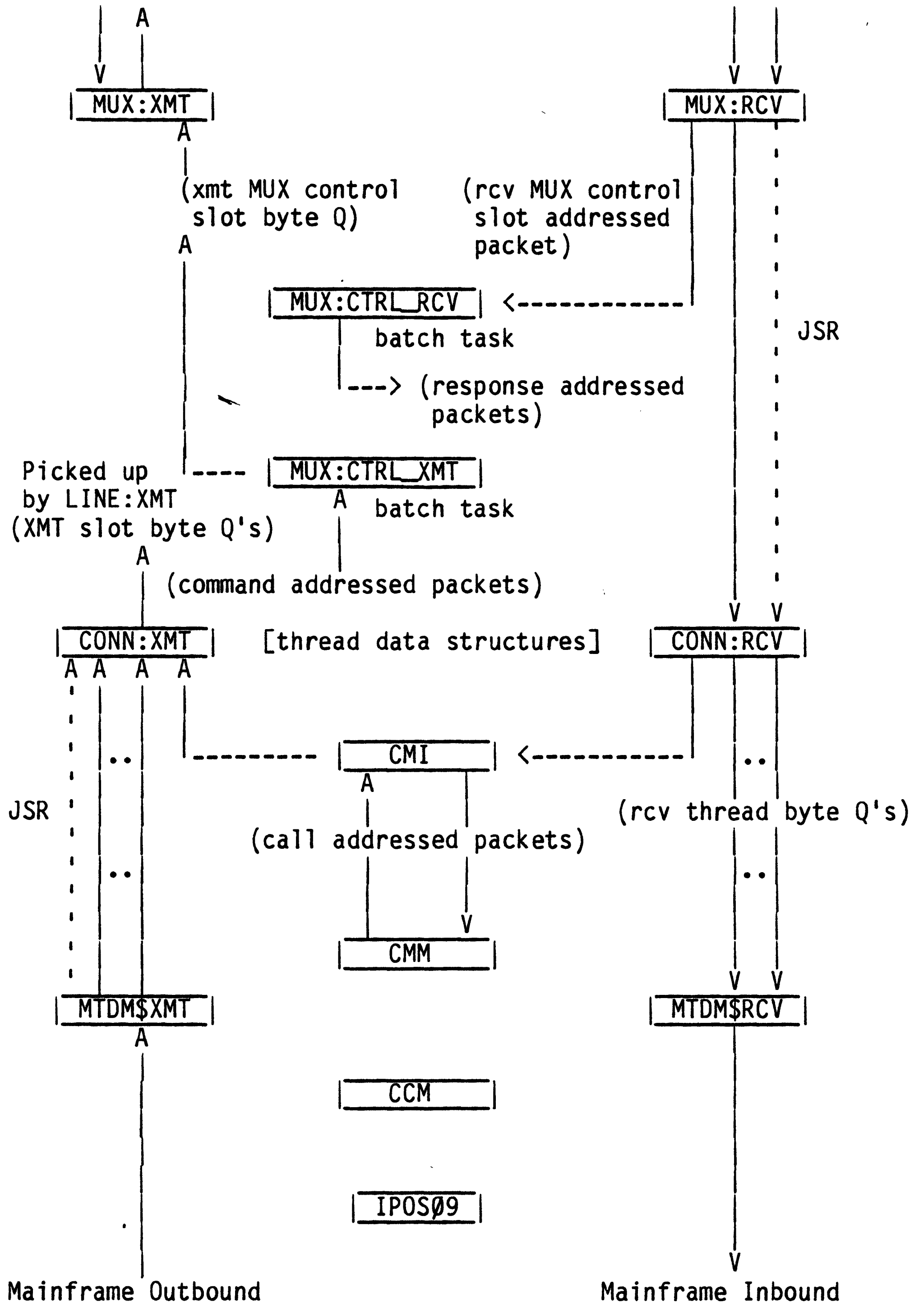
5. Retransmit Queue Size - The average number of frames in the retransmission queue.

Since there is no pre-allocated retransmission buffer space, the Retransmission Buffer Utilization as specified in the Codex Multiplex Protocol cannot be calculated. Instead, the statistics for the average Retransmit Queue Size has been added.

6.19.3 Overview of Data and Program Control Flow

Finally, a pictorial overview of the data flow and program control flow in an I/MXP is presented below.


6854 transmitter
ACSR-out
6854 receiver
ACSR-in
IRQ
IRQ
LINE:XMT
LINE:RCV
FAST-FORK
FAST-FORK
(xmt frame byte Q)
(rcv frame byte Q's)
FAST-FORK
ARQ:XMT
<C fields>
ARQ:RCV
(xmt S,U-frame byte Q's)
JSR
(rexmt I-frame byte Q's)
FORK
(xmt I-frame byte Q)
- with room for A, C fields
(same rcv frame byte Q's)


MUX:XMT
MUX:RCV
(xmt MUX control slot byte Q)
(rcv MUX control slot addressed packet)
MUX:CTRL_RCV
batch task
(response addressed packets)
JSR
Picked up by LINE:XMT (XMT slot byte Q's)
MUX:CTRL_XMT
batch task
(command addressed packets)
CONN:XMT
[thread data structures]
CONN:RCV
CMI
JSR
(rcv thread byte Q's)
(call addressed packets)
CMM
MTDM$XMT
MTDM$RCV
CCM
IPOSØ9
Mainframe Outbound
Mainframe Inbound