NASA Contractor Report 181874

Advanced Information Processing System: Input/Output System Services

Tom Masotto
Linda Alger

THE CHARLES STARK DRAPER LABORATORY, INC.
CAMBRIDGE, MA 02139

Contract NAS1-18565
August 1989

NASA
National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23665-5225

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1.0 INTRODUCTION

This purpose of this report is to document the functional requirements and detailed specifications for the I/O System Services of the Advanced Information Processing System (AIPS). This introductory section is provided to outline the overall architecture and functional requirements of the AIPS system. Section 1.1 gives a brief overview of the AIPS architecture as well as a detailed description of the AIPS fault tolerant network architecture, while Section 1.2 provides an introduction to the AIPS system software. Sections 2 and 3 describe the functional requirements and design and detailed specifications of the I/O User Interface and Communications Management modules of the I/O System Services, respectively. Section 4 illustrates the use of the I/O System Services, while Section 5 concludes with a summary of results and suggestions for future work in this area.

1.1 AIPS Architecture

The Advanced Information Processing System is designed to provide a fault- and damage-tolerant data processing architecture which can serve as the core avionics system for a broad range of aerospace vehicles being researched and developed by NASA. These vehicles include manned and unmanned space vehicles and platforms, deep space probes, commercial transports, and tactical military aircraft.

AIPS is a multicomputer architecture composed of hardware and software 'building blocks' that can be configured to meet a broad range of application requirements. The hardware building blocks are fault-tolerant, general purpose computers (GPCs), fault- and damage-tolerant inter-computer and input/output networks, and interfaces between the networks and the general purpose computers. The software building blocks are the major software functions: local system services, input/output system services, inter-computer system services and the system manager. This software provides the services necessary in a traditional real-time computer such as task scheduling and dispatching, communication with sensors and actuators, etc. The software also supplies the redundancy management services necessary in a redundant computer and the services necessary in a distributed system such as inter-function communication across processing sites, management of distributed redundancy, management of networks, and migration of functions between processing sites.

The AIPS hardware consists of a number of computers located at processing sites which may be physically dispersed throughout a vehicle. These processing sites are linked together by a reliable, damage-tolerant data communication pathway called the Inter-Computer (IC) bus. Since the hardware implementation of this "virtual bus" is a circuit-switched network, but from the GPC communication and protocol viewpoint it appears as a conventional bus, the terms "bus" and "network" are used interchangeable throughout this document. A computer at any particular processing site may also have access to varying numbers and types of Input/Output (I/O) buses, which are separate from the IC bus. The I/O buses may be global, regional or local in nature. I/O devices on the global I/O bus are available to all, or at least a majority, of the AIPS computers. Regional buses connect I/O devices in a given region to the processing sites located in their vicinity. Local buses connect a computer to the I/O devices dedicated to that computer. Additionally, I/O devices may be connected directly to the internal bus of a processor and accessed as though the I/O devices reside in the computer memory (memory mapped I/O). Both the I/O buses and the IC bus are time-division multiple-access contention buses. Figure 1 shows the laboratory engineering model for a distributed AIPS configuration. This distributed AIPS configuration includes all the hardware and software building blocks mentioned earlier and was conceived to demonstrate the feasibility of the AIPS architecture.

The laboratory configuration of the distributed AIPS system shown in Figure 1 consists of four processing sites. Each processing site has a General Purpose Computer (GPC). GPCs may be simplex or they may be FTPs of varying redundancy levels. Of the four FTPs in the laboratory configuration, one is simplex, one is duplex, and two are triplex processors. An FTP may also be quadruply redundant but none was fabricated for the AIPS laboratory demonstration. The redundant FTPs are built such that they can be physically dispersed for damage tolerance; each of the redundant channels of an FTP can be as far as 5 meters from other channels of the same FTP.

The GPCs are interconnected by a triplex circuit-switched network. Each network layer forms a full two way 'virtual bus'. The three layers are totally independent and are not cross-strapped to each other. Each layer contains a circuit-switched node for each processing site; thus every processing site is serviced by three nodes of the IC network. GPCs are designed to receive data on all three layers, but the capability of a GPC to transmit on the network depends on the GPC redundancy level. Triplex FTPs can transmit on all three layers, duplex FTPs on only two of the three layers, and simplex processors on only a single layer. In duplex and triplex FTPs, a given processor can transmit on only one network layer. Thus malicious behavior of a processor can disrupt only one layer.

The IC network and the GPC interfaces into the network are designed in strict accordance with fault-tolerant systems theory. Thus an arbitrary random hardware fault, including Byzantine faults, anywhere in the system can not disrupt communication between triplex FTPs. In other words, the triplex IC network, in conjunction with the GPC interfaces into the network, provides error-masking capability for communication between two triplex computers.

The laboratory demonstration of the I/O network is implemented using a 15-node circuit-switched network that interfaces with each of the GPCs on 1 to 6 nodes, depending on the GPC redundancy level. The 15 I/O nodes can be configured in the laboratory as global, regional, and local I/O networks to demonstrate various dimensions of the AIPS I/O concept.

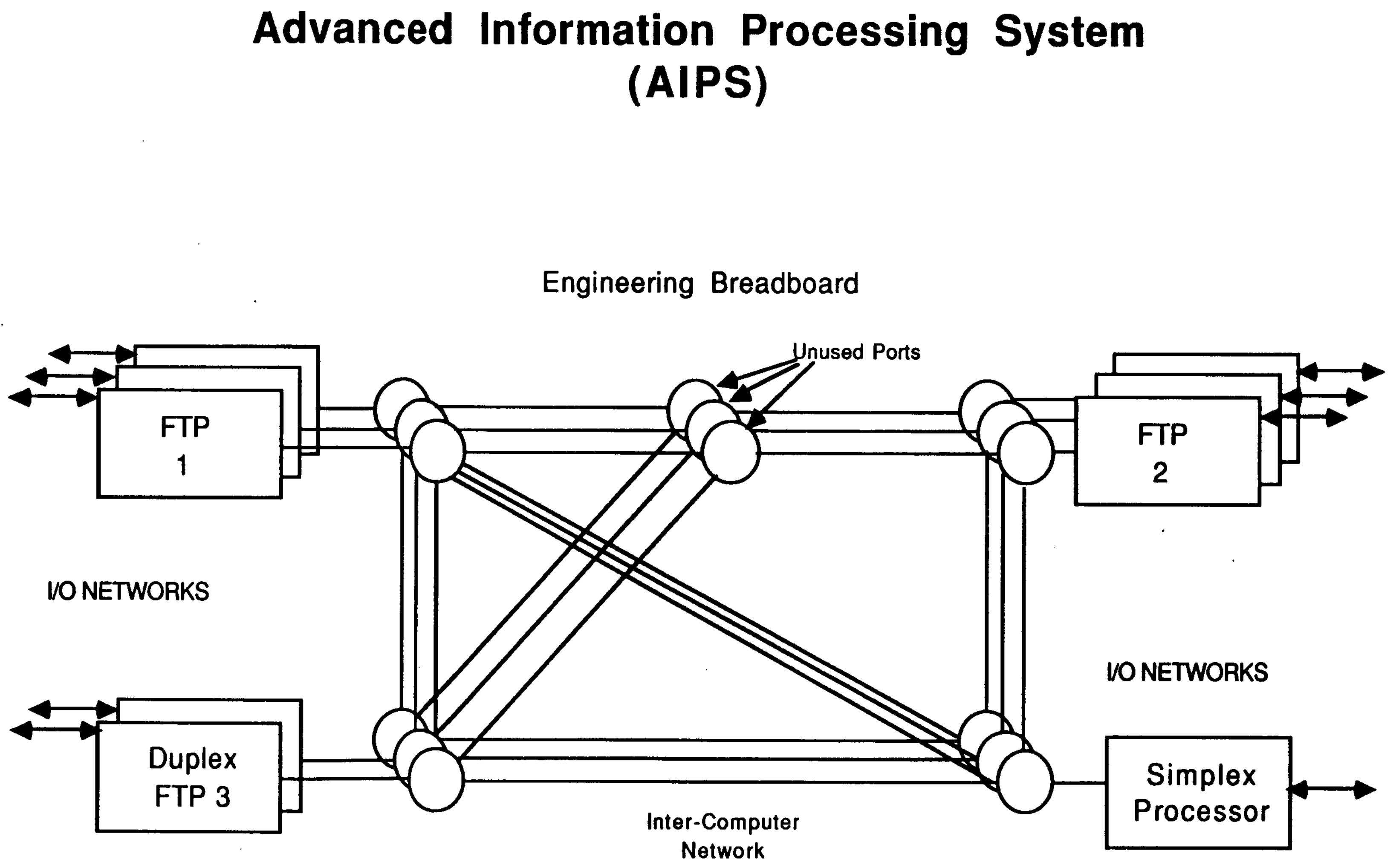


Figure 1. AIPS Distributed Configuration

1.1.1 AIPS Networks

For communication between GPCs and between a GPC and I/O devices, a damage and fault tolerant network is employed. The network consists of a number of full duplex links that are interconnected by circuit switched nodes. In steady state, the circuit switched nodes route information along a fixed communication path, or 'virtual bus', within the network, without the delays which are associated with packet switched networks. Once the virtual bus is set up within the network the protocols and operation of the network are similar to typical multiplex buses. Every transmission by any subscriber on a node is heard by all the subscribers on all the nodes just as if they were all linked together by a linear bus. Although the network performs exactly as a bus, it is far more reliable and damage tolerant than a linear bus. A single fault or limited damage can disable only a small fraction of the virtual bus, typically a node or a link connecting two nodes. Such an event does not disable the network, as would be the case for a linear bus. The network is able to tolerate such faults due to the richness of interconnections between nodes. By reconfiguring the network around the faulty element, a new virtual bus is constructed. Except for such reconfigurations, the structure of the virtual bus remains static.

The nodes are sufficiently intelligent to recognize reconfiguration commands from the network manager, which is resident in one of the GPCs. The network manager performs the necessary diagnostics to identify the failed element and can change the bus topology by sending appropriate reconfiguration commands to the affected nodes.

Damage caused by weapons or electrical shorts, overheating, or localized fire would affect only subscribers in the damaged portion of the vehicle. The rest of the network, and the subscribers on it, can continue to operate normally. If the sensors and effectors are themselves physically dispersed for damage tolerance, and the damage event does not affect the inherent capability of the vehicle to continue to fly, then the digital system would continue to function in a normal manner or in some degraded mode as determined by sensor/effector availability.

Fault isolation is much easier in the network than in multiplex buses. For example, a remote terminal transmitting out of turn, a rather common failure mode which will totally disable a linear bus, can be easily isolated in the network through a systematic search where one terminal is disabled at a time. Furthermore, for networks of moderate size, up to 50 nodes, most faults can be detected, isolated and the network reconfigured in milliseconds.

The network can be expanded very easily by linking the additional nodes to the spare ports in existing nodes. In fact, nodes and subscribers to the new nodes (I/O devices or GPCs) can even be added without shutting down the existing network. In bus systems, power to buses must be turned off before new subscribers or remote terminals can be added.

Finally, there are no topological constraints, as are encountered with linear or ring buses.

In fact, these are simply subsets of the fault-tolerant network architecture.

1.2 AIPS System Software

The AIPS system software, as well as the hardware, has been designed to provide a virtual machine architecture that hides hardware redundancy, hardware faults, multiplicity of resources, and distributed system characteristics from the applications programmer. Section 1.2.1 discusses the approach used for the AIPS system software design. Section 1.2.2 is a high level description of the system services that are provided for AIPS users.

1.2.1 AIPS Software Design Approach

The approach used to design the AIPS system software is part of the overall AIPS system design methodology. An abbreviated form of this system design methodology is shown in Figure 2. This methodology began with the application requirements and eventually led to a set of architectural specifications. The architecture was then partitioned into hardware and software functional requirements. This report documents the design approach used for I/O System Services software, beginning with the functional requirements and proceeding through detailed specifications.

Hardware and software for the AIPS architecture is being designed and implemented in two phases. The first phase is the centralized AIPS configuration. The centralized AIPS architecture, as shown in Figure 3, is configured as one triplex Fault Tolerant Processor (FTP), an Input/Output network and the interfaces between the FTP and the network, referred to as input/output sequencers (IOS). The laboratory demonstration of the input/output network consists of 15 circuit-switched nodes which can be configured as multiple local I/O networks connected to the triplex GPC. For example, the I/O network may be configured as one 15-node network, as shown in Figure 3, or as three 5-node networks. The software building blocks that have been designed and implemented for the AIPS centralized architecture include local system services and I/O system services. The following subsection 1.2.2 gives an overview of all the AIPS software building blocks. The rest of this document , Sections 2 through 4, focuses on the functional design and detailed specification of the I/O System Services.

1.2.2 AIPS System Software Overview

As shown in Figure 4, AIPS system software provides the following AIPS System Services: local system services, communication services, system management, and I/O system services. The system software is being developed in Ada. System services are modular and partitioned naturally according to hardware building blocks. The distributed AIPS configuration includes all the services. Versions of the system software for specific applications can be created by deleting unused services from this superset. The System Manager functions reside on only one GPC, but all functions of the System Manager are

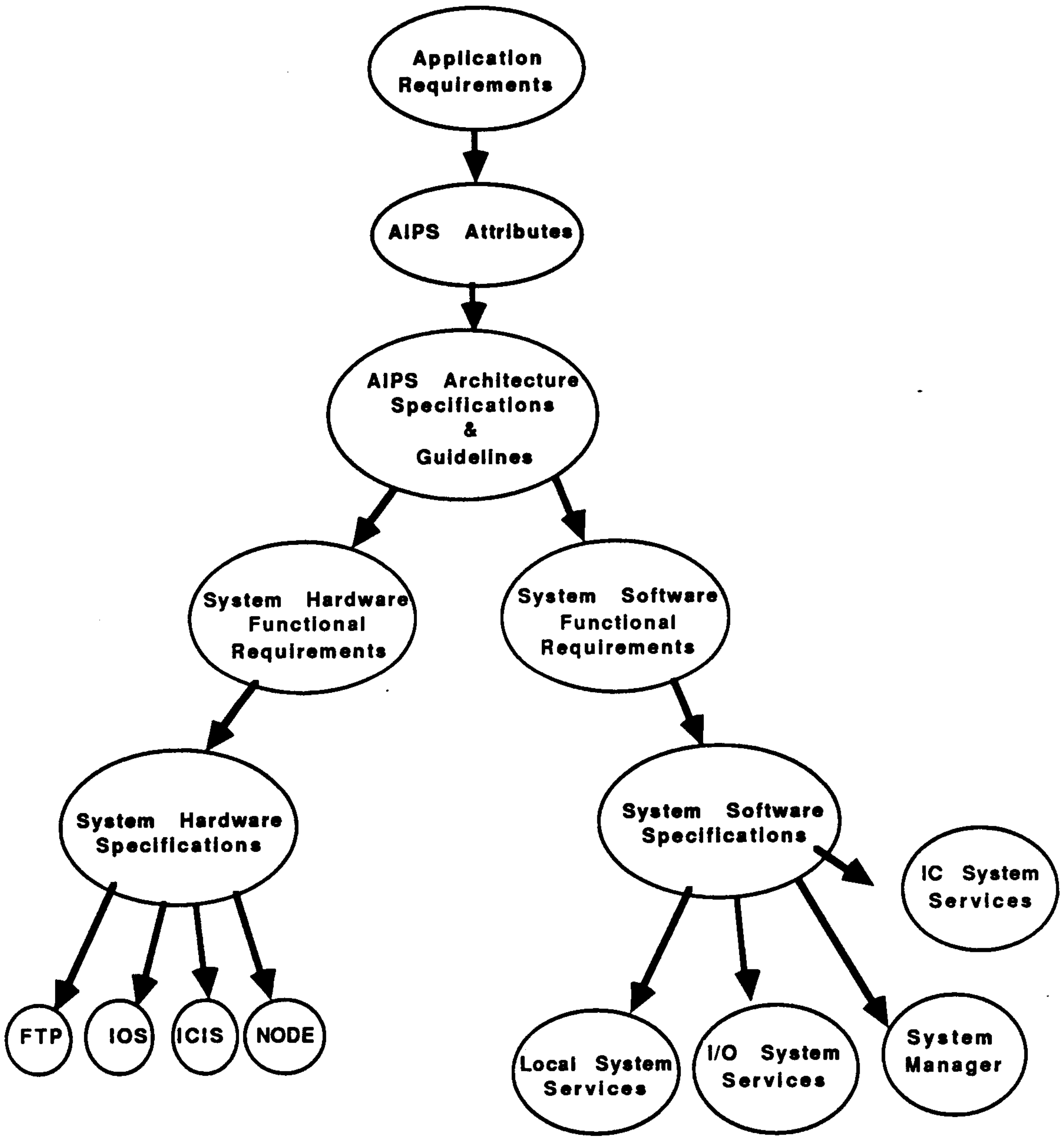

Application Requirements
AIPS Attributes
AIPS Architecture Specifications & Guidelines
System Hardware Functional Requirements
System Software Functional Requirements
System Hardware Specifications
System Software Specifications
IC System Services
FTP
IOS
ICIS
NODE
Local System Services
I/O System Services
System Manager

Figure 2. AIPS System Design Approach

not necessarily on the same GPC. The other system services are replicated in each GPC. The following is a brief description of each of the services.

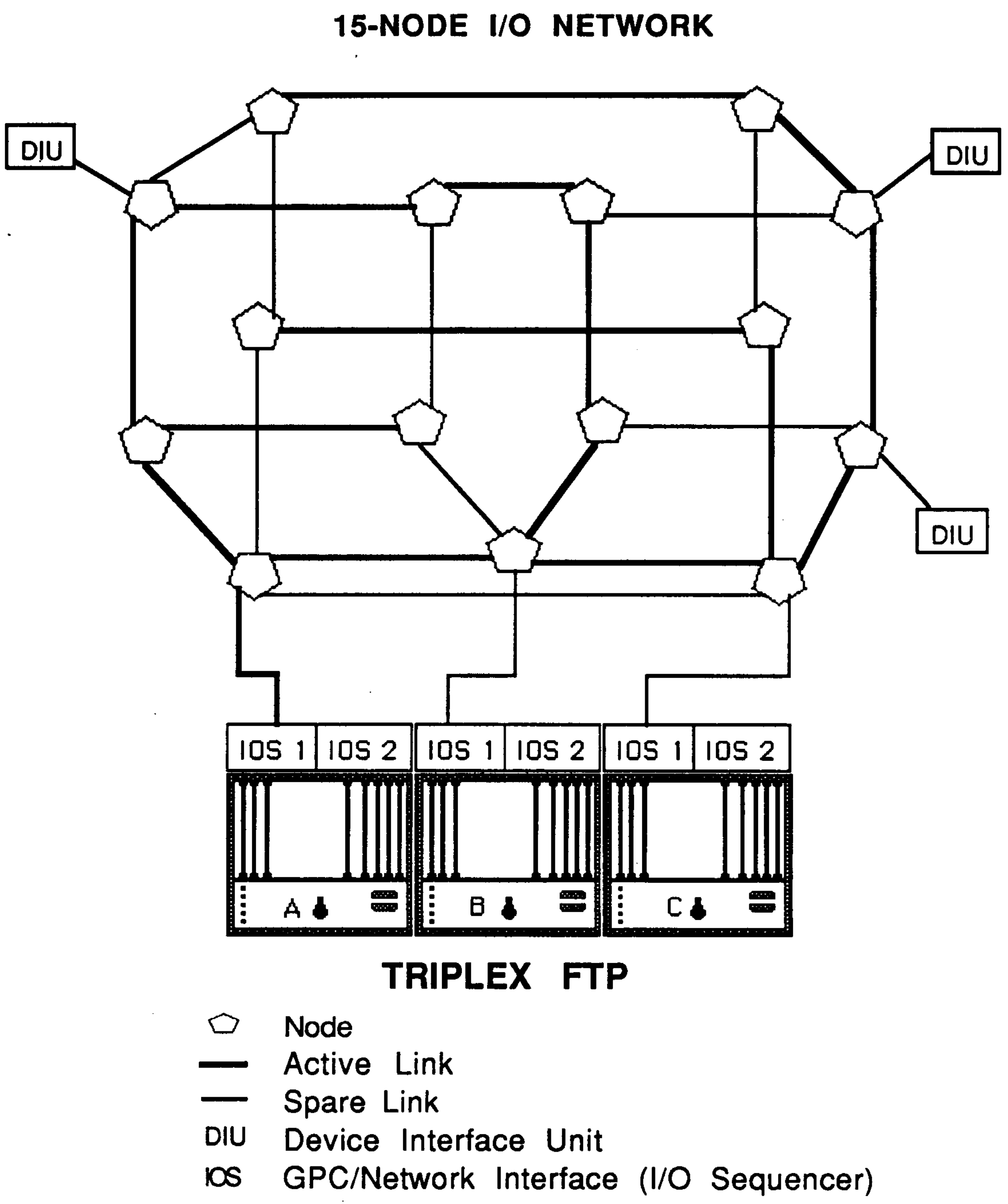


Figure 3. Centralized AIPS Configuration

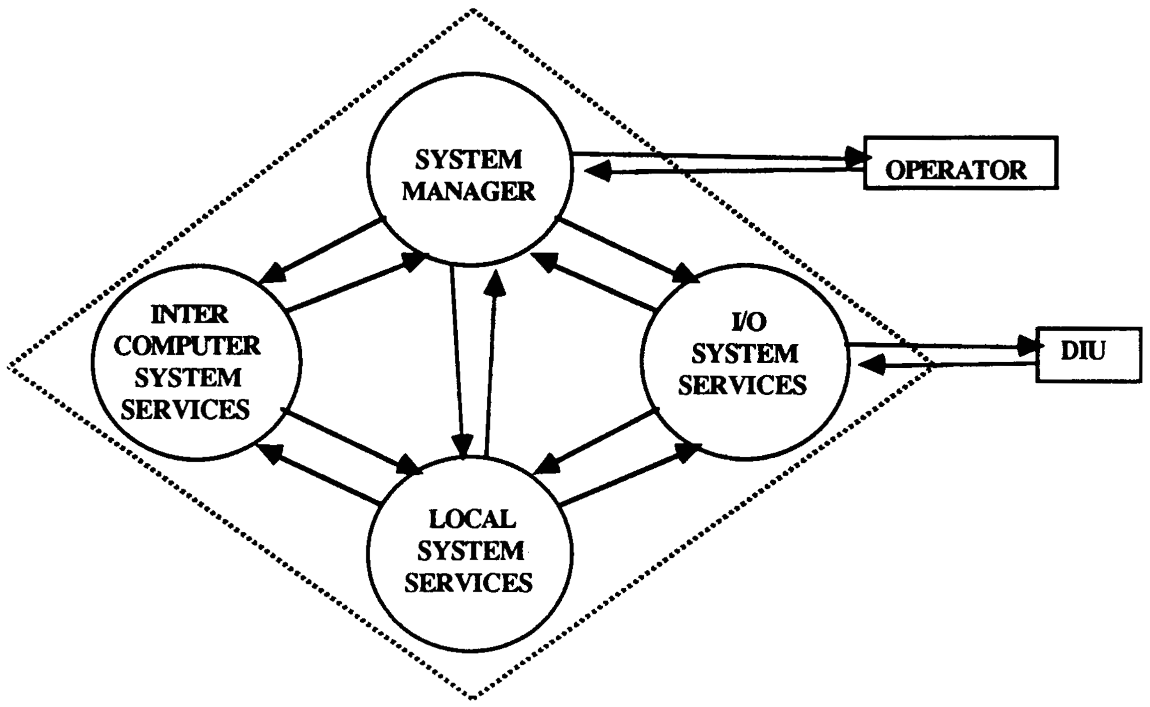


Figure 4. Top Level View Of System Services

1.2.2.1 Local System Services

The local system services provided in each GPC are: GPC initialization, real-time operating system, local resource allocation, local GPC Fault Detection, Isolation, and Reconfiguration (FDIR), GPC status reporting, and local time management (see Figure 5).

The function of GPC initialization is to bring the GPC to a known and operational state from an unknown condition (cold start). Each channel of a GPC has two processors: a computational processor (CP) and an input/output processor (IOP). GPC initialization synchronizes the CPs with each other, synchronizes the IOPs with each other, and resets or initializes the GPC hardware and interfaces (interval timers, real time clock, interface sequencers, DUART, etc.) It makes the hardware state of the redundant channels congruent by alignment of memory and control registers. It then activates the system baseline software that is common to every GPC.

The AIPS real-time operating system supports task execution management, including scheduling according to priority, time and event occurrence, and is responsible for task dispatching, suspension and termination. It also supports memory management, software exception handling, and intertask communication between companion processors (IOP and CP). The AIPS operating system resides on every CP and IOP in the system. It uses the vendor-supplied Ada Run Time System (RTS), and includes additional features required for the AIPS real-time distributed operating system.

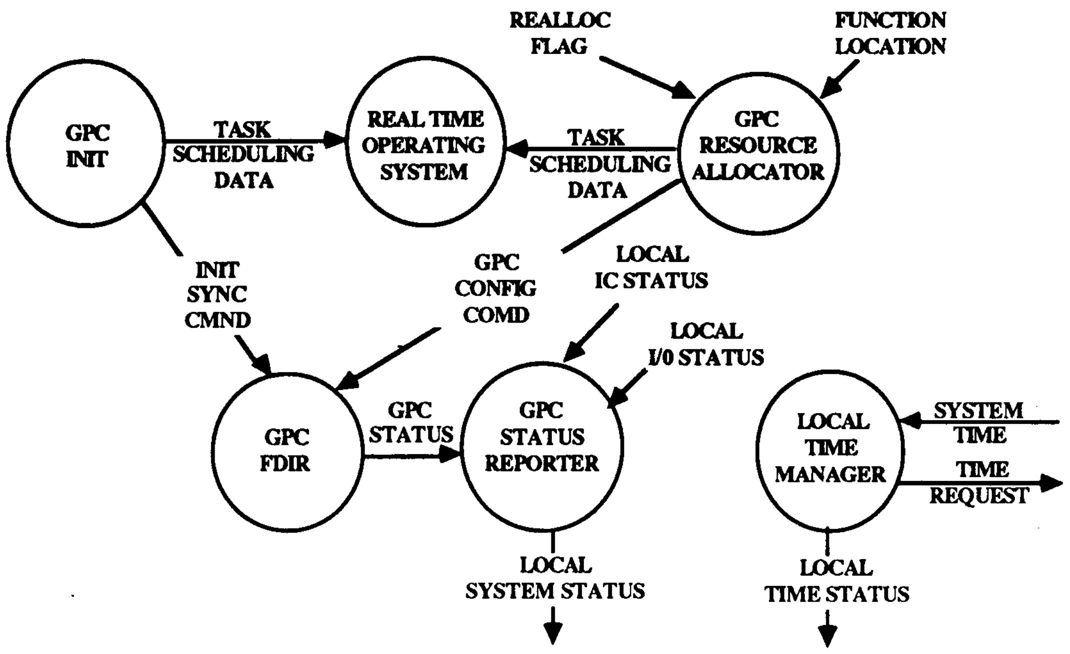


Figure 5. Local System Services

The GPC resource allocator coordinates and determines responsibility for any global or migratable functions from the system resource manager. It also monitors commands from the system resource manager to start or stop any function.

The GPC status reporter collects status information from the local functions, the local GPC FDIR, the IC system services and the I/O system services. It updates its local data base and disseminates this status information to the system manager.

GPC FDIR has the responsibility for detecting and isolating hardware faults in the CPs, IOPs, and shared hardware. It is responsible for synchronizing both groups of processors in the redundant channels of the FTP and for disabling outputs of failed channel(s) through interlock hardware. After synchronization, all CPs will be executing the same machine language instruction within a bounded skew, and all IOPs will be executing the same machine language instruction within a bounded skew. GPC FDIR logs all faults and reports status to the GPC status reporter. It is responsible for the CPU hardware exception handling and downmoding/upmoding hardware in response to configuration commands from the system manager. It is also responsible for transient hardware fault detection and for running low priority self tests to detect latent faults. This redundancy management function is transparent to the application programmer.

The local time manager works in cooperation with the system time manager to keep the local real time initialized and consistent with the universal time. It is also responsible for providing time services to all users. A detailed description of the Local System Services in provided in [1].

1.2.2.2 Inter-Computer System Services

The inter-computer system services provide two functions: (1) inter-computer user communication services, that is, communication between functions not located in the same GPC, and (2) inter-computer network management (Figure 6).

The IC user communication service provides local and distributed inter-function communication which is transparent to the application user. It provides synchronous and asynchronous communication, performs error detection and source congruency on inputs, and records and reports IC communication errors to IC network managers. Inter-computer communication can be done in either point to point or broadcast mode and is implemented in each GPC.

The IC network manager is responsible for the fault detection, isolation and reconfiguration of the network. The AIPS distributed configuration consists of three identical, independent IC network layers which operate in parallel to dynamically mask faults in a single layer and provide reliable communication. There is one network manager for each network layer. However, the three network layer managers do not need to reside in the same GPC. They are responsible for detecting and isolating hardware faults in IC nodes and links and for reconfiguring their respective network layer around any failed elements. The network manager function is transparent to all application users of the network.

1.2.2.3 System Manager

The system manager is a collection of system level services including the applications monitor, the system resource manager, the system fault detection, isolation and reconfiguration (FDIR), and the system time manager (Figure 7).

The applications monitor interfaces with the applications programs and the AIPS system operator. It accepts commands to migrate functions from one GPC to another, to display system status, to change the state of the system by requesting a hardware element state change, and to convey requests for desired hardware and software configurations to the system resource manager.

The system resource manager allocates migratable functions to GPCs. This involves the monitoring of the various triggers for function migration such as failure or repair of hardware components, mission phase or workload change, operator or crew requests and

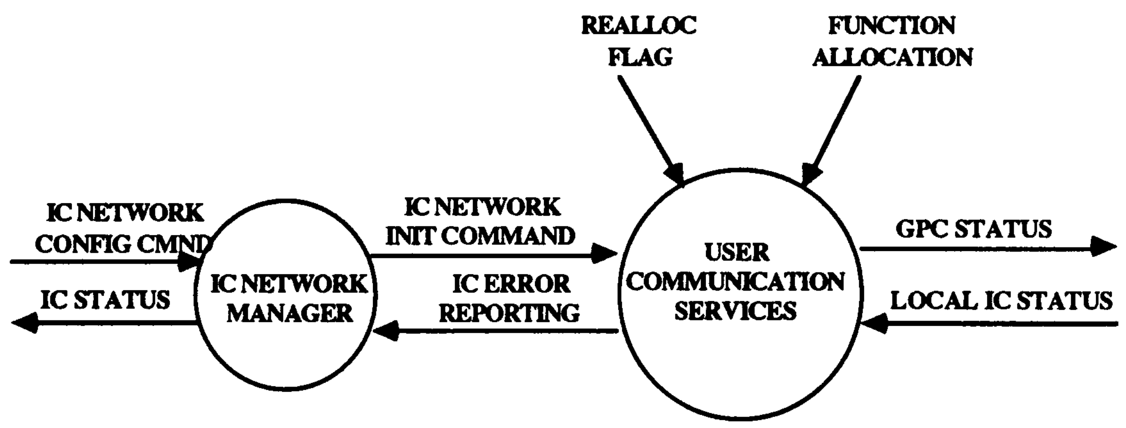


Figure 6. Inter-Computer System Services

timed events. It reallocates functions in response to any of these events. It also designates managers for shared resources and sets up the task location data base in each GPC.

The system fault detection, isolation and reconfiguration (FDIR) is responsible for the collection of status from the inter-computer (IC) network managers, the I/O network managers, and the local GPC redundancy managers. It resolves conflicting local fault isolation decisions, isolates unresolved faults, correlates transient faults, and handles processing site failures.

The system time manager, in conjunction with the local time manager on each GPC, has the job of maintaining a consistent time across all GPCs. The system time manager indicates to the local time manager when to set its value of time. It also sends a periodic signal to enable the local time manager to adjust its time to maintain consistency with an external time source such as the GPS Satellites or an internal source such as the real time clock in the GPC which hosts the system time manager software.

1.2.2.4 I/O System Services

The I/O system services provide efficient and reliable communication between the user and external devices (sensors and actuators). The I/O system services software is also responsible for the fault detection, isolation and reconfiguration of the I/O network hardware and GPC/network interface hardware (input/output sequencers).

I/O system services is made up of three functional modules: I/O user interface, I/O communication management and the I/O network manager (Figure 8).

The I/O user interface provides a user with read/write access to I/O devices or device interface units (DIUs), such that the devices appear to be memory mapped. It also gives the

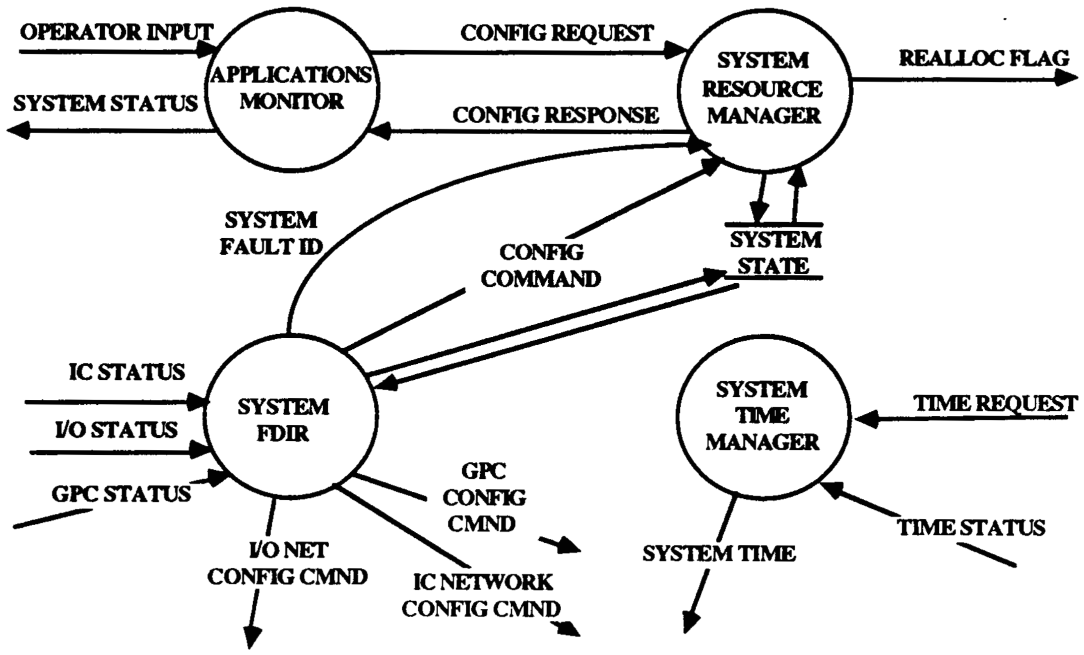


Figure 7. System Manager

user the ability to group I/O transactions into chains and I/O requests, and to schedule I/O requests either as periodic tasks or on demand tasks.

The I/O communication manager provides the functions necessary to control the flow of data between a GPC and the various I/O networks used by the GPC. It also performs source congruency and error detection on inputs, voting on all outputs, and reports communication errors to the I/O network manager. It is also responsible for the management of the I/O request queues.

The I/O network manager is responsible for detecting and isolating hardware faults in I/O nodes, links, and interfaces and for reconfiguring the network around any failed elements. The network manager function is transparent to all application users of the network.

The I/O user interface, I/O communications management, and I/O redundancy management modules are dependent processes, as illustrated in Figure 8. The I/O communication management process uses the database of I/O request specifications (I/O request database) that is constructed by the I/O user interface. In addition, the I/O user interface and communications management modules interact when communicating the I/O data,

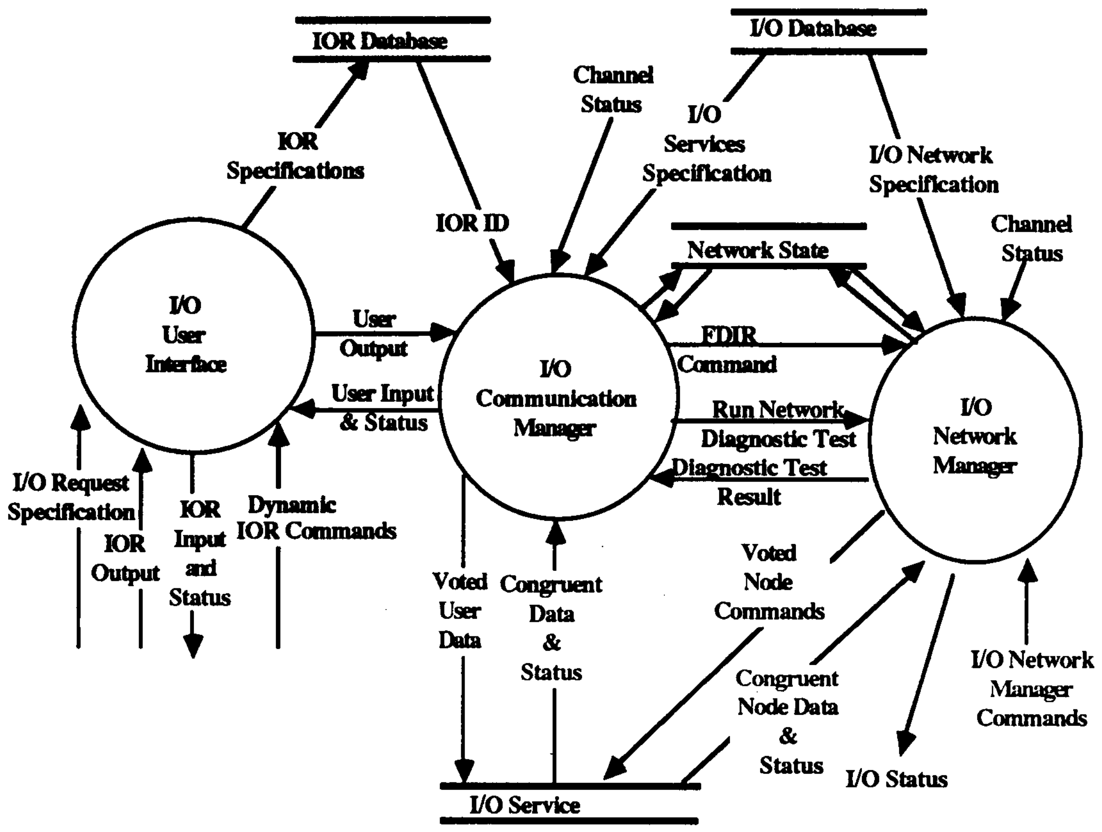


Figure 8. I/O System Services

synchronizing the CP and IOP tasks, and processing of the I/O requests. The I/O communications management process interacts with the I/O redundancy management module for the communication of network status, diagnostic information, and FDIR commands. Furthermore, the communication management and redundancy management modules both use the I/O database and I/O low level utilities.

Sections 2 and 3 describe the functional requirements and design and detailed specifications of the I/O User Interface and I/O Communications Manager, respectively. The software requirements and specifications for the I/O Network Manager are described in [2]. Section 4 illustrates the use of the I/O System Services for the applications programmer, and Section 5 concludes with a summary of results.

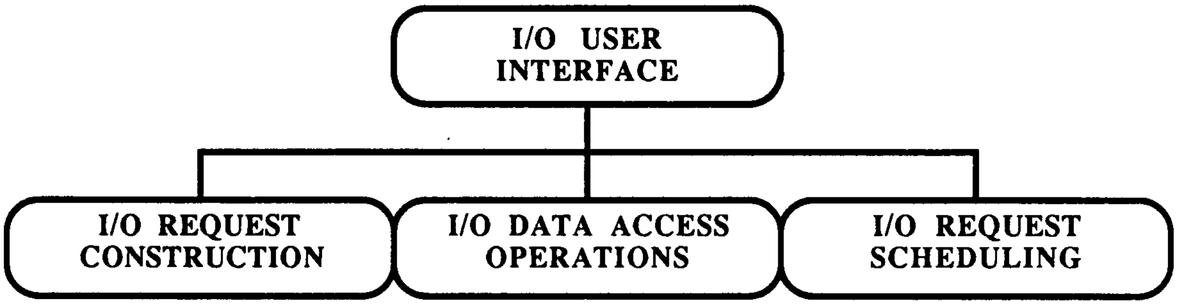
2.0 I/O USER INTERFACE

The I/O User Interface provides a user with access to I/O devices or device interface units (DIUs). It provides this access to the DIUs in such a way that to the applications user they appear to be memory mapped. That is, each DIU with which the FTP interfaces can be simply addressed by means of read/write routines that simulate memory mapped I/O to the user. It also provides the user with the option of either single or chained transactions on an I/O network (a transaction is an HDLC frame sent to a single DIU using the HDLC protocol; a chain or chained transactions is an ordered set of one or more transactions addressed to devices on one I/O network). The use of chained transactions allows very efficient use of the network bandwidth. Redundant chains can be executed in a (nearly) simultaneous fashion on redundant I/O networks to provide data to and from redundant devices with a bounded time skew. I/O activity may be scheduled to run periodically or on demand. The I/O User Interface provides the means to form I/O requests from single or chained transactions and to schedule I/O requests (an I/O request is a set of one or more single or chained I/O transactions, each of which executes on a different I/O network). These I/O request specifications result in CP/IOP shared memory assignments for data and error information for each DIU transaction. In addition, the I/O User Interface provides system calls for safely accessing those memory mapped locations. Although DIUs are connected to a fault tolerant network, all network access protocols, source congruency and error processing on inputs, and fault masking on outputs are transparent to the user.

The description of the I/O User Interface is divided into three sections: functional description, software specifications, and software process descriptions.

2.1 I/O User Interface Functional Description

The I/O User Interface is divided into three functions: I/O Request Construction, I/O Data Access Operations, and I/O Request Scheduling.



The I/O Request Construction function allows the user to create I/O transactions, specify how they will be grouped and how each I/O request will be scheduled. The I/O Data Access Operations provide the read/write routines that allow the user to access I/O chain data in shared memory, while hiding the CP/IOP protocol from the user. The I/O Request Scheduling provides the user with the flexibility to schedule each I/O request as a cyclic free running task that runs and signals the caller when each cycle has completed or an on-demand task that is only scheduled when requested.

2.1.1 I/O Request Construction

The applications user can construct transactions, chains, and I/O requests in an hierarchical manner. Initially, the parameters associated with each transaction must be specified. Secondly, the transactions that are sequentially executed as a unit on one network are grouped to form an I/O chain. Finally, the chains are grouped into I/O requests to fully maximize the bandwidth of the communications network by allowing the simultaneous execution of chains on the parallel networks of an I/O service. The applications user can construct one or more I/O requests for each I/O service (an I/O service is a logical organization imposed on I/O network use) as dictated by the requirements of the application.

The I/O User Interface requires several parameters to be specified in order to create an I/O transaction. The user has to specify whether the transaction will request information from or send a command to a DIU (input and output transaction respectively). In either case, the DIU must be specified. In addition, the user must provide the number of data bytes to be sent to the DIU (all transactions) and the number of data bytes that will be returned by the DIU (all input transactions). Accordingly, the location(s) of the appropriate data buffer(s) on the CP must be specified in order to read/write the data associated with the DIU. The user must also specify whether the output data is dynamic or static. If the data is dynamic, then it is copied into the IOS prior to each execution of the associated chain. If the data is static, then it is copied into the IOS only once. For input transactions, the user must specify the maximum number of errors that are tolerable before the transaction is bypassed (deselected by the I/O System Services) and the transaction time-out which is the maximum time that can expire before a byte of data is received from the DIU. The I/O User Interface constructs a record using this information and returns a transaction identifier (ID) to allow the user to later identify the transaction.

After the transactions are specified and created, they are grouped to form chains. Chains allow efficient use of the communications bandwidth, but are only applicable to a single network. Accordingly, the user must specify the transactions that will form the chain, and the network on which they will be executed. The I/O User Interface records the chain information and returns a chain identifier to the user.

After the chains are specified and created, they are grouped into I/O requests. An I/O request is a set of one or more chains each of which simultaneously executes on a parallel network of an I/O service. The creation of the I/O request requires the user to specify the chains that will form the request. In addition, the user must provide the I/O request time out which is the maximum length of time that the I/O request requires the I/O network (used for I/O request execution and processing). Furthermore, the desired scheduling requirements must be provided. This scheduling information specifies if the I/O request is periodic or on demand, the priority of the request, and the frequency with which I/O request uses an event to signify its completion. If the I/O request is periodic, the scheduling requirements also specify the repetition period, how it is started (after_delay, on_event, or at_absolute_time), and how it will stop (never_stop or stop_on_event). The I/O User Interface records the I/O request information and returns an I/O request identifier to the user.

In order to make the redundancy management, source congruency, multiprocessing (CP/IOP) communications protocol, and fault masking transparent to the user, the I/O specifications (transaction, chain, and I/O request) must be communicated through shared memory to the IOP. The IOP uses the information to construct a set of companion records. The companion records are transaction, chain, or I/O request specification records that are the IOP duals of CP transaction, chain, or I/O request records. The companion records are used when processing the I/O requests. A CP/IOP handshaking protocol is incorporated to insure that the I/O request specifications and associated data are not corrupted during their transmission through shared memory.

2.1.2 I/O Data Access Operations

The I/O Data Access Operations provide read/write routines that allow the user to access I/O data in shared memory that appears to the user to be memory mapped. The redundancy management of the fault-tolerant network, network access protocols, source congruency and error processing on the inputs, fault masking on the outputs, and CP/IOP communications protocol are transparent to the user. The applications user is able to write commands to and request information from the DIUs. In addition, select and deselect system calls are available allowing the user to add and remove transactions from their corresponding chains (enabling error recovery and dynamic I/O request reconfiguration). The I/O Data Access routines also allow the user to determine whether or not errors occurred in the I/O requests and to isolate the location of the error(s) if one (or more) resulted.

The user can send control data to the DIUs using the I/O User Interface write procedures. Command information (output data) can be sent a DIU by writing data to the transactions that communicate with the DIU. The data can be modified on a transaction by transaction, chain by chain, or request by request basis. The CP must write the data into shared memory to communicate it to the IOP. The IOP reads the data and writes it into the dual ported memory of the IOS prior to executing the I/O request. Since the IOP may attempt to read the data from shared memory while the CP may be modifying it, semaphores and double buffering are used to maintain consistent data sets. Since a chain is executed as a unit, the data associated with the chain must also be considered a unit. Accordingly, it is not sufficient that the semaphores simply lock access to the transactions. The semaphores must, at least, control the access to the I/O chain data.

The user can read the information returned by the DIUs (input data) using the I/O User Interface read procedures. Accordingly, the user can obtain response data from the DIUs, process the data, and generate output commands based upon the results. The data can be read on a transaction by transaction, chain by chain, or request by request basis. As with the write procedures, semaphores and double buffering are used to maintain consistent data throughout the chains. The data returned by the DIUs may be corrupted by errors. The I/O User Interface returns a parameter that notifies the user of an occurrence of an error. In addition, the data returned to the user may not have been updated since the previous read procedure ("old data"). The occurrence of "old data" signifies that a scheduling overrun has occurred except when the I/O request and its associated applications task are not synchronized (i.e. both are free running cyclic tasks). The I/O User Interface allows the user to check whether or not the data has been modified since the data was last read.

The occurrence of errors in the network may cause a DIU to become inaccessible. As a result, a transaction requesting data from that DIU would not contain any valid information. The ability to deselect a transaction allows the user to remove any undesired transactions from a chain. The deselection command is communicated to the IOP through shared memory. Since the IOP may be executing an I/O request when the deselection procedure is called, the CP may not be able to immediately send the command. The application tasks should not have to wait for the deselect command to be accepted by the IOP. As a result, the deselect procedure returns a parameter specifying whether or not the deselection request was accepted.

The restoration of failed elements of the network may cause a previously inaccessible DIUs to become reachable. The select procedure allows the user to include previously deselected (or bypassed) transactions in the I/O chains. As with the deselect procedure, the I/O User Interface returns a parameter specifying whether or not the selection request was accepted.

As previously mentioned, the data returned by the DIUs may be corrupted by errors. The user is notified of the occurrence of errors by a parameter returned by the read procedure call. The I/O User Interface allows the user to determine if the error was in a chain (chain time out) or a transaction of a chain. The user can isolate the location of an error and disregard the corrupted data.

The creation of an I/O request requires the specification of its scheduling requirements (see Section 2.1.1 - I/O Request Construction). The I/O request is scheduled on the IOP based on these specifications and is executed when the requirements are met. If the I/O request is not able to executed when the scheduling requirements have been fulfilled, a scheduling overrun occurs. The I/O User Interface allows the user to check whether or not an overrun has occurred.

2.1.3 I/O Request Scheduling

The I/O Request Scheduling provides the application with the flexibility to schedule each I/O request as a cyclic free running task that runs and signals the caller when complete or an on demand task that is only scheduled when requested. The user specifies the scheduling requirements for the I/O requests when the I/O requests are created (see Section 2.1.1 - I/O Request Construction).

Each I/O request may correspond to one or more application tasks defined by the user. Accordingly, the I/O User Interface must provide synchronization mechanisms to coordinate the I/O requests with the application tasks. The synchronization mechanisms provided by the I/O System Services are events and flags. Events are signals which are observed by the GPC Real Time Operating System. The events interrupt the co-processor and are used to activate/deactivate a task on the co-processor. Flags are passive signals which may be observed or ignored by the application tasks. The flags do not interrupt the co-processor and are used to indicate the completion of the I/O requests.

The completion of an I/O request is indicated by a flag in shared memory. The I/O System Services on the IOP sets the flag when the I/O request completes whether or not errors occurred. The application tasks on the CP can read and clear the flag.

The completion of an I/O request is indicated by an event if the user has specified this option when creating the I/O request. The I/O System Services on the IOP signals the event when the I/O request completes whether or not errors occurred. These events are used to activate application tasks (via the GPC Real Time Operating System) that are blocked waiting for the completion of an I/O request. The I/O User Interface provides a mechanism for an application task to obtain a pointer to an event, allowing the task to be scheduled as an event-driven process.

On demand I/O requests are started on the IOP only when the user issues a start command. When an on demand I/O request is created, the user must specify the priority of the request, how often the completion event should be set, and the I/O request time-out.

Periodic I/O requests are executed periodically on the IOP. Accordingly, the user must specify the period of the I/O request. In addition, the priority of the I/O request and I/O time-out must be provided. The periodic I/O requests may be scheduled to start on demand, at a specific time, or after a specific amount of time has expired. Furthermore, the periodic I/O requests may be scheduled to run forever or to stop on demand.

2.2 I/O User Interface Software Specifications

As discussed in Section 2.1, the I/O User Interface is divided into three sections: I/O Request Construction, I/O Data Access Operations, and I/O Request Scheduling.

The I/O Request Construction function involves the allocation and initialization of I/O request records on the CP, data buffers in shared memory, and program/data regions on the I/O Sequencer (IOS - see appendix C for detailed description). This function also communicates the I/O request specifications to the IOP and creates companion I/O records (the IOP transaction, chain and I/O request specification records that are the IOP duals of the transaction, chain and I/O request records on the CP) which are used when executing and processing the I/O requests.

The Data Access Operations use the shared memory data buffers allocated during the I/O Request Construction to provide the appearance of memory mapped I/O. These functions use semaphores and double buffering to maintain consistent data sets. The Data Access Operations also control the interprocessor communication required to allow the user to select/deselect transactions and obtain error information.

The I/O Request Scheduling function supports the use of flags and events to synchronize application tasks on the CP with their corresponding I/O requests.

2.2.1 I/O Request Construction

The applications user is able to construct transactions, chains, and I/O requests in an hierarchical manner. Since the I/O requests are created on the CP and all I/O activity is performed by the I/O System Services on the IOP, the I/O Request Construction process must initialize the CP, IOP and shared memory. The design of the I/O Request Construction function is discussed in the following sections.

2.2.1.1 Creation of a Transaction

The basis of the construction of an I/O request is the specification of an I/O transaction. The format for the creation of a transaction is as follows:

```
CREATE_TRANSACTION( TRANSACTION_ID,
                    TRANSACTION_INFO);
```

where

TRANSACTION_ID is an identifier (returned by the I/O User Interface) which uniquely specifies the transaction. The applications programmer uses the TRANSACTION_ID when using system calls during later operation.

TRANSACTION_INFO is a discriminated record (provided by the user) based on whether it is an INPUT or OUTPUT transaction (specified by its IO parameter).

1. The TRANSACTION_INFO record always has the following fields:

   DIU_ID is an identifier that specifies the device interface unit addressed.

   NUM_DATA_BYTES_OUT is a parameter which specifies the number of bytes that will be sent to the DIU.

   DYNAMIC_OR_STATIC is a boolean that specifies whether the output data associated with the transaction is dynamic or static.

   DATA_BUFFER_OUTPUT is a parameter which specifies the address on the CP of the user's output data buffer (for specifying output commands to the DIU for this transaction).

2. If IO = INPUT, then the TRANSACTION_INFO record has the following additional fields:

   NUM_DATA_BYTES_IN is a parameter which specifies the number of bytes that will be returned by the DIU.

   MAX_BEFORE_BYPASS is a parameter which specifies the maximum number of errors that can be tolerated before the transaction is bypassed. If the parameter is zero, then the transaction will not be bypassed.

   TIME_OUT is a parameter which specifies the maximum length of time (even number of microseconds) that can expire before an incoming data byte is received.

   DATA_BUFFER_INPUT is a parameter which specifies the address on the CP of the user's input data buffer (to record the response data from the DIU for this transaction)

The CREATE_TRANSACTION call allocates and initializes a transaction record object using the information provided by the user. The allocation of the record involves the initialization of an element of a local transaction pointer array. The local transaction pointer array accesses a transaction record using the transaction ID as an index. As a result, the I/O User Interface can directly access any information associated with a transaction, if its ID is known. The I/O User Interface returns the transaction ID to allow the application to identify each transaction. The identity of the transaction is necessary when the user constructs chains, checks for transaction errors, and selects/deselects transactions.

Due to limited dual ported memory (4000 bytes for user I/O chains and data) in the IOS, the number of transactions that can be created is limited. When the transaction is created, the size requirements of the transaction (program and data memory requirements) are calculated and compared to the available memory space. If memory required is less than the memory available, then a section of dual ported memory is reserved for this transaction. If the memory required is greater than the available memory, an exception is raised to the user indicating the problem.

After the initialization of the transaction record, the CREATE_TRANSACTION process running on the CP allocates two input data buffers (if an input transaction) and two output data buffers in shared memory. Two sets of I/O data buffers are allocated per transaction to eliminate data contention problems between the I/O System Services read/write tasks executing on the CP and the read/write tasks executing on the IOP. Data contention is eliminated because each processor is always able to obtain one of the two buffers. The processor that is writing data into shared memory determines which buffer is available (not locked by the other processor) prior to writing. The processor that is reading data determines the available buffer that contains the most current data. In order to select and lock the I/O buffers, the processes running on the CP and IOP allocate common data buffer select variables (pre-defined flags) in shared memory. These select variables are protected by semaphores so that the variables are only modified by one process at a time. In addition, two error status records are allocated in shared memory to associate error information with each set of I/O buffers. As a result, this "Double Buffering" scheme allows the simultaneous writing and reading of I/O request data (to different buffers) while maintaining consistent data and error information.

After the I/O buffers have been allocated, the output data buffers are initialized if the user provides initial data. The CP process then waits (polls a flag) until the corresponding IOP communication task acknowledges (sets the flag) that it can accept the new transaction record. The CP process then writes the transaction specifications to shared memory. The CP process also writes, into a predefined memory location, the addresses of the previously allocated shared I/O data buffers. After the information has been written, the CP process signals the IOP communication task using an event. The IOP task reads the transaction specifications, allocates a companion transaction record, initializes a local transaction pointer, and creates the transaction record. The communication protocol between the CP and IOP is illustrated in the Figure 9.

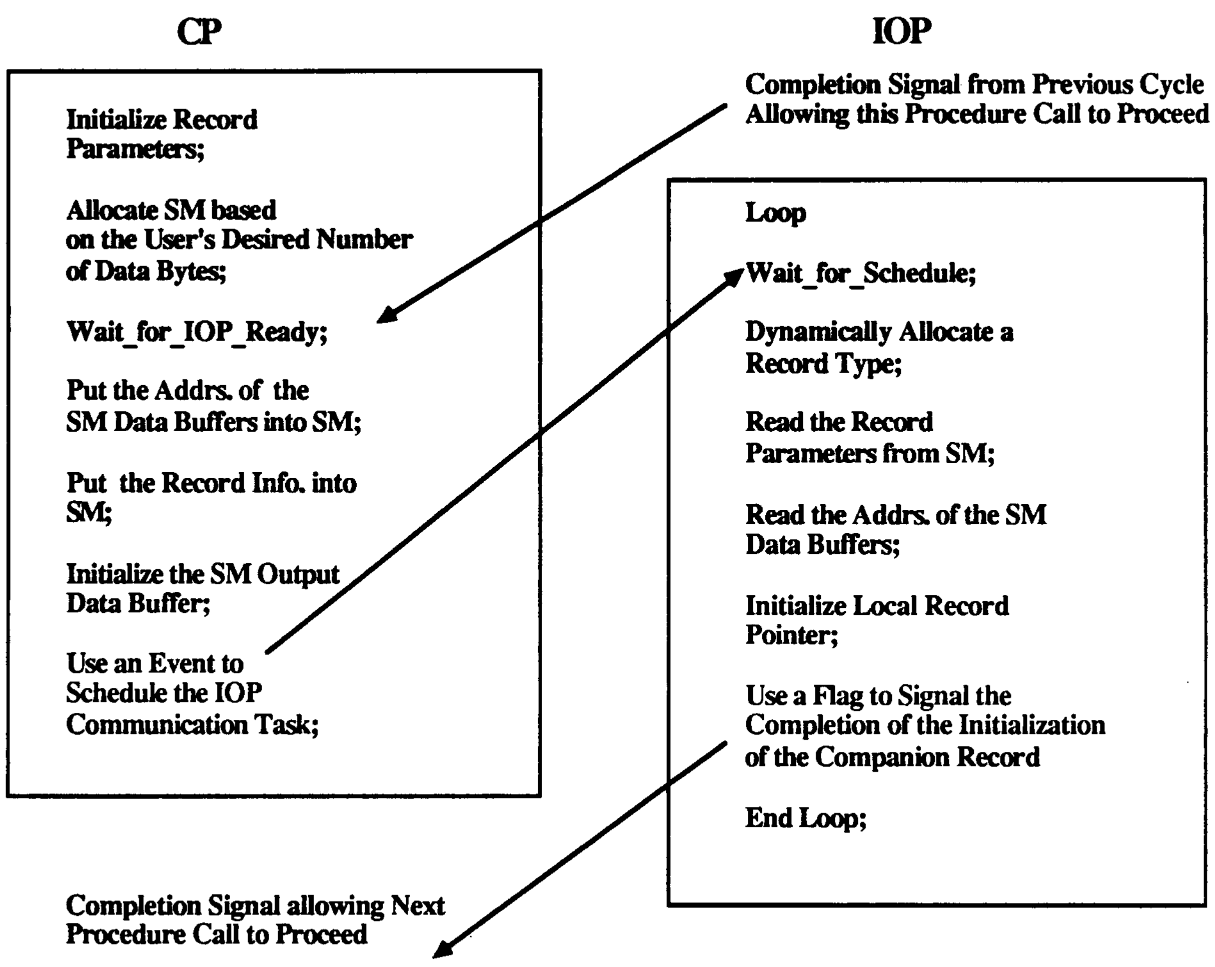

CP
IOP
Completion Signal from Previous Cycle Allowing this Procedure Call to Proceed
Initialize Record Parameters;
Allocate SM based on the User's Desired Number of Data Bytes;
Wait_for_IOP_Ready;
Put the Addrs. of the SM Data Buffers into SM;
Put the Record Info. into SM;
Initialize the SM Output Data Buffer;
Use an Event to Schedule the IOP Communication Task;
Loop
Wait_for_Schedule;
Dynamically Allocate a Record Type;
Read the Record Parameters from SM;
Read the Addrs. of the SM Data Buffers;
Initialize Local Record Pointer;
Use a Flag to Signal the Completion of the Initialization of the Companion Record
End Loop;
Completion Signal allowing Next Procedure Call to Proceed

Figure 9. Interprocessor Synchronization for Communication of I/O Request Specifications

2.2.1.2 Creation of a Chain

After the specification of the transactions, they are grouped into chains. The format for the specification of a chain is as follows:

```
CREATE_CHAIN( CHAIN_ID,
              NETWORK_ID,
              TRANSACTION_LIST)
```

where

CHAIN_ID is an identifier (returned by the I/O User Interface) which uniquely specifies the chain. The application uses the CHAIN_ID when using system calls during later operation.

NETWORK_ID is an identifier which specifies the network on which the chain will be executed (provided by the user).

TRANSACTION_LIST is a variant object which specifies the number of transactions in the chain and the corresponding array of TRANSACTION_IDs (provided by the user).

The CREATE_CHAIN call allocates and initializes a chain record object using the information provided by the user. The allocation of the record involves the initialization of an element of a local chain pointer array. The local chain pointer array accesses a chain record using the chain ID as an index. As a result, the I/O User Interface can directly access any information associated with a chain, if its ID is known. The initialization process forms a doubly linked tree between the transactions and their respective chain allowing direct access to a chain from a transaction or to a transaction from a chain (illustrated in Figure 10). The CP process returns the chain ID to allow the applications user to identify each chain. The identity of the chain is necessary when the user constructs I/O requests and checks for chain errors.

After the allocation and initialization of the chain record, the CP process communicates the information to the IOP communication task using the same protocol as involved in sending transaction records. The IOP task creates a companion chain record on the IOP and assigns an element of its local chain pointer array to point to it.

2.2.1.3 Creation of an I/O Request

After the specification of the chains, they are grouped into I/O requests. The format for the specification of an I/O request is:

```
CREATE_IOR( IOR_ID,
            CHAIN_LIST,
            SCHED_INFO)
```

where

IOR_ID is an identifier (returned by I/O User Interface) which allows the application to uniquely specify the I/O request. The applications programmer uses the IOR_ID when using system calls during later operation.

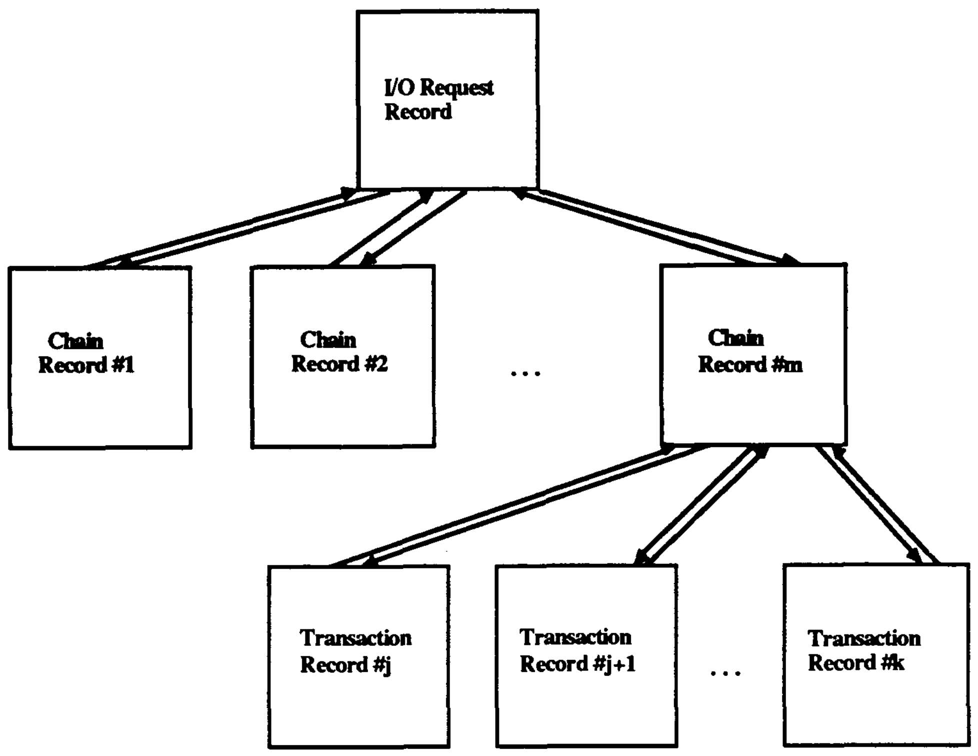


Figure 10. Data Structure used for I/O Request Construction and Interprocessor Communication

CHAIN_LIST is a variant object which specifies the number of chains in the I/O request and the corresponding array of CHAIN_IDs (provided by user).

SCHED_INFO is a discriminated record (provided by the user) based on whether it is an ON_DEMAND or PERIODIC I/O request (specified by its HOW_SCHEDULED parameter).

1. The SCHED_INFO record always has the following fields:

   PRIORITY is a parameter which specifies the priority (0 - 7) of the associated IOP task. A lower priority number implies a lower degree of urgency and an I/O request with a priority of 7 will preempt lower priority requests.

   COMPLETION_EVENT is a parameter indicating the frequency (ONCE_ONLY, ALWAYS, NEVER) with which an I/O request uses an event to signify its completion.

   IOR_TIME_OUT is a parameter which specifies the maximum length of time (in microseconds) that an I/O request actively possesses an I/O network(s) during its execution. The parameter is used by the I/O System Services when executing and processing the I/O request.

2. If HOW_SCHEDULED = PERIODIC, then the SCHED_INFO record requires the following additional fields:

   REPETITION_PERIOD is a parameter which specifies the length of the repetition period in seconds.

   WHEN_TO_STOP is an enumeration type which may be assigned to NEVER_STOP or STOP_ON_DEMAND. The parameter is used to specify how the I/O request should complete.

   START is a discriminated record based on whether the HOW_STARTED discriminate is specified to be START_ON_DEMAND, START_AFTER_DELAY, or START_AT_ABSOLUTE_TIME.

   a. If HOW_STARTED = START_ON_DEMAND
      - Additional fields are not necessary.
   b. If HOW_STARTED = START_AFTER_DELAY
      - The additional field WAIT_FOR is necessary which specifies the length of time to wait before the I/O request is initiated (in seconds).
   c. If HOW_STARTED = START_AT_ABSOLUTE_TIME
      - The additional field AT_ABSOLUTE_TIME is necessary to specify the time to start the I/O request (in seconds).

The CREATE_IOR call allocates and initializes an I/O request record object using the information provided by the user. The allocation of the record involves the initialization of an element of a local I/O request pointer array. The local I/O request pointer array accesses an I/O request record using the I/O request ID as an index. As a result, the I/O User Interface can directly access any information associated with an I/O request, if its ID is known. The initialization process forms a doubly linked tree between the chains and their respective I/O request (allowing direct access to an I/O request from a chain or to a chain from an I/O request). As a result, transactions can indirectly access their associated I/O request as illustrated in the Figure 10. The CREATE_IOR process running on the CP returns the I/O request ID to allow the applications user to identify each request. The identity of the request is necessary when referencing I/O requests from application processes/tasks.

The initialization of the I/O request record involves the calculation of a I/O System Services I/O request time out. As previously mentioned, the I/O request time out is the length of time that the I/O request actively possesses an I/O network(s) during its execution. Since each chain of the I/O request executes (nearly) simultaneously, the I/O request time out only depends on the longest chain time out. The chain time out, which is the length of time a chain requires possession of a network, is based on the number of transactions, length of the transaction time outs, and amount of I/O data associated with the chain. After the I/O System Services I/O request time out is calculated, it is compared to the I/O request time out provided by the user to determine if the user's I/O request time out is too short. If the user's I/O request time out is deemed to be too short, a warning is issued to the user.

After the allocation and initialization of the I/O request record, the CP process communicates the information to the IOP communication task using the same protocol as involved in sending transaction records. The IOP task creates a companion I/O request record and assigns an element of its local I/O request pointer array to point to it.

2.2.2 I/O Data Access Operations

The applications user is able to access the data associated with the I/O requests in a memory mapped fashion. The appearance of memory mapped I/O is accomplished by allocating data sections in shared memory to emulate the data regions of the IOS and making the reading/writing protocol transparent to the user. The I/O Data Access function also provides error detection and chain reconfiguration capabilities to the user. The design of the I/O Data Access function is discussed in the following sections.

2.2.2.1 I/O User Interface Write Procedures

The I/O User Interface allows the user to write output data to DIUs on a transaction by transaction or I/O request by I/O request basis. The ability to write data only to a transaction is desirable when a chain is mixed but primarily consists of static command frames. The ability to write data to all of the output data buffers is desirable when one or more of the chains of an I/O request are mixed but primarily consist of dynamic command frames. The procedures for writing output data to DIUs are as follows:

```
1. WRITE_TRANSACTION( TRANSACTION_ID,
                      IOR_DATA_IS_CONSISTENT,
                      LOCKED);

2. WRITE_IOR( IOR_ID,
              LOCKED);

3. WRITE_INITIAL_IOR_DATA( IOR_ID,
                           LOCKED);
```

where

TRANSACTION_ID is a parameter which uniquely identifies the associated transaction. It is assigned by the system when the transaction is created and must be provided by the user for the WRITE_TRANSACTION system call.

IOR_ID is a parameter which uniquely identifies the associated I/O request. It is assigned by the system when the I/O request is created and must be provided by the user for the WRITE_IOR system call.

IOR_DATA_IS_CONSISTENT is a boolean (provided by the user) which, when TRUE, signifies that the output data for the I/O request is consistent.

LOCKED is a boolean (returned by I/O User Interface) which, when TRUE, signifies that the shared memory buffer select area was locked when the applications process/task attempted to select an output buffer.

Since the I/O System Services on the IOP controls access to the IOS dual ported memory, the output data is written into shared memory from the CP rather than directly to the IOS. An output data buffer select variable is used to determine an available buffer. The buffer select variable is guarded by a semaphore to guarantee mutual exclusion. After a buffer is selected, the output data is written and the buffer is made available to the I/O System Services on the IOP. As a result, the chain output data sets of the I/O request are consistent and I/O data contention is avoided.

The WRITE_IOR procedure performs a test and set operation on the semaphore that guards the output data buffer select variable for the I/O request and if the select region is unlocked, it selects an available buffer, unlocks the select region, and writes the dynamic output data for the entire I/O request into shared memory. If the buffer select variable is locked, the procedure continues to perform the test and set operation until the region becomes unlocked or 100 test and set iterations pass. If 100 iterations pass and the select region is still locked, it is assumed that a fault has caused a deadlock situation (select region is locked but neither processor has control of it). If such a deadlock situation occurs, the procedure disregards the semaphore, determines the available buffer, resets the semaphore, sets the LOCKED parameter, and writes the output data.

After the output data has been written into shared memory, the select variables must be updated to specify the available buffer with the most current data. The WRITE_IOR procedure performs the test and set process as previously described, and when the select region is unlocked, the buffer select variable (in shared memory) is set equal to the buffer into which the output data was written.

Similarly, the WRITE_TRANSACTION procedure checks the locking mechanism, selects a buffer, and writes the output data for the specified transaction. Yet, the user may want to change one transaction in each chain before the associated I/O request is executed. Accordingly, the Write Transaction procedure allows the user specify if the data is consistent throughout the I/O request. If the data is not consistent, the buffer is not made available to the I/O System Services on the IOP, and the I/O System Services will read the other data buffer until the user states that the data is consistent. As a result, the user does not have to worry about an I/O request being executed with inconsistent data.

The WRITE_INITIAL_IOR_DATA procedure is identical to the WRITE_IOR procedure except that it writes all of the I/O request output data (dynamic and static) into shared memory. Typically, this procedure is used to initialize the IOS output buffers.

2.2.2.2 I/O User Interface Read Procedures

The I/O User Interface allows the user to read input data from DIUs only on an I/O request by I/O request basis. Since the input data is dynamic, all of the input data associated with an I/O request is desired. The format for reading data from DIUs is as follows:

```
READ_IOR( IOR_ID,
          LOCKED,
          ERROR,
          OLD_DATA);
```

where

IOR_ID is a parameter which uniquely identifies the associated I/O request. It is assigned by the system when the I/O request is created and must be provided by the user for the READ_IOR system call.

LOCKED is a boolean (returned by I/O User Interface) which, when TRUE, signifies that the shared memory buffer select area was locked when the applications process/task attempted to select an input buffer.

ERROR is a boolean (returned by I/O User Interface) which, when TRUE, signifies that an error occurred in at least one of the chains of the I/O request. The application can isolate the location of the error by invoking the procedures TRANSACTION_ERROR and CHAIN_ERROR.

OLD_DATA is a boolean (returned by I/O User Interface) which, when TRUE, signifies that the I/O request data was NOT updated by the IOP since the previous READ_IOR call. The data was NOT read during this call.

Since the I/O System Services running on the IOP controls access to the IOS dual ported memory, the input data is read from shared memory to the CP rather than directly from the IOS. An input data buffer select variable is used to determine the available buffer that contains the most current data. A semaphore is used to guarantee that the buffer select variable is only modified by one processor at a time (note that this is a different semaphore than that used for output data buffers). After the available buffer is selected, it is locked by the I/O System Services on the CP to maintain consistent data throughout the buffer (prevents the I/O System Services on the IOP from writing into the buffer while the CP is reading from it), and the input data and associated error information is read into the CP's local memory.

The READ_IOR procedure performs a test and set operation on the semaphore that guards the input data buffer select variable for the I/O request and if select region is unlocked, the procedure selects/locks an available buffer, unlocks the select region, and reads the input data and error status for the I/O request. If the buffer select variable is locked, the procedure continues to perform the test and set operation until the region becomes unlocked or 100 test and set iterations pass. If 100 iterations pass and the select region is still locked, it is assumed that a fault has caused a deadlock situation (region is locked but neither processor has control of it). If such a deadlock situation occurs, the procedure disregards the semaphore, determines the available buffer, resets the semaphore, sets the LOCKED parameter, and reads the input data and error information.

After the input data and error status has been read from shared memory, the input data buffer select variable must be updated to unlock the input data buffer. The Read I/O request procedure performs the test and set process as previously described, and when the select region is unlocked, the buffer is unlocked by modifying the select variable in shared memory.

The I/O Request Completion Function (discussed in Section 3.1.1.3) checks for chain and transaction errors. The errors are communicated to the I/O System Services on the CP by setting flags in shared memory. The ERROR flag is an ORing of all chain and transaction errors.

An "old data" flag is returned to the application task to notify the task that the input data has not been updated since it was last read. This flag is set by the Read I/O Request process, and it is reset when the IOP writes new data into the shared data buffers. If the Read I/O Request procedure returns a true value in the OLD_DATA flag and the application process is synchronized with the I/O request, then a scheduling overrun has occurred.

2.2.2.3 I/O User Interface Selection and Deselection Procedures

The I/O User Interface allows the user to select and deselect transactions from their corresponding chains. As a result, the user has some error recovery control and can dynamically reconfigure the I/O requests. The format for selecting and deselecting transactions is as follows:

```
1. SELECT_TRANSACTION( TRANSACTION_ID,
                       REQUEST_ACCEPTED)

2. DESELECT_TRANSACTION( TRANSACTION_ID,
                         REQUEST_ACCEPTED)
```

where

TRANSACTION_ID is a parameter which uniquely identifies the associated transaction. It is assigned by the system when the transaction is created and must be provided by the user for the SELECT_TRANSACTION or DESELECT_TRANSACTION system call.

REQUEST_ACCEPTED is a boolean (returned by I/O User Interface) that, if TRUE, signifies that the command was accepted.

As previously mentioned, the I/O System Services on the IOP controls access to the IOS dual ported memory. Accordingly, the transaction selection (or deselection) command must be communicated to the IOP. The basic unit of information that must be transmitted is the type of command (selection or deselection) and transaction ID. The I/O User Interface uses two five element arrays, two semaphores and two boolean flags to communicate transaction selection and deselection commands from the CP to the IOP. The arrays (one for selection and one for deselection) are buffers to communicate the transaction IDs to the IOP. The semaphores are used to maintain consistent sets of IDs, and the boolean flags are used to notify the IOP of the selection and deselection commands. The CP control flow involved in the communication of a SELECT_TRANSACTION procedure call is outlined below:

1) The application invokes the SELECT_TRANSACTION system call.
2) The CP Select Transaction process test and sets the semaphore locking the shared memory select buffer.
3) If the IOP is reading from or writing to the buffer (is locking the region), the Select Transaction process notifies the application that the request was not accepted.

4) If the IOP is not reading from or writing to the buffer (the region is available), the Select Transaction process searches for the first unused element in the array (the test and set function has locked the buffer).
5) If the array is full (five selection requests are pending), the Select Transaction process notifies the application that the request was not accepted and unlocks the buffer.
6) If an element of the array is available, the Select Transaction process initializes the element, sets a shared memory flag to notify the IOP, and unlocks the buffer.

The IOP control flow involved in the processing of an I/O request (with respect to the SELECT_TRANSACTION and DESELECT_TRANSACTION procedure calls) is outlined below:

1) The Queue Manager task accepts a pending I/O request to be processed.
2) The Queue Manager task checks the selection shared memory flag to determine if a selection request has been made.
3) If the flag is not set, then the Queue Manager task processes the I/O request.
4) If the flag is set, the task test and sets the semaphore locking the shared memory select buffer.
5) If the CP is reading from or writing to the buffer (is locking the region), the Queue Manager processes the I/O request.
6) If the CP is not reading from or writing to the buffer (the region is available), the Queue Manager process reads the first transaction ID (the test and set function has locked the buffer).
7) The Queue Manager task modifies the IOS program area to select the transaction, initializes the transaction ID to a null value, and checks the array to determine if another transaction is to be selected.
8) If another transaction ID is in the shared memory buffer, then (7) is repeated. If not, then the Queue Manager task unlocks the select buffer.
9) After the select transaction requests have been processed, the Queue Manager repeats the same process (2-8) for deselection requests.
10) After the deselect transaction requests have been processed, then the Queue Manager task processes the I/O request.

To deselect a transaction, the Queue Manager task modifies the chain program so that the IOS skips over the set of instructions used to execute the deselected transaction. To select a transaction, the task modifies the chain program so that the IOS does not skip over the relevant set of instructions.

2.2.2.4 I/O User Interface Error Checking Procedures

Errors may occur during the execution of the I/O requests. The user is notified of the occurrence of an error(s) when the input data corresponding to the I/O request is read. The I/O User Interface provides system calls to allow the user to determine the location of the error(s). The formats for these calls are as follows:

```
1. TRANSACTION_ERROR(TRANSACTION_ID)

2. TRANSACTION_IS_BYPASSED(TRANSACTION_ID)

3. CHAIN_ERROR( CHAIN_ID,
                ERROR_IN_CHAIN,
                ALL_TRANSACTIONS_ARE_BAD,
                CHAIN_DID_NOT_COMPLETE,
                TRANSACTION_NOT_EXECUTED,
                NETWORK_STATUS)
```

where

TRANSACTION_ID is a parameter which uniquely identifies the associated transaction. It is assigned by the system when the transaction is created and must be provided by the user for the TRANSACTION_ERROR function call.

CHAIN_ID is a parameter which uniquely identifies the associated chain. It is assigned by the system when the chain is created and must be provided by the user for the CHAIN_ERROR procedure call.

ERROR_IN_CHAIN is a boolean parameter (returned by the I/O User Interface) that, if TRUE, signifies that an error occurred during the execution of the chain.

ALL_TRANSACTIONS_ARE_BAD is a boolean parameter (returned by the I/O User Interface) that, if TRUE, signifies that errors occurred in all of the transactions of the chain.

CHAIN_DID_NOT_COMPLETE is a boolean parameter (returned by the I/O User Interface) that, if TRUE, signifies that the chain did not complete when the I/O request was processed.

TRANSACTIONS_NOT_EXECUTED is a boolean parameter (returned by the I/O User Interface) that, if TRUE, signifies that one or more transactions in the chain were not executed because they were deselected and/or bypassed.

NETWORK_STATUS is a enumerated object (returned by the I/O User Interface) that signifies the state of the network when the I/O request was executed (In_Service, Temporarily_Out_of_Service, Permanently_Out_of_Service).

As discussed in the I/O Communications Management Functional Requirements (Section 3.1), the I/O Traffic Control function performs the I/O Request Completion processing. The I/O Request Completion function primarily consists of chain and transaction error processing. If an error is detected, then flags are set in shared memory to notify the I/O System Services on the CP of its existence and location. Each transaction has two boolean status flags (to convey error and bypass information) and each chain has five status flags (in correspondence with the parameters returned by the CHAIN_ERROR procedure). The CHAIN_ERROR procedure returns the state of the corresponding error flags to the user. If the state of the ERROR_IN_CHAIN boolean is true, then the transaction(s) which has an error can be isolated using the TRANSACTION_ERROR procedure. Furthermore, if the state of the TRANSACTION_NOT_EXECUTED flag is true, the TRANSACTION_IS_BYPASSED procedure can be used to determine whether or not a transaction has been bypassed by the I/O System Services. The TRANSACTION_IS_BYPASSED procedure specifies which transactions have been bypassed by the I/O System Services (not deselected by the user). If one or more transactions have been deselected by the user, the TRANSACTION_NOT_EXECUTED flag will be set by the I/O System Services, but the associated TRANSACTION_IS_BYPASSED fields will not be set. The application must account for the transactions that it deselects.

The status information returned by the CHAIN_ERROR, TRANSACTION_ERROR, and TRANSACTION_IS_BYPASSED procedures is only valid after the READ_IOR procedure has been called due to the "Double Buffering" scheme. Since two sets of I/O buffers are used for interprocessor data communication, two sets of status buffers are required to maintain consistent data/status information (each data buffer must its own status buffer). When the READ_IOR procedure is called, one set of data and status buffers is written into the local memory on the CP from shared memory. The CHAIN_ERROR, TRANSACTION_ERROR, and TRANSACTION_IS_BYPASSED procedures read the status buffer in local memory when returning status information to the application. As a result, the READ_IOR procedure must be called prior to invoking any of these error checking procedures.

2.2.2.5 I/O User Interface Overrun Check

If an I/O request can not be executed when its scheduling requirements have been fulfilled, then a scheduling overrun occurs. The I/O User Interface allows the user to determine if an overrun has occurred using the following function call:

```
IOR_HAS_OVERRUN(IOR_ID)
```

where

> IOR_ID is a parameter which uniquely identifies the associated I/O request. It is assigned by the system when the I/O request is created and must be provided by the user for the IOR_HAS_OVERRUN system call.

The occurrence of an overrun is detected by the Posting task associated with the I/O request (a Posting task is a task that is scheduled on the IOP based on the scheduling requirements of an I/O request). A parameter (integer number) in shared memory is updated to reflect the occurrence (or frequency of occurrences) of an overrun. This parameter is read from shared memory and returned to the user by the IOR_HAS _OVERRUN function.

2.2.3 I/O Request Scheduling

Each I/O request may correspond to one or more application tasks executing on the CP. Since the I/O requests are processed on the IOP, the I/O User Interface must provide flags and events (synchronization mechanisms) to coordinate the I/O requests with the application tasks.

2.2.3.1 Synchronization Using Flags

Flags are used to indicate the completion of I/O requests. The I/O Request Completion processing involves the setting of a flag in shared memory to notify the user that the request has been executed and processed. The I/O User Interface allows the user to read and clear these flags.

1. IOR_READY(IOR_ID)

2. CLEAR_IOR_READY(IOR_ID)

3. IOR_READY_AND_CLEAR(IOR_ID)

where

> IOR_ID is a parameter which uniquely identifies the associated I/O request. It is assigned by the I/O System Services when the I/O request is created and must be provided by the user for the system call.

The IOR_READY function call reads the completion flag associated with the I/O request identified by the IOR_ID parameter. If the I/O request has completed, then the flag will be set (true). The flag may be cleared using the procedure CLEAR_IOR_READY. In addition, the flag can be read and cleared in a single step using the procedure IOR_READY_AND_CLEAR.

Flags are also used to synchronize the I/O System Services on the CP and the IOP. The IOP must wait for the CP to create and communicate the I/O request specifications. Alternatively, the CP must wait for the IOP to initialize the I/O Services. Accordingly, functions are provided to allow the CP to signal and wait for the IOP.

1. CP_COMPLETED

2. WAIT_UNTIL_IOP_COMPLETED

The CP_COMPLETED function sets a flag in shared memory to acknowledge the completion of I/O request initialization process. The WAIT_UNTIL_IOP_COMPLETED function allows the CP to wait until the IOP initializes the I/O Services before continuing.

2.2.3.2 Synchronization Using Events

The completion of an I/O request is indicated by an event if user has specified this option when creating the I/O requests. The I/O User Interface allows the user to obtain a pointer to the I/O request completion event allowing the scheduling of application tasks through the GPC Real Time Operating System. The format for the function call is as follows:

IOR_COMPLETION_EVENT(IOR_ID)

where

IOR_ID is a parameter which uniquely identifies the associated I/O request. It is assigned by the system when the I/O request is created and must be provided by the user for the system call.

The frequency that the I/O User Interface uses an event to signal the completion of an I/O request is specified when the request is created. The completion event may be specified to occur once only, always, or never. The event is used to either initially synchronize an application task with its corresponding I/O request or to periodically trigger an on demand application task.

Events are also used to start and stop I/O requests. The following system calls are available to the user.

1. START_IOR(IOR_ID)

2. STOP_IOR(IOR_ID)

where

IOR_ID is a parameter which uniquely identifies the associated I/O request. It is assigned by the system when the I/O request is created and must be provided by the user for the system call.

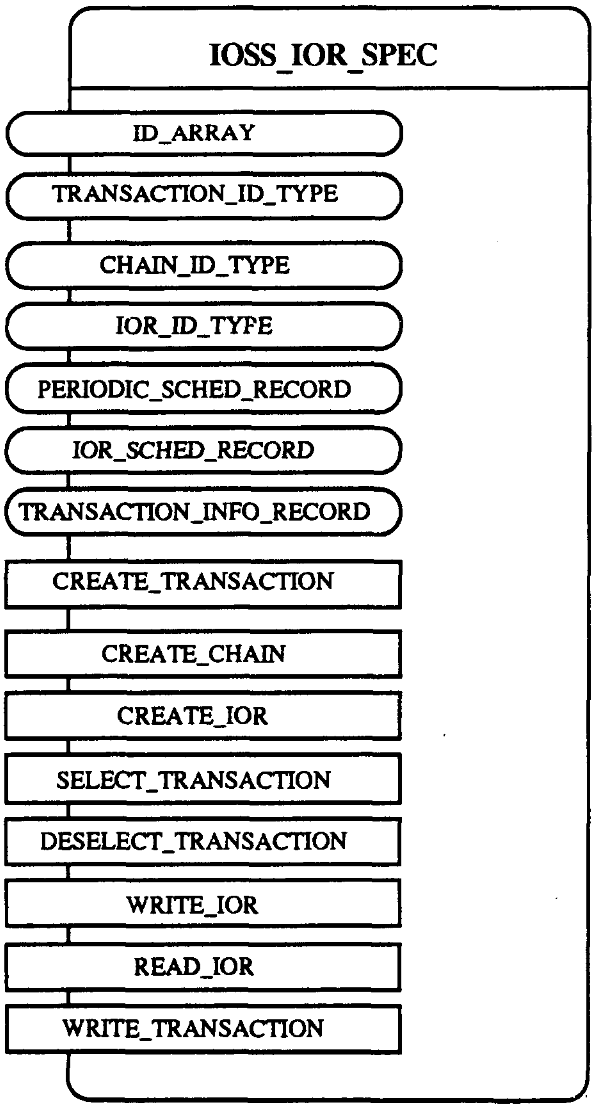
These procedure calls are used to start or stop an I/O request after a periodic request has been created with the START_ON_DEMAND or STOP_ON_DEMAND option. The START_IOR procedure is also used to start on demand I/O requests.

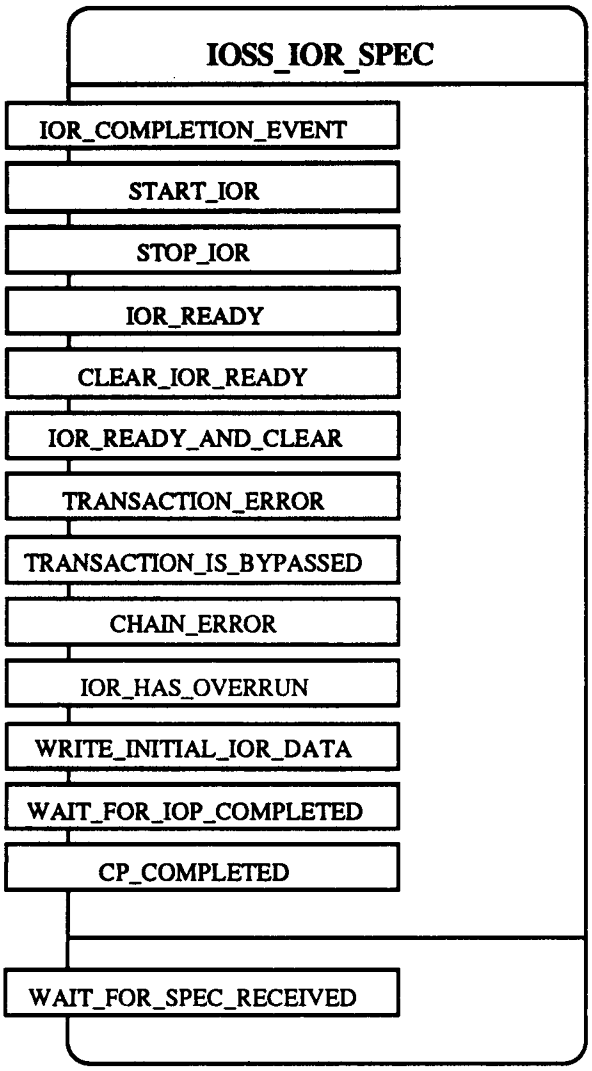
2.3 I/O User Interface Software Process Descriptions

The I/O User Interface Software Process Descriptions divide the description of the I/O User Interface into functional packages. This section uses Booch diagrams (a high level diagrammatic design methodology put forward by Grady Booch - see [3]) and process descriptions to present the Software Specifications in more detail. The Booch diagrams are used to map the I/O User Interface Software Specifications into functional packages, tasks, and subprograms. The process descriptions are used to describe these functional groups in detail.

The I/O User Interface is divided into two functional packages: I/O System Services I/O Request Specification and Application Log.

2.3.1 I/O System Services I/O Request Specification




IOSS_IOR_SPEC
IOR_COMPLETION_EVENT
START_IOR
STOP_IOR
IOR_READY
CLEAR_IOR_READY
IOR_READY_AND_CLEAR
TRANSACTION_ERROR
TRANSACTION_IS_BYPASSED
CHAIN_ERROR
IOR_HAS_OVERRUN
WRITE_INITIAL_IOR_DATA
WAIT_FOR_IOP_COMPLETED
CP_COMPLETED
WAIT_FOR_SPEC_RECEIVED

2.3.1.1 Process Name:	Create Transaction
Inputs:	Transaction Information Record
Outputs:	Transaction Identifier Transaction Specifications
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.1 I/O User Interface Software Specifications, Section 2.2.1.1
Notes:	None

Description:

The Create Transaction process allocates memory on the CP and initializes a transaction record (transaction specifications) based on the transaction information provided by the application. The application can create two types of transactions:

1) An input transaction which involves an output sequence of instructions that requests information from a DIU and an input sequence of instructions that waits for the DIU response.
2) An output transaction which consists of an output sequence of instructions that sends information to a DIU and does not expect a response.

The application must provide the following information for all transactions:

1) The type of transaction - input or output.
2) DIU identifier.
3) Number of output data bytes.
4) Type of output data - dynamic or static.
5) Pointer to the output data buffer on the CP.

The application must provide the following additional information for input transactions:

1) Number of input data bytes.
2) Maximum number of errors allowable before system bypass.
3) Allowable time out before an incoming data byte is received.
4) Pointer to the input data buffer on the CP.

The process calculates the transaction identifier and returns it to the application. The identity of the transaction is necessary when the user constructs I/O chains, checks for transaction errors, and selects/deselects transactions. The process also initializes an element of a local transaction pointer array which allows the I/O User Interface to access any information associated with the transaction if its ID is known. In addition, the process initializes internal variables (variables that are transparent to the application) which are used for I/O error processing.

After the transaction record is initialized, the Create Transaction process communicates this information to the IOP communication task through shared memory. The communication process is initiated by waiting for the IOP task to acknowledge that it is ready for new data. After the acknowledgement has been received, the Create Transaction process writes the transaction record information into pre-defined locations. The process then allocates two sets of I/O buffers in shared memory to communicate data between the CP and IOP. The pointers to these data buffers are communicated to the IOP task through pre-defined locations in shared memory. In addition, if the application provides initial output data, the process initializes the shared memory output data buffers. After the allocation and initialization is finished, the Create Transaction process uses an event to notify an IOP communication task that the new transaction specifications and data pointers are available and can be read.

When the event activates the IOP communication task, the task creates and initializes a companion transaction record on the IOP and assigns an element of its local transaction pointer array to point to it.

2.3.1.2 Process Name:	Create Chain
Inputs:	Network Identifier Transaction List
Outputs:	Chain Identifier Chain Specifications
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.1 I/O User Interface Software Specifications, Section 2.2.1.2
Notes:	None

Description:

The Create Chain process allocates and initializes a chain record object using the information provided by the application. The allocation of the record involves the initialization of an element of a local chain pointer array. The local chain pointer array accesses a chain record using the chain identifier as an index. As a result, the I/O User Interface can directly access any information associated with a chain, if its ID is known. The initialization process forms a doubly linked tree between the transactions and their respective chain (allowing direct access to a chain from a transaction or to a transaction from a chain). The Create Chain process returns the chain ID to allow the applications user to identify each chain. The identity of the chain is necessary when the user constructs I/O requests and checks for chain errors.

After the allocation and initialization of the chain record, the Create Chain process communicates the information to the IOP communication task using the same protocol as involved in sending transaction records. The IOP task creates a companion chain record on the IOP and assigns an element of its local chain pointer array to point to it.

2.3.1.3 Process Name:	Create I/O Request
Inputs:	Scheduling Information Chain List
Outputs:	I/O Request Identifier I/O Request Specifications
Requirements Reference:	I/O User Interface Functional Requirements , Section 2.1.1 I/O User Interface Software Specifications, Section 2.2.1.2
Notes:	None

Description:

The Create I/O Request call allocates and initializes an I/O request record object using the information provided by the user. The allocation of the record involves the initialization of an element of a local I/O request pointer array. The local I/O request pointer array accesses an I/O request record using the I/O request identifier as an index. As a result, the I/O User Interface can directly access any information associated with an I/O request, if its ID is known. The initialization process forms a doubly linked tree between the chains and their respective I/O request (allowing direct access to an I/O request from a chain or to a chain from an I/O request). As a result, transactions can indirectly access their associated I/O request. The CP returns the I/O request ID to allow the applications user to identify each request. The identity of the request is necessary when referencing I/O requests from application processes/tasks.

After the allocation and initialization of the I/O request record, the CP communicates the information to the IOP using the same protocol as involved in sending transaction records. The IOP creates a companion I/O request record and assigns an element of its local I/O request pointer array to point to it.

2.3.1.4 Process Name:	Select Transaction
Inputs:	Transaction Identifier
Outputs:	Confirmation of Select Request being Accepted Transaction Select Flag Transaction Select Array
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.3
Notes:	None

Description:

The Select Transaction process sends a selection request to the I/O Communications Management function. The protocol involved in the communication of the selection request is discussed in detail in Section 2.2.2.3 of the I/O User Interface Software Specifications.

2.3.1.5 Process Name:	Deselect Transaction
Inputs:	Transaction Identifier
Outputs:	Confirmation of Deselect Request being Accepted Transaction Deselect Flag Transaction Deselect Array
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.3
Notes:	None

Description:

The Deselect Transaction process sends a deselection request to the I/O Communications Management function. The protocol involved in the communication of the deselection request is discussed in detail in Section 2.2.2.3 of the I/O User Interface Software Specifications.

2.3.1.6 Process Name:	Write I/O Request
Inputs:	I/O Request Identifier
Outputs:	State of the Locking Semaphore Dynamic Output Data
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.1
Notes:	None

Description:

The Write I/O Request procedure performs a test and set operation on the semaphore that guards the output data buffer select variable for the I/O request. If the select region is unlocked, it selects an available buffer, unlocks the select region, and writes the dynamic output data for the entire I/O request into shared memory. If the buffer select region is locked, the procedure continues to perform the test and set operation until the region becomes unlocked or 100 test and set iterations pass. If 100 iterations pass and the select region is still locked, it is assumed that a fault has caused a deadlock situation (select region is locked but neither processor has control of it). If such a deadlock situation occurs, the procedure disregards the semaphore, determines the available buffer, resets the semaphore, sets the LOCKED parameter, and writes the output data.

After the output data has been written into shared memory, the output data buffer select variable must be updated to specify the available buffer with the most current data. The Write I/O Request procedure performs the test and set process as previously described, and when the select region is unlocked, the buffer select variable (in shared memory) is set equal to the buffer into which the output data was written.

2.3.1.7 Process Name:	Read I/O Request
Inputs:	I/O Request Identifier
Outputs:	State of the Locking Semaphore Error Flag Old Data Flag Input Data
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.2
Notes:	None

Description:

The Read I/O Request procedure performs a test and set operation on the semaphore that guards the input data buffer select variable for the I/O request. If select region is unlocked, the procedure selects/locks an available buffer, unlocks the select region, and reads the input data and error status for the I/O request. If the buffer select region is locked, the procedure continues to perform the test and set operation until the region becomes unlocked or 100 test and set iterations pass. If 100 iterations pass and the select region is still locked, it is assumed that a fault has caused a deadlock situation (region is locked but neither processor has control of it). If such a deadlock situation occurs, the procedure disregards the semaphore, determines the available buffer, resets the semaphore, sets the LOCKED parameter, and reads the input data and error information.

The I/O Request Completion function checks for chain and transaction errors. The errors are communicated to the CP by setting flags in shared memory. An error flag is returned to the application, and it represents an ORing of all chain and transaction errors.

An "old data" flag is returned to the application task to notify the task that the input data has not been updated since it was last read. This shared memory flag is set by the Read I/O Request process, and it is reset when the IOP writes new data into the shared data buffers. If the Read I/O Request procedure returns a true value in the "old data" flag and the application process is synchronized with the I/O request, then a scheduling overrun has occurred.

After the input data has been read from shared memory, the input data buffer select variable must be updated to unlock the input data buffer. The Read I/O request procedure performs the test and set process as previously described, and when the select region is unlocked, the buffer is unlocked by modifying the select variable in shared memory .

2.3.1.8 Process Name:	Write Transaction
Inputs:	Transaction Identifier Data is Consistent Boolean
Outputs:	State of the Locking Semaphore Output Data for the Transaction
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.1
Notes:	None

Description:

The Write Transaction procedure performs a test and set operation on the semaphore that guards the output data buffer select variable for the I/O request. If the select region is unlocked, it selects an available buffer, unlocks the select region, and writes the output data for the transaction into shared memory. If the buffer select region is locked, the procedure continues to perform the test and set operation until the region becomes unlocked or 100 test and set iterations pass. If 100 iterations pass and the select region is still locked, it is assumed that a fault has caused a deadlock situation (select region is locked but neither processor has control of it). If such a deadlock situation occurs, the procedure disregards the semaphore, determines the available buffer, resets the semaphore, sets the LOCKED parameter, and writes the output data.

The user may want to change one transaction in each chain before the associated I/O request is executed. Accordingly, the Write Transaction procedure allows the user specify if the data is consistent throughout the I/O request. If the data is not consistent, the buffer is not made available to the I/O System Services on the IOP. As a result, the user does not have to worry about an I/O request being executed with inconsistent data.

After the output data has been written into shared memory, the output data buffer select variable must be updated to specify the available buffer with the most current data (if the data is consistent). The Write Transaction procedure performs the test and set process as previously described, and when the select region is unlocked, the buffer select variable (in shared memory) is set equal to the buffer into which the output data was written.

2.3.1.9 Process Name: I/O Request Completion Event

Inputs: I/O Request Identifier

Outputs: Event Pointer

Requirements Reference: I/O User Interface Functional Requirements, Section 2.1.3
I/O User Interface Software Specifications, Section 2.2.3.2

Notes: None

Description:

The completion of an I/O request is indicated by an event if user has specified this option when creating the I/O requests. The I/O Request Completion Event process allows the application to obtain a pointer to the I/O request completion event allowing the scheduling of application tasks through the GPC Real Time Operating System. The Completion Event is used to either synchronize an application task with its corresponding I/O request or periodically trigger an on_demand application task.

2.3.1.10 Process Name: Start I/O Request

Inputs: I/O Request Identifier

Outputs: I/O Request Start Event

Requirements Reference: I/O User Interface Functional Requirements, Section 2.1.3
I/O User Interface Software Specifications, Section 2.2.3.2

Notes: None

Description:

Events are used to start and stop I/O requests. The Start I/O Request process is used to start an I/O request after a periodic request has been created with the START_ON_DEMAND option. The Start I/O Request process is also used to start on_demand I/O requests.

2.3.1.11 Process Name: Stop I/O Request

Inputs: I/O Request Identifier

Outputs: I/O Request Stop Event

Requirements Reference: I/O User Interface Functional Requirements, Section 2.1.3
I/O User Interface Software Specifications, Section 2.2.3.2

Notes: None

Description:

Events are used to start and stop I/O requests. The Stop I/O Request process is used to stop an I/O request after a periodic request has been created with the STOP_ON_DEMAND option.

2.3.1.12 Process Name: I/O Request Ready

Inputs: I/O Request Identifier

Outputs: I/O Request Completion Flag

Requirements Reference: I/O User Interface Functional Requirements, Section 2.1.3
I/O User Interface Software Specifications, Section 2.2.3.1

Notes: None

Description:

Flags are used to indicate the completion of I/O requests. The I/O Request Completion processing involves the setting of a flag in shared memory to notify the application that the request has been executed and processed. The I/O User Interface allows the user to read and clear these flags.

The I/O Request Ready process reads the completion flag associated with the I/O request identifier (provided by the application). If the I/O request has completed, then the flag that is returned to the application will be set (true).

2.3.1.13 Process Name:	Clear I/O Request Ready
Inputs:	I/O Request Identifier
Outputs:	I/O Request Completion Flag
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.3 I/O User Interface Software Specifications, Section 2.2.3.1
Notes:	None

Description:

The Clear I/O Request Ready procedure clears the completion flag that is associated with the I/O request identifier (provided by the application).

2.3.1.14 Process Name:	I/O Request Ready and Clear
Inputs:	I/O Request Identifier
Outputs:	I/O Request Completion Flag
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.3 I/O User Interface Software Specifications, Section 2.2.3.1
Notes:	None

Description:

Flags are used to indicate the completion of I/O requests. The I/O Request Completion processing involves the setting of a flag in shared memory to notify the application that the request has been executed and processed. The I/O User Interface allows the user to read and clear these flags.

The I/O Request Ready and Clear process reads and resets the completion flag associated with the I/O request identifier (provided by the application).

2.3.1.15 Process Name:	Transaction Error
Inputs:	Transaction Identifier
Outputs:	Transaction Error Flag
Requirements Reference:	I/O User Interface Functional Requirements , Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.4
Notes:	None

Description:

The I/O Request Completion function performs chain and transaction error processing. If an error(s) occur in the execution of the I/O request, then the I/O Request Completion function sets flags in shared memory to notify the I/O System Services on the CP. The Transaction Error process checks for the occurrence of an error in a transaction by reading the associated error flag.

The status information returned by the Transaction Error procedure is only valid after the Read I/O Request procedure has been called due to the "Double Buffering" scheme. Since two sets of I/O buffers are used for interprocessor data communication, two sets of status buffers are required to maintain consistent data/status information (each data buffer must its own status buffer). When the Read I/O Request procedure is called, one set of data and status buffers is written into the local memory on the CP from shared memory. The Transaction Error procedure reads the status buffer in local memory when returning status information to the application. As a result, the Read I/O Request procedure must be called prior to invoking the Transaction Error procedure.

2.3.1.16 Process Name:	Transaction is Bypassed
Inputs:	Transaction Identifier
Outputs:	Transaction Bypassed Flag
Requirements Reference:	I/O User Interface Functional Requirements , Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.4
Notes:	None

Description:

The I/O Request Completion function performs chain and transaction error processing. If a transaction was not executed because it was deselected or bypassed, then the I/O Request Completion function sets flags in shared memory to notify the I/O System Services on the CP. The Transaction Is Bypassed process checks whether or not a transaction is bypassed by reading the associated status flag.

The status information returned by the Transaction Is Bypassed procedure is only valid after the Read I/O Request procedure has been called due to the "Double Buffering" scheme. Since two sets of I/O buffers are used for interprocessor data communication, two sets of status buffers are required to maintain consistent data/status information (each data buffer must its own status buffer). When the Read I/O Request procedure is called, one set of data and status buffers is written into the local memory on the CP from shared memory. The Transaction Is Bypassed procedure reads the status buffer in local memory when returning status information to the application. As a result, the Read I/O Request procedure must be called prior to invoking the Transaction Is Bypassed procedure.

2.3.1.17 Process Name:	Chain Error
Inputs:	Chain Identifier
Outputs:	Chain Error Flag All Transactions are Bad Error Flag Chain Did Not Complete Error Flag Transaction Not Executed Flag Network Status Flag
Requirements Reference:	I/O User Interface Functional Requirements , Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.4
Notes:	None

Description:

The I/O Request Completion function performs chain and transaction error processing. If an error(s) occur in the execution of the I/O request or a transaction(s) has not been executed because it has been deselected or bypassed, then the I/O Request Completion function sets flags in shared memory to notify the I/O System Services on the CP. The Chain Error procedure checks for the occurrence of errors and deselected/bypassed transactions in a chain by reading the associated status flags.

The I/O Request Completion function records the status of the I/O networks on which an I/O request is executed (recorded when the I/O request is processed). The Chain Error procedure returns a field to the user giving the status of the associated I/O network at the time the chain was executed.

The status information returned by the Chain Error procedure is only valid after the Read I/O Request procedure has been called due to the "Double Buffering" scheme. Since two sets of I/O buffers are used for interprocessor data communication, two sets of status buffers are required to maintain consistent data/status information (each data buffer must its own status buffer). When the Read I/O Request procedure is called, one set of data and status buffers is written into the local memory on the CP from shared memory. The Chain Error procedure reads the status buffer in local memory when returning status information to the application. As a result, the Read I/O Request procedure must be called prior to invoking the Chain Error procedure.

2.3.1.18 Process Name:	I/O Request Has Overrun
Inputs:	I/O Request Identifier
Outputs:	Number of Overruns
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.5
Notes:	None

Description:

If an I/O request can not be executed when its scheduling requirements have been fulfilled, then a scheduling overrun occurs. The I/O Request Has Overrun process allows the application to determine if an overrun has occurred.

The occurrence of an overrun is detected by the Posting task (discussed in Section 3.2.1.1) associated with the I/O request. A parameter (integer number) in shared memory is updated to reflect the occurrence (or frequency of occurrences) of an overrun. This parameter is read from shared memory and returned to the application by the I/O Request Has Overrun process.

2.3.1.19 Process Name:	Write Initial I/O Request Data
Inputs:	I/O Request Identifier
Outputs:	State of the Locking Semaphore Static and Dynamic Output Data
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.2 I/O User Interface Software Specifications, Section 2.2.2.1
Notes:	None

Description:

The Write Initial I/O Request Data procedure performs a test and set operation on the semaphore that guards the output data buffer select variable for the I/O request. If the select region is unlocked, it selects an available buffer, unlocks the select region, and writes the static and dynamic output data for the entire I/O request into shared memory. If the buffer select region is locked, the procedure continues to perform the test and set operation until the region becomes unlocked or 100 test and set iterations pass. If 100 iterations pass and the select region is still locked, it is assumed that a fault has caused a deadlock situation (select region is locked but neither processor has control of it). If such a deadlock situation occurs, the procedure disregards the semaphore, determines the available buffer, resets the semaphore, sets the LOCKED parameter, and writes the output data.

After the output data has been written into shared memory, the output data buffer select variable must be updated to specify the available buffer with the most current data. The Write Initial I/O Request Data procedure performs the test and set process as previously described, and when the select region is unlocked, the buffer select variable (in shared memory) is set equal to the buffer into which the output data was written.

2.3.1.20 Process Name:	Wait Until IOP Completed
Inputs:	IOP Completed Flag
Outputs:	None
Requirements Reference:	I/O User Interface Functional Requirements, Section 2.1.3 I/O User Interface Software Specifications, Section 2.2.3.1
Notes:	None

Description:

Flags are used to synchronize the I/O System Services on the CP and the IOP. The IOP must wait for the CP to create and communicate the I/O request Specifications. Alternatively, the CP must wait for the IOP to initialize the I/O Services.

The Wait Until IOP Completed process allows the CP to wait until the IOP initializes the I/O Services.

2.3.1.21 Process Name: CP Completed

Inputs: None

Outputs: CP Completed Flag

Requirements Reference: I/O User Interface Functional Requirements, Section 2.1.3
I/O User Interface Software Specifications, Section 2.2.3.1

Notes: None

Description:

Flags are used to synchronize the I/O System Services on the CP and the IOP. The IOP must wait for the CP to create and communicate the I/O request specifications. Alternatively, the CP must wait for the IOP to initialize the I/O Services.

The CP Completed process sets a flag in shared memory to acknowledge the completion of I/O request creation process.

2.3.1.22 Process Name: Wait for Specification Received

Inputs: Ready for New Specification Flag

Outputs: None

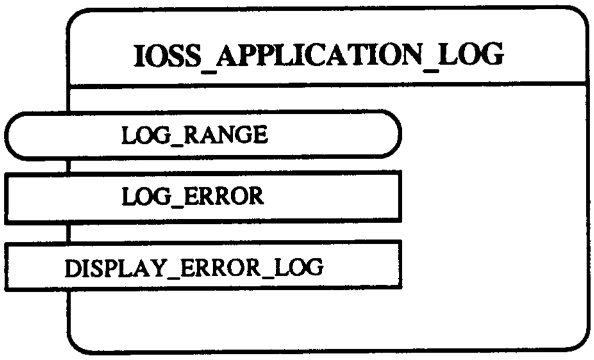
Requirements Reference: None

Notes: None

Description:

The Wait for Specification Received process polls the Ready for New Specification flag until it is set (true). The process is used to synchronize the I/O System Services on the CP and IOP during the interprocessor communication of the I/O request specifications.

2.3.2 I/O System Services Application Log



2.3.2.1 Process Name:	Log Error
Inputs:	Application Log Identifier Node Identifier Error Test String Error Description String
Outputs:	Entry in Application Log
Requirements Reference:	None
Notes:	None

Description:

The Log Error process logs the Application Log Identifier, Node Identifier, Error Test string, and Error Description string in an Application Log (identified by the Log ID). An Application Log is a 15 line cyclic log which is used to record information associated with the application processes, I/O request processing, and interprocessor communication.

2.3.2.2 Process Name:	Display Error Log
Inputs:	Application Log Identifier
Outputs:	None
Requirements Reference:	None
Notes:	None

Description:

The Display Error Log process displays the contents of the Application Log which is identified by the Application Log identifier.

2.4 I/O User Interface Data Dictionary

Chain ID Type: The range of possible values for a chain ID (0 - Maximum Chain ID).

ID Array : A variant record type available to the applications user for the creation of arrays of transactions or chains. The applications programmer must use this record to group transactions into a chain and chains into an I/O request.

I/O Request ID Type: The range of possible values for an I/O request ID (0 - Maximum IOR ID).

I/O Request Scheduling Record : A discriminate record available to the applications user for specifying scheduling parameters for an I/O request. The record always has fields indicating the completion event, I/O request time out, and priority of the I/O request. The record is discriminated on a two state variable indicating the type of I/O request (on demand or periodic). If the I/O request is periodic, then fields indicating its period, how it will be started, and how it will be stopped must also be provided.

Maximum Chain ID : An integer constant which specifies the maximum number of chains.

Maximum IOR ID : An integer constant which specifies the maximum number of I/O requests.

Maximum Transaction ID : An integer constant which specifies the maximum number of transactions.

Periodic Scheduling Record : A discriminate record available to the applications user for specifying start parameters for a periodic I/O request. The record is discriminated on a three state variable indicating how the periodic I/O request should be started (Start_On_Demand, Start_After_Delay, or Start_At_Absolute_Time). If the discriminate is Start_After_Delay or Start_At_Absolute_Time, then information concerning the delay or specific start time, respectively, must also be provided.

Transaction ID Type: The range of possible values for a transaction ID (0 - Maximum Transaction ID).

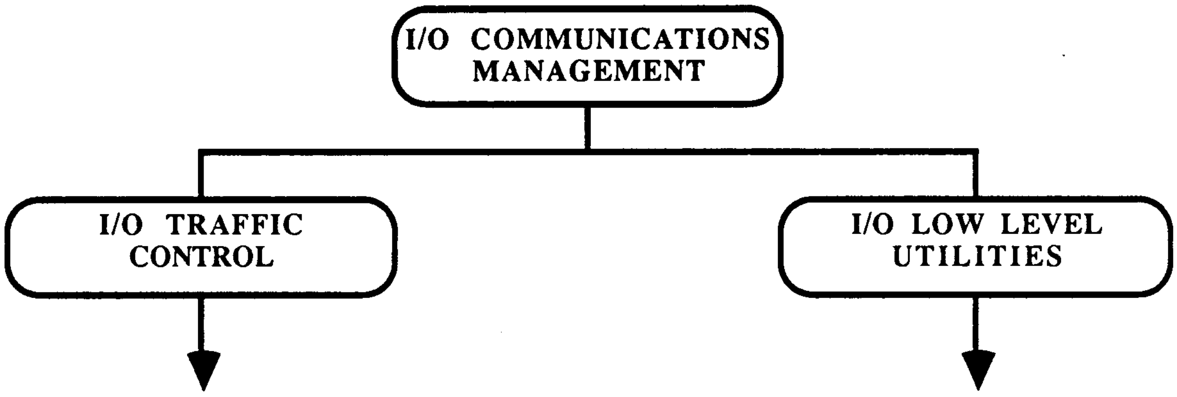
Transaction Information Record : A discriminate record available to the applications user for specifying I/O parameters for a transaction. The record always has fields indicating the destination DIU, number of output data bytes, type of output data (dynamic or static), and local CP address of the application output data buffer. The record is discriminated on a two state field indicating the type of transaction (input or output). If the transaction is an input transaction, then fields indicating the number of input data bytes, maximum number of errors before bypass, time out, and local CP address of the application input data buffer must also be provided.

3.0 I/O COMMUNICATIONS MANAGEMENT

The description of the I/O Communications Management is divided into three sections: Functional Description, Software Specifications, and Software Process Descriptions.

3.1 I/O Communications Management Functional Description

I/O Communications Management provides the processes necessary to control the flow of data between the GPC and the various I/O networks used by the GPC. This work is divided between two functions, I/O Traffic Control and I/O Low Level Utilities.

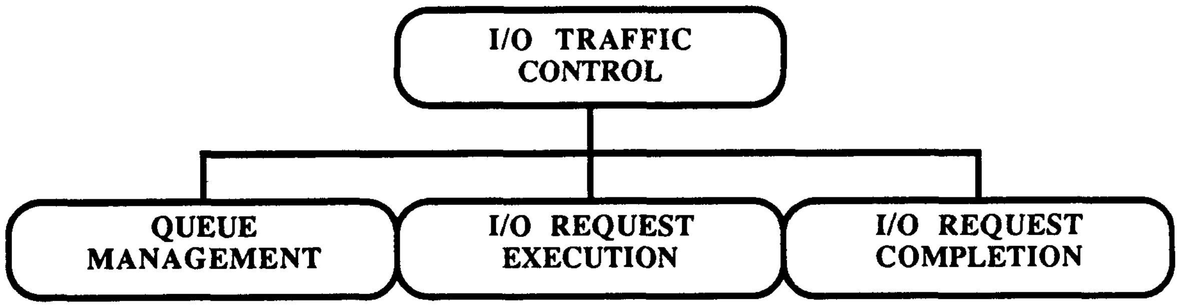


The function of the I/O Traffic Control includes the processing that must be done to place an I/O request in its proper place in the priority queue of a given network and to transfer data between shared memory and its correct location in the dual ported memory of the IOS. In addition, the I/O Traffic Control function coordinates the simultaneous execution of chains on an I/O service which has redundant networks and processes any errors detected during the execution of the I/O request.

The Low Level Utilities are responsible for congruently distributing inputs to the redundant channels of the GPC, voting output data, and screening input data for errors. This error processing involves error detection (including chain time out and byte count errors) and error logging. The Low Level Utilities function is also responsible for performing the correct physical to logical mapping for each network on the GPC.

3.1.1 I/O Traffic Control

The main responsibilities of the I/O Traffic Control process are to manage the I/O request queues associated with each I/O service accessible by the GPC, to cause chains in an I/O request to be executed, and to process any errors which occurred during chain execution. Accordingly, the I/O Traffic Control process is divided into three functions: Queue Management, I/O Request Execution, and I/O Request Completion.



3.1.1.1 Queue Management

I/O requests are conducted on an I/O service. These I/O services may be regional or local, and if they are local, they may involve a set of redundant networks. A GPC can only post I/O requests to I/O services to which it is connected. A list of all the I/O services accessible by a GPC is available in its local I/O database.

The Queue Management function initializes each I/O service that is connected to the GPC. The initialization of an I/O service means that the network and data structures associated with the service are initialized, the service is grown, and the tasks required by the service are activated. Accordingly, the Queue Management process activates a Queue Manager task for each I/O service, schedules a Posting task for each I/O request, and constructs a set of priority queues for each Queue Manager task.

The primary role of the Queue Management process is to create and manage a set of priority queues for each I/O service in the local database. Each priority queue holds a prioritized list of service requests (e.g. a network restoration request, spare element cycling request, or I/O request) for that I/O service. The Queue Management process must provide a way to post a service request to an I/O service, to supply the next service request which should execute on that service, and to indicate that no requests are pending.

The Queue Management process activates an I/O Posting task for each I/O request created by the application. A Posting task is a task that is scheduled on the IOP based on the scheduling requirements of the I/O request. When the scheduling requirements of the I/O request are met, the Posting task posts a service request to the correct priority queue of the appropriate I/O service.

A Queue Manager task is activated for each I/O service. This task controls all access to the I/O service. If I/O requests are pending, the Queue Manager task accepts the I/O request with the highest priority, invokes the I/O Request Execution function, and calls the I/O Request Completion process. If a fault in an I/O network causes errors to occur in the I/O request, then the Queue Manager task takes the appropriate network out of service and calls the network manager FDIR. If no I/O requests are pending, then the Queue Manager can accept service requests to cycle the spare components of the I/O service. In addition, the Queue Manager task controls the modification of the I/O chains (transaction selection/deselection) and the restoration of failed network elements.

3.1.1.2 I/O Request Execution

Each I/O service requires a Queue Manager task to control the execution of I/O requests on that service. Whenever an I/O service is "ready" or "idle" and any I/O requests have been posted to that service, the Queue Manager task will begin to process the pending I/O request which has the highest priority.

When an I/O request has just completed on a service, that service is designated as "ready"; that is, ready to begin another I/O request. If one or more I/O requests are pending at that service, the request with the highest priority will be started as soon as the service becomes ready. However, if no request is pending at a service when it becomes ready, the I/O service is considered "idle". An I/O request which is posted to an idle I/O service will be started immediately. This scheme is intended to achieve a high degree of I/O service utilization by eliminating unnecessary delays in starting I/O request execution.

An I/O service may utilize only one network or it may require a set of parallel redundant networks. Within each service, a given network may be in or out of service. Typically, a network will be out of service for one of two reasons: it has not yet been grown by its network manager or a fault(s) exists in the network causing it to be pulled out of service to allow FDIR activity to proceed. While a network is out of service, user chains will not be executed on the network but manager chains, of course, will be allowed. If the service is not redundant, user chains will not be executed until the network is back in service. However, if the service is redundant, the unfailed networks will remain in service while FDIR is conducted on the network which experienced errors. Thus, user chains in the unfailed networks will continue to execute at their normal rate, unimpeded by the repair activity of the FDIR processes. This scheme is intended to provide an application with an uninterrupted flow of I/O data even in the presence of hardware faults (when a redundant I/O service is part of the system).

Another aspect of I/O Request Execution is updating, whenever necessary, the chain program or set of redundant chain programs in dual ported memory. This is necessary to support the transaction selection/deselection option open to a user and the transaction bypassing feature available to the I/O System Services. The I/O System Services may bypass a particular transaction if it is repeatedly the source of errors and the application allows that transaction to be bypassed.

To bypass or deselect a transaction, the chain program is modified so that the IOS will skip over the set of instructions used to execute the deselected transaction. To select a transaction, the chain program is modified so that the IOS will not skip the relevant set of instructions. A chain program consists of a header and a linked list of transactions. The header contains instructions that affect the entire chain such as the polling priority and number of residual bits, and it ends with a branch instruction to the first transaction of the chain. Each transaction in the linked list is a sequence of IOS instructions which ends with a branch instruction to the first instruction of the next transaction or to an end of chain program (which causes the IOS to get ready for another command from the interface command register). To deselect a specific transaction N, the operand of the branch instruction of the transaction that precedes transaction N is modified to point to the transaction that succeeds transaction N. To select a specific transaction N, the operand of the branch instruction of the transaction that precedes transaction N is modified to point to transaction N, and the operand of transaction N's branch instruction is set equal to its preceding transaction's branch operand.

Another function of I/O Request Execution process is the transfer of output data from shared memory to dual ported memory (DPM) of the IOS. The output data for a transaction must be written to the DPM of the IOS prior to the execution of the command frame of the transaction, that part of a transaction which carries data from the GPC to the DIU. Command frames may be static or dynamic. Data for static command frames are updated only once, whereas data for dynamic command frames are updated each time the chain executes. When a chain contains only static command frames, the entire chain is designated as having static output. This designation saves processing time since each transaction does not need to be tested to determine whether or not it contains a dynamic command frame. Similarly, when a chain contains only dynamic command frames, the entire chain is designated as having dynamic output. Again, processing time is saved, since the data transfer can take place without a test of whether or not a transfer is necessary. Chains with both dynamic and static command frames are designated as mixed. This type of chain requires additional processing time and, where performance is a factor, should be avoided.

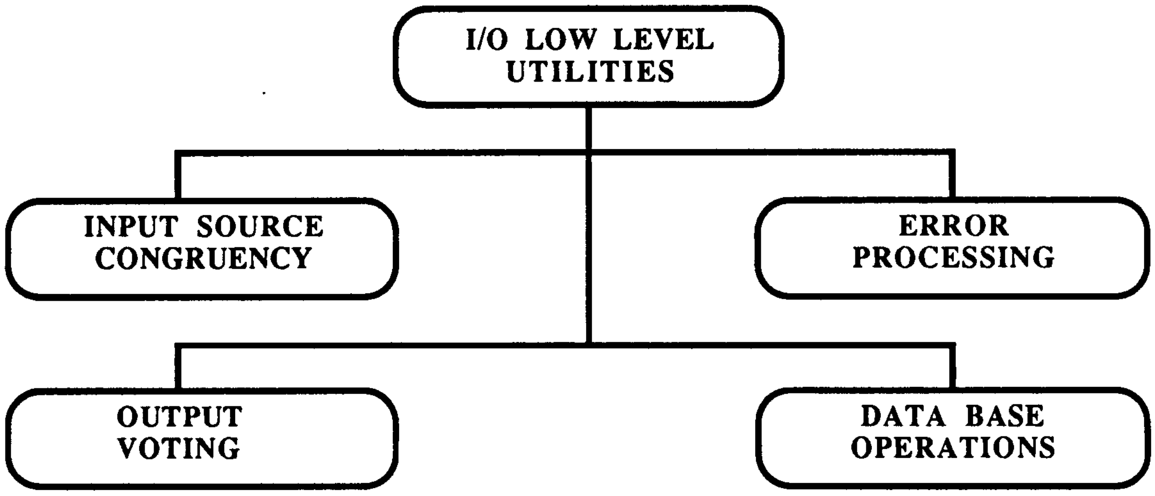
Once the output data and chain program for all the chains in the I/O request have been updated, the IOS is commanded to start the chains by means of a word written to the interface command register (ICR) of the IOS. In the case of redundant chains, this command must be written to more than one ICR and should be performed with the shortest possible delay between writes. If possible, the commands should be written simultaneously. If not, they should be written with consecutive machine instructions from the GPC. After starting the chains in the I/O request, the I/O Request Execution process is complete.

3.1.1.3 I/O Request Completion

The I/O Request Completion function is triggered after the I/O request time out (the length of time that the I/O request actively possesses an I/O network or networks during its execution) has expired. This processing involves chain completion processing for each chain of the I/O request. Chain completion processing entails chain error processing, chain error logging, transaction error processing, transaction error logging, and the transfer of transaction input data from DPM to shared memory. After the completion processing for the I/O request is finished, the application is signalled that the I/O request has completed and the data in shared memory is readable.

3.1.2 I/O Low Level Utilities

The I/O Low Level Utilities perform source congruency and error detection on inputs from external devices and consistent error free outputs to external devices. They are also responsible for the mapping of physical to logical devices (nodes, links, DIUS) and the assurance that a consistent network database exists for each I/O network attached to the GPC. The I/O Low Level Utilities are made up of four functional modules: Input Source Congruency, Output Voting, Error Processing, and Database Operations.



The I/O Low Level Utilities are used by both the I/O Redundancy Management and I/O Communications Management. They record and report communication errors to both functions.

3.2 The I/O Communications Management Software Specifications

As specified in Section 3.1, the I/O Communications Management is divided between two functions: I/O Traffic Control and I/O Low Level Utilities.

The I/O Traffic Control process is responsible for allocating and initializing the tasks that control the processing of the I/O requests. These tasks include the Posting tasks, priority queue processes, and Queue Manager tasks. The I/O Traffic Control function is also responsible for initializing the I/O services, executing the I/O requests, and I/O Request error processing.

The I/O Low Level Utilities are responsible for supporting source congruency on the inputs, voting on the outputs, and fault masking.

3.2.1 I/O Traffic Manager

As previously discussed in the I/O Communications Management Functional Requirements, the I/O Traffic Manager is divided into three sections: Queue Management, I/O Request Execution, and I/O Request Completion. The I/O Queue Management process is responsible for initializing the I/O services, activating the Posting tasks, and managing the priority queues. The I/O Request Execution function is responsible for execution of the chains of the I/O requests, while the I/O Request Completion function is responsible for the error processing and logging required by the I/O requests.

3.2.1.1 Queue Management

The Queue Management process accesses the local I/O database to determine the I/O services to which the GPC is connected. A Queue Manager task is allocated for each I/O service, and each task initializes its corresponding I/O service. The initialization of an I/O service involves the initialization of the IOSes, I/O networks, and priority queues of the service. The Queue Manager task loads the dual ported memory of the IOSes with the I/O chains and output data specified by the application. The task then waits for the associated I/O network Managers to grow the I/O networks of the service. After the networks are active (the state of each network is "in_service") and all of the I/O request specifications have been communicated to the IOP, the Queue Manager constructs and initializes the priority queues. In addition, it schedules the Spare Link Cycling task (if enabled by the application) and activates the Posting tasks. When the I/O service has been initialized, the Queue Manager sets a flag in shared memory to notify the I/O System Services on the CP.

The initialization of an IOS involves partitioning the program region of the IOS into a header section and a linked list of transaction modules. Each transaction section of the IOS is used by only one chain, whereas the header section is used by all chains. The initialization of the IOS program region entails the linking of the transaction sections to make the chains. The linking of the transactions is accomplished by modifying each transaction's branch instruction to point to the next transaction in the chain or to the end of the chain program.

The I/O requests are assigned a priority of 0 through 7 (where priority 0 is the lowest and priority 7 preempts priorities 0 through 6). An I/O request priority queue is constructed to control the processing of the I/O requests. The number of records in the queue is equal to the number of I/O requests that are created by the application for the I/O service. Each record has three fields: the I/O request ID, a boolean flag which indicates whether or not the scheduling requirements for the request have been met, and a pointer to the next record in the queue. The I/O request priority queue is a linked list of I/O request records and its ordering is based on the priorities of the I/O requests (the highest priority request is first, the lowest last, etc.).

The Queue Manager task must manage three queues of service requests for the I/O service. These queues are the I/O request priority queue (previously described), spare component cycling queue, and restoration queue. The Queue Manager task accepts service requests posted to each queue, removes service requests for processing, and indicates when a priority queue is empty. In addition, the task determines the next service request to be processed. The Queue Manager determines the request to be serviced in the following manner:

a) Process any preempted I/O requests.
b) Process the highest I/O request pending.
c) If an I/O request with a priority of 0 - 6 is being processed and an I/O request with a priority of 7 is posted, preempt the request being executed and process the priority 7 request.
d) If no I/O requests are pending and a cycle spare link command is posted, cycle the spare link.
e) If a spare link is being cycled and an I/O request is posted, preempt the cycling request and process the I/O request.
f) If no I/O requests are pending and a restore element command is posted, restore the network element.

Each I/O request that is created by the application has a corresponding Posting task that executes on the IOP. A Posting task is a task that is scheduled through the GPC Real Time Operating System based on the scheduling requirements of the I/O request. The Posting task is activated by the Queue Manager task during initialization and is blocked until its scheduling requirements of the I/O request are met. When scheduling requirements have been fulfilled, the Posting task sets an execution flag in the I/O request priority queue and calls the Queue Manager task (posts a service request). The task is then blocked until the Queue Manager task accepts the service call. After the I/O request has been accepted, the Posting task checks to see if an overrun has occurred, updates the overrun parameter in shared memory to reflect the number of overruns that have occurred, and loops back to again wait for scheduling requirements of the I/O request to be met.

After the priority queue process and posting tasks have been initialized, the Queue Manager task waits until a service request is posted. When more than one service request is pending, the task determines the next service request to be processed. If the service request accepted is an I/O request, the Queue Manager task invokes the I/O Request Execution function to execute the request and the I/O Request Completion function to process the corresponding input response (if necessary). Alternatively, if the service request accepted is a spare component cycling or restore command, the associated I/O network (or networks) is taken out of service and the corresponding network manager(s) is called to execute the request.

3.2.1.2 I/O Request Execution

The I/O Request Execution function is invoked by the Queue Manager task after an I/O request sent to the I/O service has been accepted. An I/O service may utilize one network or a set of parallel redundant networks. Each network is either in_service, temporarily_out_of_service, repaired, or permanently_out_of_service. The network is in_service during no fault operation. The network is temporarily_out_of_service when the network FDIR is invoked, while network components are restored, or during spare component cycling. The network is repaired after the network FDIR has completed, a network element has been restored, or a spare component has been cycled. The network is permanently_out_of_service when it is unreachable from the GPC. The network is set to in_service from repaired and to temporarily_out_of_service from in_service by the Queue Manager task. The network is set to repaired (or permanently_out_of_service) from temporarily_out_of_service by the network manager tasks. If the network is in_service or repaired, the chains can be executed on the network whereas, if the network is out_of_service (either temporarily or permanently), they can not be executed on the network.

The control flow of the I/O Request Execution function (along with the I/O Request Completion function) is illustrated in the Figure 11. The bold path depicts the no fault control path.

The I/O Request Execution function initially checks the transaction selection/deselection shared memory flags to determine if there are any transactions to be selected or deselected. If either of the flags is set, the Request Execution function follows the selection/deselection procedure outlined in Section 2.2.2.3. If the flag is not set, then the function begins processing the I/O request.

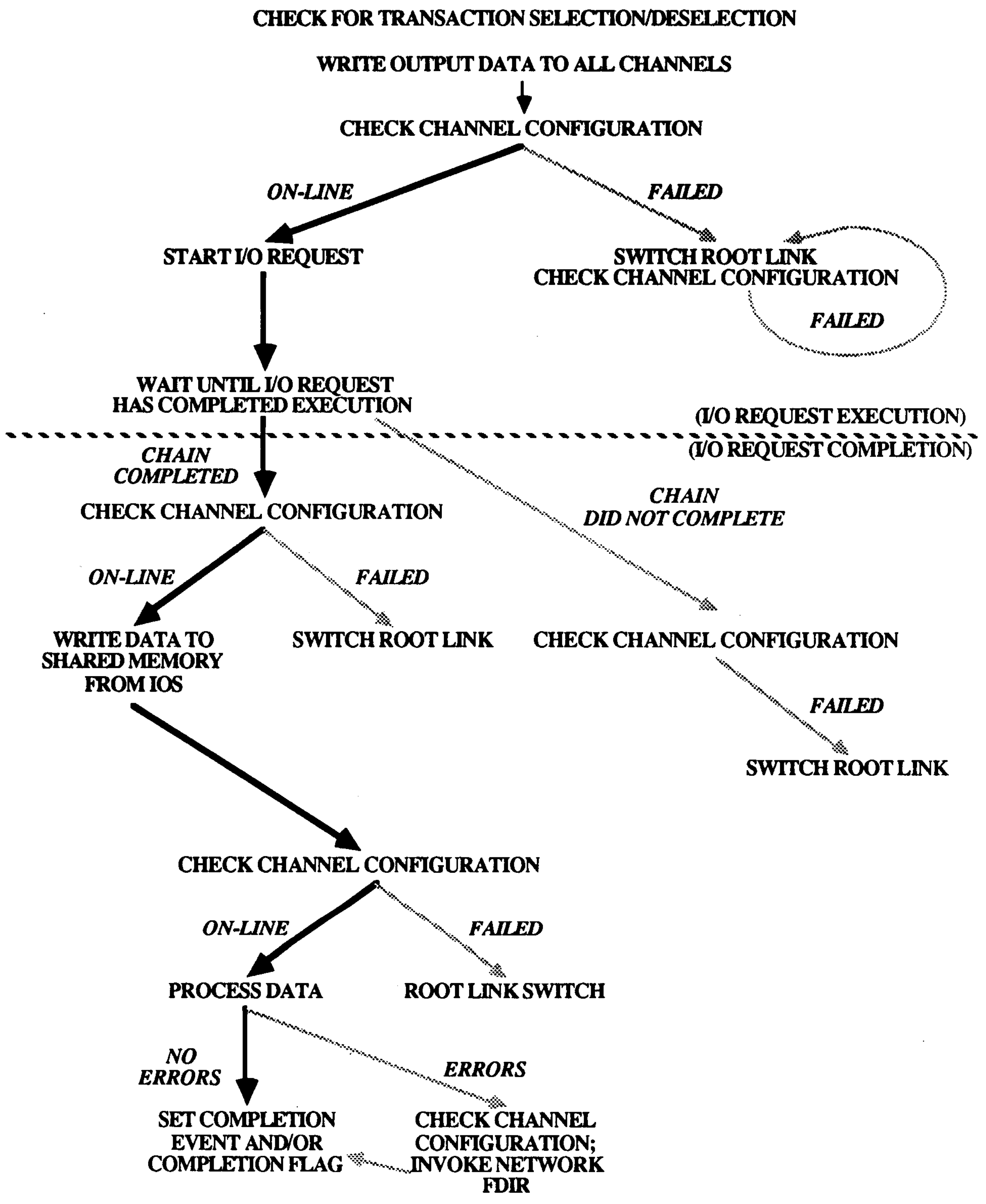

CHECK FOR TRANSACTION SELECTION/DESELECTION
WRITE OUTPUT DATA TO ALL CHANNELS
CHECK CHANNEL CONFIGURATION
ON-LINE
FAILED
START I/O REQUEST
SWITCH ROOT LINK
CHECK CHANNEL CONFIGURATION
FAILED
WAIT UNTIL I/O REQUEST HAS COMPLETED EXECUTION
(I/O REQUEST EXECUTION)
(I/O REQUEST COMPLETION)
CHAIN COMPLETED
CHECK CHANNEL CONFIGURATION
CHAIN DID NOT COMPLETE
ON-LINE
FAILED
WRITE DATA TO SHARED MEMORY FROM IOS
SWITCH ROOT LINK
CHECK CHANNEL CONFIGURATION
FAILED
SWITCH ROOT LINK
CHECK CHANNEL CONFIGURATION
ON-LINE
FAILED
PROCESS DATA
ROOT LINK SWITCH
NO ERRORS
ERRORS
SET COMPLETION EVENT AND/OR COMPLETION FLAG
CHECK CHANNEL CONFIGURATION; INVOKE NETWORK FDIR

Figure 11. Control Flow for I/O Request Execution and Completion

The processing of the I/O request begins with the loading of the IOSes of the I/O service with the dynamic output data (both the static and dynamic data must be written if the I/O request is being executed for the first time). To write the output data to the DPM of the IOS from shared memory, the I/O Request Execution process performs a test and set operation on the semaphore that guards the output data buffer select variable for the I/O request. If select region is unlocked, the procedure selects/locks the available buffer, unlocks the select region, calculates a sumcheck over the data, and writes the output data and sumcheck to the IOS. If the buffer select region is locked, the procedure continues to perform the test and set operation until the region becomes unlocked or 100 test and set iterations pass. If 100 iterations pass and the select region is still locked, it is assumed that a fault has caused a deadlock situation (region is locked but neither processor has control of it). If such a deadlock situation occurs, the procedure disregards the semaphore, determines the available buffer, resets the semaphore, calculates the sumcheck, and writes the output data and sumcheck to the IOS.

After the output data has been written from shared memory to IOS, the output data buffer select variable must be updated to unlock the output data buffer. The I/O Request Execution process performs the test and set process as previously described, and when the select region is unlocked, the buffer is unlocked by modifying the select variable in shared memory .

After the output data buffer is unlocked, the channel configuration is checked to see if the channel(s) executing the I/O request is on-line. If the channel is not on-line, the root link is switched until either an active root link is found or the corresponding network(s) is determined to be unreachable. If the channel is on-line, the I/O request is prepared for execution by setting up each chain of the I/O request. The set up procedure involves several steps:

a) The modification of the branch instruction of the header to point to the first transaction of the chain.
b) The modification of the poll instruction in the header of the chain to start immediately if the associated network is local or start with a poll otherwise.
c) Clear the Chain Status Register.
d) Set the Solicited Chain Pointer of the IOS to point to the header of the chain.

After the completion of the chain setup procedure, the chains of the I/O request can be executed. To execute a chain, an execute command is written to the Interface Command Register (ICR). In the case of two redundant chains, the execution commands are written in consecutive machine instructions. In the case of three or more redundant chains, the commands are written in a "tight" machine instruction loop.

As previously discussed in Section 3.2.1.1, I/O requests with a priority of 7 preempt lower priority I/O requests. A check for pending priority 7 requests is made before the output data for an I/O request is written to the IOS. In addition, a similar check is performed prior to the I/O request being executed. If a priority 7 I/O request is pending and a lower priority request is being processed, the state of the lower priority request is saved, the request is suspended, and the pending request is processed. When the processing of the priority 7 I/O request has completed, the state of the preempted request is restored and processing of the request is continued.

During the execution of an I/O request, the I/O request must actively possess the network for a specific amount of time; that is, the I/O request time out. Accordingly, the Queue Manager must wait for the I/O request time out to expire before it can invoke the I/O Request Completion function.

3.2.1.3 I/O Request Completion

The control flow for the I/O Request Completion function is illustrated in Figure 11. Initially, the I/O Request Completion function checks the channel configuration and the chain complete bit of the Chain Status Register. If the configuration check detects that the channel has failed, then the root link is switched. If the channel is on-line and the chain did complete, then the input data returned by the I/O request is written into shared memory from the DPM of the IOS (using a process similar to the Write I/O Request procedure - Section 2.3.1.6).

After the input data has been written to shared memory, the channel configuration is again checked. If the configuration check detects that the channel is down, then the active root link is switched. If the channel is on-line, then the data is processed for errors. The error detection process (I/O Low Level Utilities) checks for transmission, byte count, and sumcheck errors. The error logging process sets shared memory flags to notify the application of any chain and/or transaction errors that result. If any errors are detected, the channel configuration is checked and the data exchange tests are performed.

To support the I/O request preemption capability, a check for priority 7 requests is made before the input data is written to shared memory. If a priority 7 I/O request is pending and a lower priority request is being processed, the state of the lower priority request is saved, the request is suspended, and the pending request is processed. When the processing of the priority 7 I/O request has completed, the state of the preempted request is restored and processing of the request is continued.

After the error processing has been completed, the I/O completion flag for the I/O request is set. In addition, if the application selects the completion event option during the creation of the I/O request, the I/O completion event is signalled.

3.2.2 I/O Low Level Utilities

As discussed in Section 3.1.2, the I/O Low Level Utilities is made up of four functional modules: Input Source Congruency, Output Voting, Error Processing, and Database Operations.

The Input Source Congruency function is implicitly preformed by the shared data exchange hardware.

The Output Voting is performed by writing data (bytes or words) into a data exchange transmit register (see [1] for a detailed description). The I/O Communications Management and Redundancy Management processes use this register to vote all output data written to the IOSes.

The Error Processing capabilities of the GPC are used to determine I/O network errors. I/O network errors are determined by analyzing status information returned by the network nodes. This error information is used by the I/O Communications Management and Redundancy Management functions.

The Database Operation function is used by the I/O Communications Management and I/O Redundancy Management function to access information in the I/O database concerning the I/O services supported by the GPC site. It is responsible for the mapping of physical to logical devices (nodes, links, DIUS) and the assurance that a consistent network database exists for each I/O network attached to the GPC.

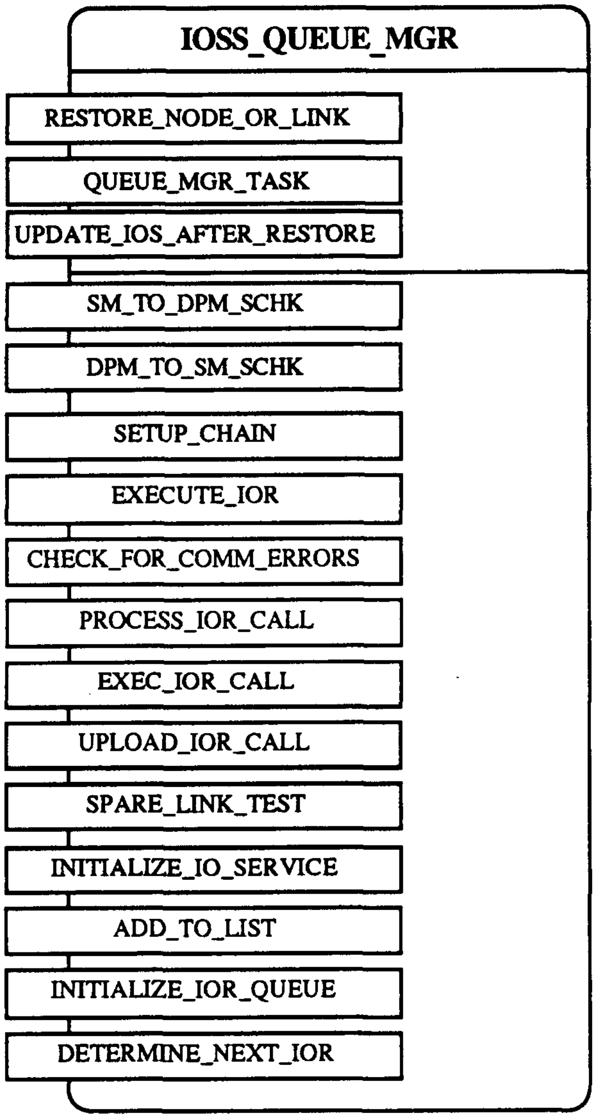
3.3 I/O Communications Management Software Process Descriptions

The I/O Communications Management Software Process Descriptions divide the description of the I/O Communications Management into functional packages. This section uses Booch diagrams and process descriptions to present the Software Specifications in more detail. The Booch diagrams are used to map the I/O Communications Management Software Specifications into functional packages, tasks, and subprograms. The process descriptions are used to describe these functional groups in detail.

The I/O Communications Management is divided into ten functional packages:

1) I/O System Services Queue Manager
2) I/O System Services IOP Construct I/O Requests
3) I/O System Services IOP Main Initialization
4) I/O System Services Communication of Specifications Task
5) I/O System Services Posting Tasks
6) I/O System Services IOP Powerup
7) I/O System Services Global Memory Utilities
8) I/O System Services Shared Memory Allocation
9) I/O System Services Dual Ported Memory Map
10) I/O System Services Private ID Types

3.3.1 I/O System Services Queue Manager



3.3.1.1 Process Name:	Restore Node or Link
Inputs:	I/O Network Identifier Node Identifier Port Identifier Type of Restoration
Outputs:	Confirmation that Restoration was Accepted Restoration Service Request
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

The Restore Node or Link process posts a restoration service request to the Queue Manager task. Initially, the process determines if the request is to restore a node or link. Next, the service request is posted to the restoration priority queue. If the Queue Manager is not processing a different service request, the restoration request is accepted and processed. If the Queue Manager is processing a different service request, the restoration request is retracted. The service request is retracted to avoid the possibility of the Restore Node or Link process being endlessly blocked if the Queue Manager is busy processing other jobs (restoration is a low priority job). If the service request is retracted, the process delays for one second and resubmits the request. If the Queue Manager can not accept the second request, the process delays again for a second time and then resubmits the request a third time. If the third request is not successful, then the Restore Node or link process notifies the calling process that the restoration was not accepted. If the restoration request was accepted in one of the three tries, then the process notifies the calling process that the Queue Manager accepted the request.

3.3.1.2 Process Name:	Queue Manager
Inputs:	I/O Request Hierarchical Database I/O Network Identifier I/O Transaction Identifier I/O Request Output Data I/O Transaction Selection or Deselection Record I/O Network Restoration Record Current Channel Network State Channel Configuration I/O Service Configuration
Outputs:	I/O Network Identifier Network State I/O Request Errors I/O Request Input Data Current Channel Connected Networks I/O Network Restoration Record IOS DPM Address Pointer I/O Transaction Identifier
Requirements Reference:	I/O Communications Management Functional Requirements, Section 3.1.1 I/O Communications Management Software Specification, Section 3.2.1.1
Notes:	None

Description:

The Queue Manager task is a process that controls the Queue Management, I/O Request Execution, and I/O Request Completion functions associated with a particular I/O service. The Queue Management function constructs and manages a set of priority queues and initializes the I/O service. The I/O Request Execution function writes the dynamic output data from shared memory to the appropriate IOSes, updates the chain headers, and executes the I/O request. The I/O Request Completion function checks for the occurrence of errors and writes the input data from the DPM of the IOS to shared memory.

The Queue Manager task is blocked until the I/O System Services on the CP has communicated all of the I/O request specifications to the IOP and the companion I/O request records have been constructed. The task then constructs an I/O request priority queue using the priorities assigned to the requests. The I/O requests with the higher priorities are placed closer to the front of the queue whereas those with the lower priorities are placed nearer to the bottom.

After the I/O request priority queue is created, the Queue Manager task creates the priority queues (FIFO) for the spare link cycling and restoration processes. In addition, the I/O chains are constructed, and the spare link cycling task is scheduled. The construction of the I/O chains involves the modification of branch instructions to link the transaction program modules into a chain module. This procedure also initializes the I/O data pointers and output data buffers. The spare link cycling task is scheduled as a background task, and it periodically posts a reconfiguration service request to the I/O service.

The I/O requests can not be scheduled until the I/O networks associated with the I/O service are grown. The growth of the I/O networks is performed by the I/O Network Managers, and the Queue Manager waits until these I/O networks are in_service. After the growth of the networks is complete, the Queue Manager task schedules an I/O Posting task for each I/O request created by the application. The Posting task is scheduled on the IOP in the manner specified by its corresponding I/O request. When the scheduling requirements of an I/O request have been met, the associated I/O Posting task sets an execution flag in the I/O request priority queue and calls the Queue Manager task (posts a service request).

After the I/O Posting tasks have been scheduled, the Queue Manager task sets a flag to notify the CP application tasks that the initialization of the I/O service has completed. At this time, the application tasks can start any on demand I/O requests.

The Queue Manager task controls access to the I/O service. Several processes contend for the service. These processes are:

1) The execution of an I/O request.
2) The cycling of a spare link.
3) The restoration of a node or link.

Each of the contending processes posts a service request to its associated queue when its scheduling requirements have been fulfilled. The service request to be next processed by the Queue Manager task is selected based on priority scheme described in Section 3.2.1.1.

If an I/O request is accepted by the Queue Manager task, then the I/O Request Execution function is called to process the request. The processing of an I/O request involves several steps. Initially, a check is made to determine if there are any transactions to be selected or deselected. Secondly, the dynamic output data is written from shared memory to the DPM of the IOSes (the static output data is also written if the I/O request has not been previously executed) and the chain headers are initialized to point to the first transaction of the I/O chains. After the headers have been updated, the I/O request is executed. The I/O request is started by writing a command to the Interface Command Register (ICR) of an IOS. In the case of two redundant chains, the execution instructions are written in consecutive machine instructions to two IOSes. In the case of three or more simultaneous chains, the commands are written in a tight machine instruction loop (to multiple IOSes) to minimize the delay between the writes while allowing the flexibility of variable size I/O requests. After the I/O request execution has started, the Queue Manager task waits until the I/O request time out has expired. At this time, the Queue Manager task invokes the I/O Request Completion process to perform error checking (byte count, transmission, and sumcheck), notify the application of the occurrence of errors, bypass the transactions which have exceeded their maximum error limitations, invoke the network FDIR (if necessary) and write the input data from the DPM of the IOSes to shared memory. After the input data has been written, the processing of the I/O request is finished, and the Queue Manager task notifies the associated CP application of the completion of the I/O request using a completion event and/or flag.

The processing of a spare link cycling service request involves two steps: the collection of node status from the I/O networks of the I/O service and the reconfiguration of the virtual circuit path if there are not any errors in the networks. The collection of node status information is required to determine if all of the expected network nodes are reachable prior to reconfiguration. If one or more of the nodes does not send its status or sends inconsistent status (inconsistent with the I/O network topology database), then a network fault is assumed to exist. If a fault is present in a network, the network is taken out of service and a repair request is sent to the associated I/O Network Manager task. In the case of redundant networks, this status collection is performed in near simultaneous manner similar to the execution of simultaneous I/O chains. If there are not any errors in the I/O service, then a spare link of each I/O network is made active by reconfiguring the virtual path. If redundant networks are involved, the reconfiguration is performed in near simultaneous manner.

The processing of a network element (link or node) restoration service request involves sending a restore request to the appropriate I/O Network Manager task.

The Queue Manager accepts service requests from the contending processes until it is descheduled.

3.3.1.3 Process Name:	Update IOS After Channel Restoration
Inputs:	None
Outputs:	Restore IOS Data Flags
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.2
Notes:	None

Description:

The Update IOS After Channel Restoration process sets the Restore IOS Data flags to notify the Queue Manager that the static output data for the I/O requests must be restored. The loss of a channel due to a fault may cause the data in the IOS to be corrupted. Since the static data is not updated each time an I/O request is processed, the data must be updated when the channel is restored.

3.3.1.4 Process Name:	Shared Memory to DPM with Sumcheck
Inputs:	Chain Identifier User Program and Data Pointer Output Data
Outputs:	Value of Locking Semaphore Voted Output Data Voted Sumcheck Voted Number of Data Bytes
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.2
Notes:	None

Description:

The Shared Memory to DPM process updates output buffers of the DPM of the IOSes. The output data has been written into shared memory by the application. The process updates the dynamic output data each time a chain is executed. The process updates the static output data prior to the first execution of the chain and after a chain error has occurred.

The Shared Memory to DPM process performs a test and set operation on the semaphore that guards the output data buffer select variable for the I/O request. If select region is unlocked, the procedure selects/locks the available buffer, unlocks the select region, calculates a sumcheck over the data, and writes/votes the output data and sumcheck to the IOS. If the buffer select region is locked, the procedure continues to perform the test and set operation until the region becomes unlocked or 100 test and set iterations pass. If 100 iterations pass and the select region is still locked, it is assumed that a fault has caused a deadlock situation (region is locked but neither processor has control of it). If such a deadlock situation occurs, the procedure disregards the semaphore, determines the available buffer, resets the semaphore, calculates a sumcheck over the data, and writes/votes the output data and sumcheck to the IOS.

After the output data has been read from shared memory, the output data buffer select variable must be updated to unlock the output data buffer. The Shared Memory to DPM process performs the test and set procedure as previously described, and when the select region is unlocked, the buffer is unlocked by modifying the select variable in shared memory .

3.3.1.5 Process Name:	DPM to Shared Memory with Sumcheck
Inputs:	Chain Identifier User Program and Data Pointer Input Data
Outputs:	Value of Locking Semaphore Array of Transaction Data Sumchecks Input Data
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.2
Notes:	None

Description:

The DPM to Shared Memory process reads input data from the DPM of the IOSes and updates the shared memory input data buffers. The input data is the response from a DIU to a request given by the application. The process updates the shared memory input data buffers each time an input response from an I/O request is received.

The DPM to Shared Memory process performs a test and set operation on the semaphore that guards the input data buffer select variable for the I/O request. If the select region is unlocked, it selects an available buffer, unlocks the select region, calculates a sumcheck over the data, resets the Old Data flag (notifying the CP that the input buffers have been updated since the previous data read), and writes the input data for the entire I/O request into shared memory. If the buffer select region is locked, the procedure continues to perform the test and set operation until the region becomes unlocked or 100 test and set iterations pass. If 100 iterations pass and the select region is still locked, it is assumed that a fault has caused a deadlock situation (select region is locked but neither processor has control of it). If such a deadlock situation occurs, the procedure disregards the semaphore, determines the available buffer, resets the semaphore, calculates a sumcheck over the data, resets the Old Data flag (notifying the CP that the input buffers have been updated since the previous data read), and writes the input data to shared memory.

After the input data has been written into shared memory, the input data buffer select variable must be updated to specify the available buffer with the most current data. The DPM to Shared Memory process performs the test and set procedure as previously described, and when the select region is unlocked, the buffer select variable (in shared memory) is set equal to the buffer into which the input data was written.

3.3.1.6 Process Name:	Setup Chain
Inputs:	Chain Identifier User Program and Data Pointer DPM Program Pointer
Outputs:	IOS Solicited Chain Pointer Chain Header Branch Instruction IOS Poll Instruction
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.2
Notes:	None

Description:

The Setup Chain process sets up the chain header section of the user program region in preparation for the execution of the chain. The process involves several steps:

a) The modification of the branch instruction of the header to point to the first transaction of the chain.
b) The modification of the poll instruction in the header of the chain to start immediately if the associated chain is local or start with a poll otherwise.
c) Clear the Chain Status Register.
d) Set the Solicited Chain Pointer of the IOS to point to the header of the chain.

3.3.1.7 Process Name:	Execute I/O Request
Inputs:	I/O Request Identifier
Outputs:	IOS Interface Command Register
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.2
Notes:	None

Description:

The Execute I/O Request process executes the I/O request after the output data buffers have been updated. To execute a chain, an execute command is written to the ICR of an IOS. To execute the simultaneous chains, execute commands are written to multiple ICRs with a minimal delay between the writes.

After the execute commands are written, the process waits until the I/O request time out has expired.

3.3.1.8 Process Name: Check for Communication Errors

Inputs:	User Data and Program Pointer DPM Program Pointer Array of Data Sumchecks Chain Identifier Active Root Link
Outputs:	Error in Chain Flag Shared Memory Error Flags
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.2
Notes:	None

Description:

The Check for Communication Errors process determines if an error(s) occurred during the execution of a chain and if so, it isolates and logs the type and location of the error(s). The process checks for the following types of errors:

1) Channel failure during chain execution.
2) Chain did not complete execution.
3) Transmission Errors.
4) Byte count errors.
5) Sumcheck errors.

The Check for Communication Errors process updates several shared memory flags to notify the application of the occurrence and type of errors. The process also determines if a transaction should be bypassed.

If a response from a DIU has been corrupted by errors and the DIU is reachable, then the Check for Communication Errors process calls the associated I/O Network Manager process to repair the network (except for sumcheck errors).

3.3.1.9 Process Name:	Process I/O Request Procedure Call
Inputs:	I/O Request Identifier I/O Request Output Data IOS Status Registers Network Status
Outputs:	Completion Flag Completion Event I/O Request Input Data
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.2
Notes:	None

Description:

The Process I/O Request procedure performs the I/O Request Execution and I/O Request Completion functions, as discussed in I/O Communications Management Functional Requirements and Software Specifications. In short, the process performs the following:

1) Checks for a transaction selection/deselection request.
2) Writes the output data from shared memory to DPM.
3) Sets up the chain header(s) for the I/O request.
4) Executes the I/O request.
5) Writes the input data from DPM to shared memory.
6) Checks for communication errors.
7) Notifies the application of the completion of the I/O request via event and/or flag.

3.3.1.10 Process Name:	Execute I/O Request Procedure Call
Inputs:	I/O Request Identifier IOS Status Registers Network Status
Outputs:	Completion Flag Completion Event I/O Request Input Data
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.2
Notes:	None

Description:

During the processing of an I/O request, checks are made to determine if the I/O request being processed should be preempted. The I/O request should be preempted if its priority is between 0 and 6 and a priority 7 I/O request is pending. If an I/O request is preempted, its state is recorded to avoid unnecessary reprocessing. The Execute I/O Request procedure is invoked when an I/O request is preempted before the it is executed. In short, the process performs the following:

1) Sets up the chain header(s) for the I/O request.
2) Executes the I/O request.
3) Writes the input data from DPM to shared memory.
4) Checks for communication errors.
5) Notifies the application of the completion of the I/O request via event and/or flag.

3.3.1.11 Process Name:	Upload I/O Request Procedure Call
Inputs:	I/O Request Identifier IOS Status Registers Network Status
Outputs:	Completion Flag Completion Event I/O Request Input Data
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.2
Notes:	None

Description:

During the processing of an I/O request, checks are made to determine if the I/O request being processed should be preempted. The I/O request should be preempted if its priority is between 0 and 6 and a priority 7 I/O request is pending. If an I/O request is preempted, its state is recorded to avoid unnecessary reprocessing. The Upload I/O Request procedure is invoked when an I/O request is preempted before the input response from the I/O request is processed. In short, the process performs the following:

1) Writes the input data from DPM to shared memory.
2) Checks for communication errors.
3) Notifies the application of the completion of the I/O request via event and/or flag.

3.3.1.12 Process Name:	Spare Link Test
Inputs:	Spare Link Test Count Channel Configuration I/O Service Database Network Status IOS Status Registers
Outputs:	None
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

The Spare Link Test process controls the cycling of the spare links of the network(s) of an I/O Service.

The Spare Link Cycling procedure involves two steps: the collection of node status from the I/O networks of the I/O Service and the reconfiguration of the virtual circuit path if there are not any errors in the networks. The collection of node status information is required to determine if all of the expected network nodes are reachable prior to reconfiguration. If one or more of the nodes does not send its status or sends inconsistent status (inconsistent with the I/O network Topology database), then a network fault is assumed to exist. If a fault is present in a network, the network is taken out of service and a repair request is sent to the associated I/O Network Manager task. In the case of redundant networks, this status collection is performed in near simultaneous manner similar to the execution of simultaneous I/O chains. If there are not any errors in the I/O Service, then a spare link of each I/O network is made active by reconfiguring the virtual path. If redundant networks are involved, the reconfiguration is performed in near simultaneous manner.

3.3.1.13 Process Name:	Initialize I/O Service
Inputs:	Connected Networks Network State
Outputs:	None
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

The Initialize I/O Service process initializes the I/O Service associated the Queue Manager task. The process involves constructing the user chains in the IOS, scheduling the spare link testing task, waiting for the networks to be grown by the I/O network managers, and scheduling the Posting tasks.

3.3.1.14 Process Name:	Add to List
Inputs:	I/O Request Identifier
Outputs:	I/O Request Priority Queue
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

The Add to List process adds the designated I/O request to the I/O request priority queue. The position of the I/O request in the queue depends on the priority assigned to it by the application. The I/O request priority queue is used to determine the next service request to be processed.

3.3.1.15 Process Name:	Initialize I/O Request Queue
Inputs:	I/O Request Identifiers
Outputs:	I/O Request Priority Queue
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

The Initialize I/O Request Queue process initializes the I/O request priority queue by ordering the I/O requests and invoking the Add to List procedure for each request. The position of the I/O request in the queue depends on the priority assigned to it by the application. The I/O request priority queue is used to determine the next service request to be processed.

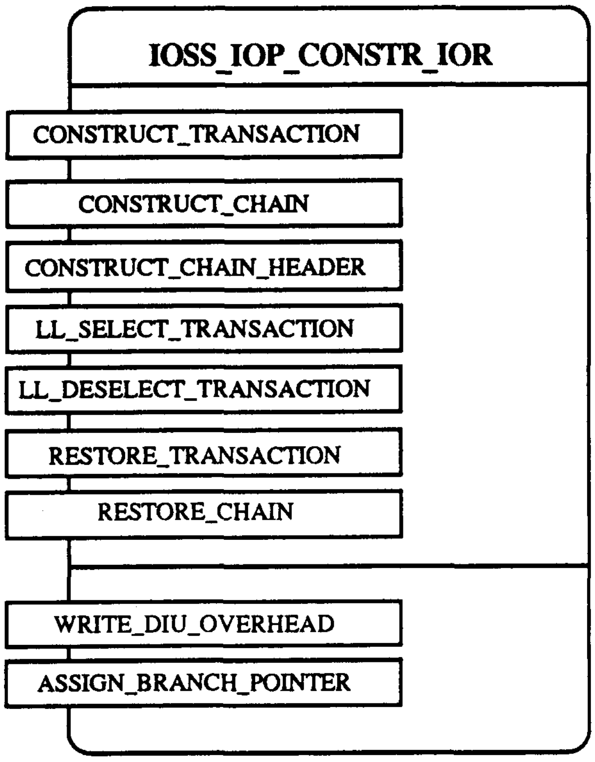
3.3.1.16 Process Name:	Determine Next I/O Request
Inputs:	I/O Request Priority Queue
Outputs:	I/O Request Identifier Queue Empty Flag
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

The Determine Next I/O Request process determines which pending I/O request should be next processed. The process parses the I/O request priority queue and returns the first pending request to the Queue Manager for processing. Since the priority queue was ordered from highest to lowest priority, the first pending entry has the highest priority.

It is possible for the I/O request priority queue to be empty. Accordingly, the process returns a boolean flag to notify the Queue Manager that no I/O requests are pending.

3.3.2 I/O System Services IOP Construct I/O Requests



3.3.2.1 Process Name: Construct Transaction

Inputs: IOS User Program and Data Pointer
Transaction Pointers

Outputs: Transaction Output Data
Transaction Branch Instruction

Requirements Reference: I/O Communications Management Software Specifications, Section 3.2.1.1

Notes: None

Description:

To execute the I/O requests created by the application, the IOSes have to be initialized. The program region of the IOS is partitioned into a header section and a linked list of transaction sections. Each transaction section of the IOS is used by only one chain, whereas the header section is used by all chains. The construction of the IOS program region entails the linking of the transaction sections to make the chains. The linking of the transactions is accomplished by modifying each transaction's branch instruction to point to the next transaction in the chain or to the end of chain program.

The Construct Transaction process initializes one transaction region in the DPM of a specific IOS. In addition to modifying the branch instruction of the transaction module, the Construct Transaction process allocates the I/O data buffers necessary to satisfy the transaction I/O requirements. The process also initializes the output data buffer and the input and/or output data buffer pointers.

3.3.2.2 Process Name:	Construct Chain
Inputs:	IOS User Program and Data Pointer IOS Network Manager Pointer Chain Pointer
Outputs:	Chain Header Branch Instruction
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

To execute the I/O requests created by the application, the IOSes have to be initialized. The program region of the IOS is partitioned into a header section and a linked list of transaction sections. Each transaction section of the IOS is used by only one chain, whereas the header section is used by all chains. The construction of the IOS program region entails the linking of the transaction sections to make the chains. The linking of the transactions is accomplished by modifying each transaction's branch instruction to point to the next transaction in the chain or to the end of chain program.

The Construct Chain process links a set of transaction regions into a chain. This procedure involves calling the Construct Transaction process for each transaction in the chain.

The Construct Chain process also records the address of the first transaction of the chain which is necessary to execute the I/O request.

3.3.2.3 Process Name:	Construct Chain Header
Inputs:	IOS User Program and Data Pointer Chain Pointer
Outputs:	Chain Header Branch Instruction Solicited Chain Pointer
Requirements Reference:	I/O Communications Management Functional Requirements, Section 3.1.1.3
Notes:	None

Description:

In order to execute an I/O request, the Solicited Chain Pointer register of the IOSes must be initialized to pointer to the header of the chains. Since the header section of the IOS program region is used by all chains, the branch instruction of the header section must be modified to point to the first transaction of the chain prior to executing the I/O request. This procedure is completed by the Construct Chain Header process.

3.3.2.4 Process Name:	Low Level Select Transaction
Inputs:	IOS User Program and Data Pointer for Writing to IOS IOS User Program and Data Pointer for Reading from IOS Transaction Pointer
Outputs:	Transaction Branch Instructions
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.2.3
Notes:	None

Description:

The Low Level Select Transaction process is called by the Select Transaction procedure and completes the low level procedures required to select a transaction. The Low Level Select Transaction process determines the preceding and succeeding transactions to the transaction to be selected. The preceding transaction's branch instruction is modified to point to the selected transaction while the branch instruction of the selected transaction is modified to point to the succeeding transaction.

If the selected transaction is the first transaction of the chain, then its address is recorded as the new chain header address (the chain header address is the address written into the branch instruction of the header section of the IOS program region prior to executing the chain). Alternatively, if the selected transaction is the last transaction in the chain, then its branch instruction is modified to point to the end of chain program (the end of chain program is a sequence of instructions which saves several registers and performs the transition from solicited to unsolicited mode).

3.3.2.5 Process Name:	Low Level Deselect Transaction
Inputs:	IOS User Program and Data Pointer for Writing to IOS IOS User Program and Data Pointer for Reading from IOS Transaction Pointer
Outputs:	Transaction Branch Instructions
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.2.3
Notes:	None

Description:

The Low Level Deselect Transaction process is called by the Deselect Transaction procedure and completes the low level procedures required to deselect a transaction.

The Low Level Deselect Transaction process determines the preceding transaction to the transaction to be selected and then modifies its branch instruction to point to the transaction that succeeds the transaction to be deselected. If the deselected transaction is the first transaction of the chain, then the address of its succeeding transaction is recorded as the new chain header address (the chain header address is the address written into the branch instruction of the header section of the IOS program region prior to executing the chain). Alternatively, if the deselected transaction is the last transaction in the chain, then the branch instruction of its preceding transaction is modified to point to the end of chain program (the end of chain program is a sequence of instructions which saves several registers and performs the transition from solicited to unsolicited mode).

3.3.2.6 Process Name:	Restore Transaction
Inputs:	IOS User Program and Data Pointer Transaction Pointers
Outputs:	Transaction Output Data Transaction Branch Instruction
Requirements Reference:	None
Notes:	None

Description:

The Restore Transaction process initializes one transaction region in the DPM of a specific IOS. In addition to modifying the branch instruction of the transaction module, the process initializes the input and/or output data buffer pointers.

3.3.2.7 Process Name:	Restore Chain
Inputs:	IOS User Program and Data Pointer IOS Network Manager Pointer Chain Pointer
Outputs:	None
Requirements Reference:	None
Notes:	None

Description:

When a channel is restored from a failed state, the user program region of the IOSes of that channel has to be restored. The Restore Chain process restores a chain in an IOS by linking a set of transaction regions into a chain (after the IOS is reinitialized). This procedure involves calling the Restore Transaction process for each transaction in the chain.

C-2

3.3.2.8 Process Name:	Write DIU Overhead
Inputs:	IOS User Program and Data Pointer Transaction Pointer
Outputs:	DIU Address Encoded DIU Address Residual Bits
Requirements Reference:	None
Notes:	None

Description:

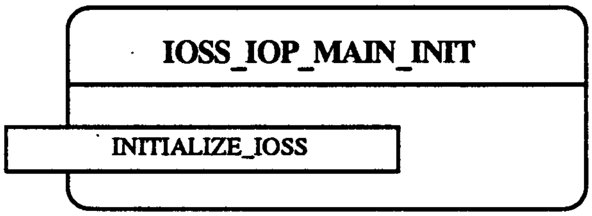
The Write DIU Overhead process initializes the data region in an IOS with the DIU address, encoded DIU address, and residual bits for a transaction.

3.3.2.9 Process Name:	Assign Branch Pointer
Inputs:	IOS User Program and Data Pointers Transaction Identifiers
Outputs:	Transaction Branch Instruction
Requirements Reference:	None
Notes:	None

Description:

The Assign Branch Pointer process bypasses the designated transaction by modifying the branch instruction of the preceding transaction.

3.3.3 I/O System Services Main Initialization

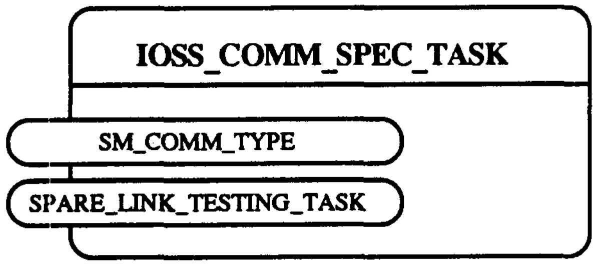


3.3.3.1 Process Name:	Initialize IOSS
Inputs:	None
Outputs:	Queue Manager Start Command
Requirements Reference:	None
Notes:	None

Description:

The Initialize IOSS process controls the initialization of the I/O Services. The process calls the Powerup_IOP procedure to schedule an IOP task to allow the communication of the I/O request specifications. The process waits for the CP to complete the communication of the specifications and then schedules the Queue Manager task which initializes the I/O Service.

3.3.4 I/O System Services Communication of Specifications Task



3.3.4.1 Process Name:	Shared Memory Communication Task Type
Inputs:	None
Outputs:	Companion Transaction Records Companion Chain Records Companion I/O Request Records
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.1
Notes:	None

Description:

The Shared Memory Communication task reads the I/O request specifications from shared memory and creates the IOP database of companion records.

The task is scheduled on the IOP through the GPC Real Time Operating System as an on demand process. The task is synchronized with the Create_Transaction, Create_Chain, and Create_IOR processes which reside on the CP. The interprocessor synchronization is achieved using an event and a flag. The event is used to signal that a transaction, chain, or I/O request record has been written into shared memory by the I/O System Services on the CP and can be read by the I/O System Services on the IOP. The flag notifies the CP that the IOP has read the record.

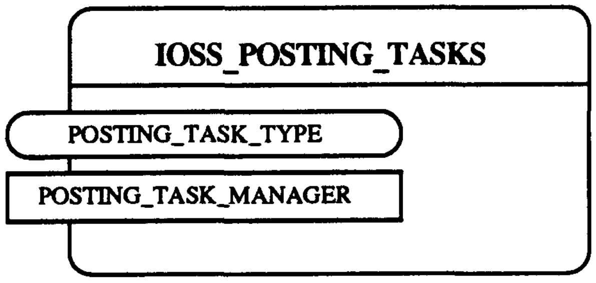
After being signaled by the CP, the IOP Shared Memory Communication task accesses shared memory and determines the type of record (transaction, chain, I/O request) being communicated. The task then reads the specification record, allocates memory for the companion record, and initializes the companion record. Finally, the task notifies the I/O System services on the CP that the companion record is created and that another record can be communicated.

3.3.4.2 Process Name:	Spare Link Test Task
Inputs:	None
Outputs:	Spare Link Cycle Service Request
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

The Spare Link Test task posts Spare Link Cycling service requests to the Queue Manager. It is scheduled through the GPC Real Time Operating System as a periodic task by the Spare Link Test Scheduling procedure.

3.3.5 I/O System Services Posting Tasks



3.3.5.1 Process Name:	Posting Task Type
Inputs:	I/O Request Identifier
Outputs:	Service Request for I/O Request Execution Number of Overruns I/O Request Preemption Flag Spare Link Test Preemption Flag
Requirements Reference:	I/O Communications Management Functional Requirements, Section 3.1.1.1 I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

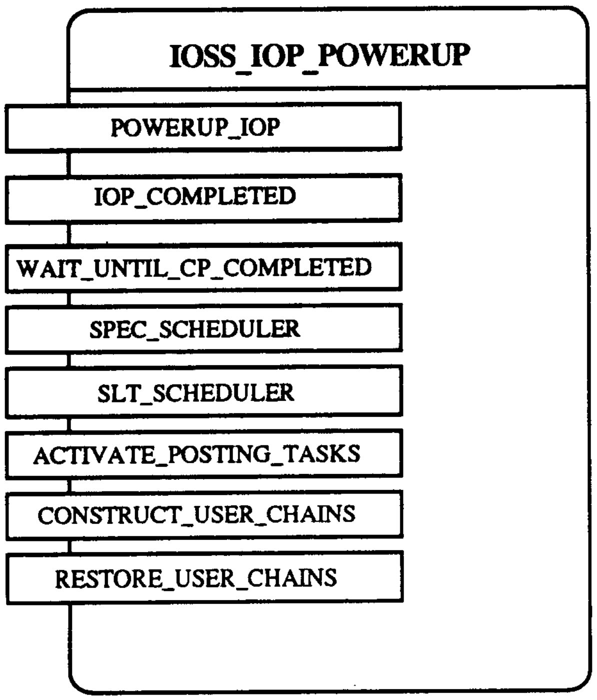
Each I/O request that is created has a corresponding Posting task on the IOP. A Posting task is a task that is scheduled through the GPC Real Time Operating System based on the scheduling requirements of the I/O request. The Posting task is activated by the Queue Manager task during initialization and is blocked until the scheduling requirements of the I/O request are met. When scheduling requirements have been fulfilled, the Posting task sets its execution flag in the I/O request priority queue and calls the Queue Manager task (posts a service request). The task is then blocked until the Queue Manager task accepts the call. After the I/O request has been accepted, the Posting task checks to see if it has overrun, updates the overrun parameter in shared memory to reflect the number of overruns that occurred, and loops back to again wait for its scheduling requirements to be met.

3.3.5.2 Process Name:	Posting Task Manager
Inputs:	I/O Request Identifier
Outputs:	None
Requirements Reference:	I/O Communications Management Functional Requirements, Section 3.1.1.1 I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

The Posting Task Manager allocates a Posting task for a particular I/O request and schedules it based on the scheduling requirements of the I/O request.

3.3.6 I/O System Services IOP Powerup



3.3.6.1 Process Name:	Powerup IOP
Inputs:	None
Outputs:	None
Requirements Reference:	None
Notes:	None

Description:

The Powerup IOP process calls the Specifications Scheduler procedure to schedule the IOP Shared Memory Communication task that accepts the I/O request specifications.

3.3.6.2 Process Name: IOP Completed

Inputs: None

Outputs: IOP Completed Flag

Requirements Reference: I/O User Interface Functional Requirements, Section 2.1.3
I/O User Interface Software Specifications, Section 2.2.3.1

Notes: None

Description:

Flags are used to synchronize the I/O System Services on the CP and the IOP. The IOP must wait for the CP to create and communicate the I/O request specifications. Alternatively, the CP must wait for the IOP to initialize the I/O services.

The IOP Completed process sets a flag in shared memory to acknowledge the completion of I/O service initialization process.

3.3.6.3 Process Name: Wait Until CP Completed

Inputs: CP Completed Flag

Outputs: None

Requirements Reference: I/O User Interface Functional Requirements, Section 2.1.3
I/O User Interface Software Specifications, Section 2.2.3.1

Notes: None

Description:

Flags are used to synchronize the I/O System Services on the CP and the IOP. The IOP must wait for the CP to create and communicate the I/O request specifications. Alternatively, the CP must wait for the IOP to initialize the I/O services.

The Wait Until CP Completed process polls the CP Completed flag in shared memory waiting for the I/O System Services on the CP to finish communicating the I/O request specifications.

3.3.6.4 Process Name:	Specifications Scheduler
Inputs:	None
Outputs:	None
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.1.1
Notes:	None

Description:

The Specifications Scheduler process uses the GPC Real Time Operating System to schedule the IOP Shared Memory Communication task. The Shared Memory Communication task interacts with the CP to transfer the I/O request specifications from the CP to the IOP.

3.3.6.5 Process Name:	Spare Link Test Scheduler
Inputs:	None
Outputs:	None
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.1
Notes:	None

Description:

The Spare Link Test Scheduler process uses the GPC Real Time Operating System to schedule a Spare Link Test task. The Spare Link Test task posts spare link cycling service requests to the Queue Manager task.

3.3.6.6 Process Name: Activate Posting Tasks

Inputs: Number of I/O Requests

Outputs: None

Requirements Reference: I/O Communications Management Software Specifications, Section 3.2.1.1

Notes: None

Description:

The Activate Posting Tasks process calls the Posting Task Manager procedure to schedule a Posting task for each I/O Request.

3.3.6.7 Process Name: Construct User Chains

Inputs: Number of I/O Chains

Outputs: None

Requirements Reference: None

Notes: None

Description:

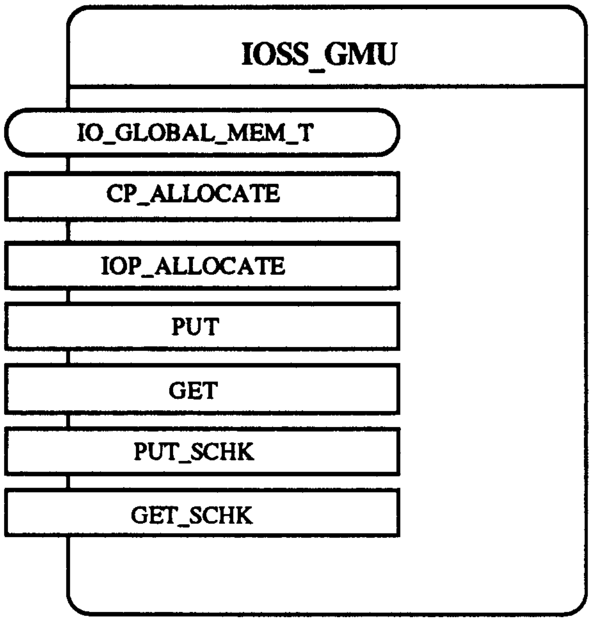
The Construct User Chains process calls the Construct Chain procedure to initialize an IOS program region for each I/O Chain created by the application.

3.3.6.8 Process Name:	Restore User Chains
Inputs:	Number of I/O Chains Deselected Transactions
Outputs:	None
Requirements Reference:	None
Notes:	None

Description:

The Restore User Chains process calls the Restore Chain procedure to restore an IOS program region (after a channel fails and returns) for each I/O Chain created by the application.

3.3.7 I/O System Services Global Memory Utilities



3.3.7.1 Process Name:	CP Allocate
Inputs:	Number of Bytes
Outputs:	Address of Buffer in Shared Memory
Requirements Reference:	None
Notes:	None

Description:

The CP Allocate process reserves a region in shared memory for an I/O buffer. The process returns the address of the buffer to the calling process.

3.3.7.2 Process Name:	IOP Allocate
Inputs:	Shared Memory Address Record
Outputs:	Address of Buffer in Shared Memory
Requirements Reference:	None
Notes:	None

Description:

The IOP Allocate process extracts the address of a shared memory I/O buffer from a shared memory address record. The address record is communicated from the CP to the IOP so that both processors know the location of the buffer.

3.3.7.3 Process Name:	Put
Inputs:	Number of Bytes Destination Address (private type) Source Address
Outputs:	None
Requirements Reference:	None
Notes:	None

Description:

The Put process moves and votes the designated number of bytes from the source address to the destination address.

3.3.7.4 Process Name: Get

Inputs:
Number of Bytes
Destination Address
Source Address (private type)

Outputs: None

Requirements Reference: None

Notes: None

Description:

The Get process moves the designated number of bytes from the source address to the destination address.

3.3.7.5 Process Name: Put with Sumcheck

Inputs:
Number of Bytes
Destination Address (private type)
Source Address

Outputs: Sumcheck

Requirements Reference: None

Notes: None

Description:

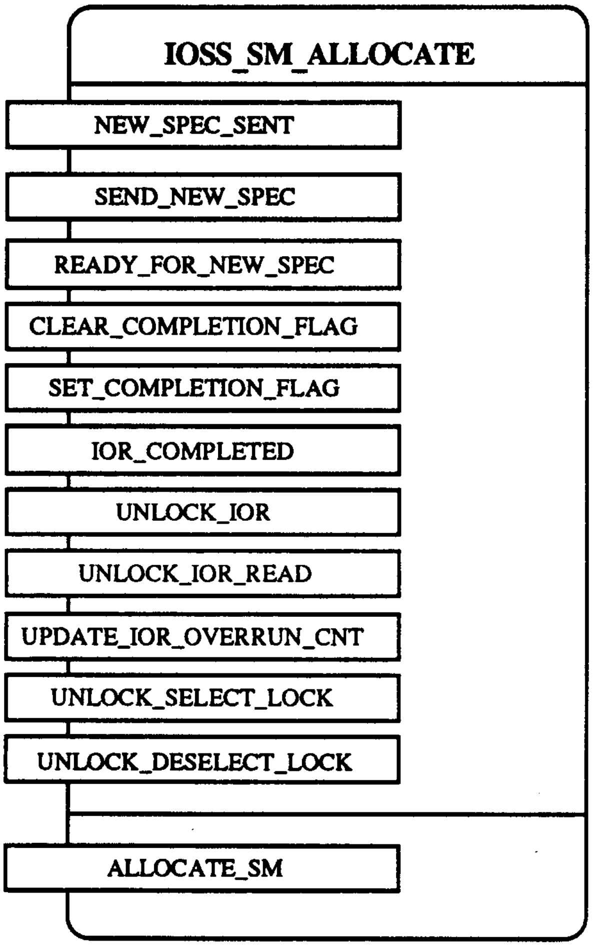
The Put With Sumcheck process moves and votes the designated number of bytes from the source address to the destination address while calculating a modulus 256 sumcheck over the data.

3.3.7.6 Process Name:	Get with Sumcheck
Inputs:	Number of Bytes Destination Address Source Address (private type)
Outputs:	Sumcheck
Requirements Reference:	None
Notes:	None

Description:

The Get With Sumcheck process moves the designated number of bytes from the source address to the destination address while calculating a modulus 256 sumcheck over the data.

3.3.8 I/O System Services Shared Memory Allocation



3.3.8.1 Process Name:	New Specification Sent
Inputs:	None
Outputs:	Flag for CP/IOP Interprocessor Communication of I/O Requests
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.1.1
Notes:	None

Description:

The New Specification Sent process resets a flag in shared memory to coordinate the communication of the transaction, chain, and I/O request specifications from the CP to the IOP. The process is used by an I/O System Services CP process to synchronize the interprocessor communication.

3.3.8.2 Process Name:	Send New Specification
Inputs:	None
Outputs:	Flag for CP/IOP Interprocessor Communication of I/O Requests
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.1.1
Notes:	None

Description:

The Send New Specification process sets a flag in shared memory to coordinate the communication of the transaction, chain, and I/O request specifications from the CP to the IOP. The process is used by the IOP Shared Memory Communication task to synchronize the interprocessor communication.

3.3.8.3 Process Name:	Ready for New Specification
Inputs:	Flag for CP/IOP Interprocessor Communication of I/O Requests
Outputs:	Flag for CP/IOP Interprocessor Communication of I/O Requests
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.1.1
Notes:	None

Description:

The Ready for New Specification process reads a flag in shared memory used to coordinate the communication of transaction, chain, and I/O request specifications. The process is used by a CP process to determine if the IOP is able to receive more information.

3.3.8.4 Process Name:	Clear Completion Flag
Inputs:	I/O Request Identifier
Outputs:	I/O Request Completion Flag
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.3
Notes:	None

Description:

The Clear Completion Flag process resets the shared memory completion flag associated with the designated I/O request identifier. This process is used by a CP application task to clear a completion flag prior to polling it.

3.3.8.5 Process Name:	Set Completion Flag
Inputs:	I/O Request Identifier
Outputs:	I/O Request Completion Flag
Requirements Reference:	I/O Communications Management Software Specifications, Section 3.2.1.3
Notes:	None

Description:

The Set Completion Flag process sets the shared memory completion flag associated with the designated I/O request identifier. This process is used by an IOP I/O Request Completion process to notify the CP application that the I/O request has been processed.

3.3.8.6 Process Name:	I/O Request Completed
Inputs:	I/O Request Identifier
Outputs:	I/O Request Completion Flag
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.3.1
Notes:	None

Description:

The I/O Request Completed process reads the shared memory completion flag associated with the designated I/O request identifier. This process is used by a CP application task to determine if the I/O request has been completely processed.

3.3.8.7 Process Name:	Unlock I/O Request
Inputs:	I/O Request Identifier
Outputs:	I/O Request Semaphore for Output Buffers
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.2.1, 2.2.2.2
Notes:	None

Description:

The Unlock I/O Request process unlocks the shared memory output data buffer select region of the designated I/O request by resetting the locking semaphore.

3.3.8.8 Process Name:	Unlock I/O Request Read
Inputs:	I/O Request Identifier
Outputs:	I/O Request Semaphore for Input Buffers
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.2.1, 2.2.2.2
Notes:	None

Description:

The Unlock I/O Request Read process unlocks the shared memory input data buffer select region of the designated I/O request by resetting the locking semaphore.

3.3.8.9 Process Name:	Update I/O Request Overrun Count
Inputs:	I/O Request Identifier Number of Overruns
Outputs:	Number of Overruns Variable in Shared Memory
Requirements Reference:	None
Notes:	None

Description:

The Update I/O Request Overrun Count process sets an integer variable in shared memory to equal the designated number of overruns. The process is used to notify the application of the occurrence of an overrun (or frequency of occurrences).

3.3.8.10 Process Name:	Unlock Select Lock
Inputs:	None
Outputs:	Semaphore for Select Buffers
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.2.3
Notes:	None

Description:

The Unlock Select Lock process unlocks the shared memory select buffers by resetting the locking semaphore. The process is called by the CP Select process after writing the transactions to be selected into shared memory and IOP I/O Request Execution process after selecting the specified transactions.

3.3.8.11 Process Name:	Unlock Deselect Lock
Inputs:	None
Outputs:	Semaphore for Deselect Buffers
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.2.3
Notes:	None

Description:

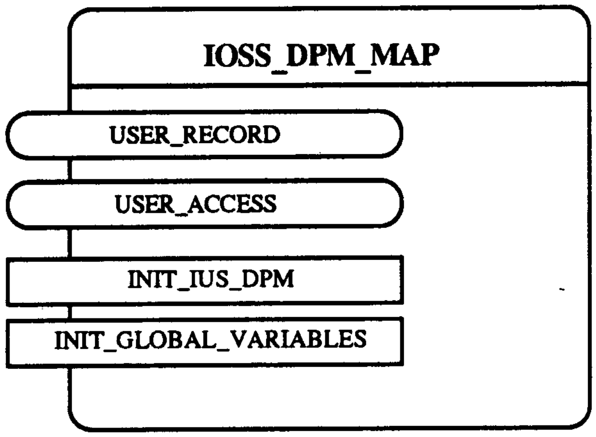
The Unlock Deselect Lock process unlocks the shared memory deselect buffers by resetting the locking semaphore. The process is called by the CP Deselect process after writing the transactions to be deselected into shared memory and IOP I/O Request Execution process after deselecting the specified transactions.

3.3.8.12 Process Name:	Allocate Memory
Inputs:	None
Outputs:	Shared Memory Flags Shared Memory Events Shared Memory Communication Buffers
Requirements Reference:	I/O User Interface Software Specifications, Section 2.2.2.4
Notes:	None

Description:

The Allocate Memory process allocates and initializes flags, events, and buffers in shared memory for the I/O System Services. These flags, events, and buffers are used for interprocessor data communication and synchronization.

3.3.9 I/O System Services Dual Port Memory Map



3.3.9.1 Process Name:	Initialize I/O User Services DPM
Inputs:	User Program and Data Pointer Network Manager Program and Data Pointer
Outputs:	None
Requirements Reference:	None
Notes:	None

Description:

To execute the I/O requests created by the application, the IOSes have to be initialized. The program region of the IOS is partitioned into a header section and a linked list of transaction sections. Each transaction section of the IOS is used by only one chain, whereas the header section is used by all chains.

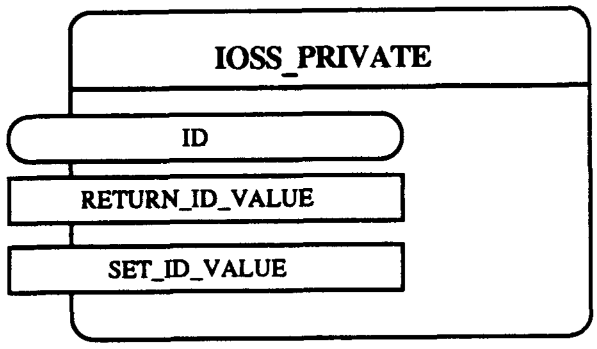
The Initialize I/O User Services DPM process initializes the program region of the IOSes of a GPC with the chain header and linked list of transaction modules. The process must initialize the DPMs prior to the construction of the I/O user chains (see Process 3.3.2.2 - Construct Chain).

3.3.9.2 Process Name:	Initialize Global Variables
Inputs:	None
Outputs:	Global Transaction Count Global Chain Count Global I/O Request Count Memory Index Array
Requirements Reference:	None
Notes:	None

Description:

The Initialize Global Variables process initializes the Transaction, Chain, and I/O request identifier global variables (global to the I/O System Services on a processor). It also initializes the Memory Index Array which is used to initialize the IOS(es) of a GPC.

3.3.10 I/O System Services Private ID Types



3.3.10.1 Process Name:	Return Identifier Value
Inputs:	Identifier of the private type ID
Outputs:	Identifier of the type Integer
Requirements Reference:	None
Notes:	None

Description:

A private type is used for the transaction and chain identifiers to protect the I/O User Interface from errors introduced by the application.

The Return Identifier Value process converts an identifier from the private type to an integer type.

3.3.10.2 Process Name: Set Identifier Value

Inputs:	Identifier of the type Integer
Outputs:	Identifier of the private type ID
Requirements Reference:	None
Notes:	None

Description:

A private type is used for the transaction and chain identifiers to protect the I/O User Interface from errors introduced by the application.

The Set Identifier Value process converts an identifier from an integer type to the private type.

3.4 I/O Communications Management Data Dictionary

Absolute Chain ID : The number of chains that have been created by the applications user (range 1 - Maximum Chain ID).

Absolute I/O Request ID : The number of I/O requests that have been created by the applications user (range 1 - Maximum I/O Request ID).

Absolute Transaction ID : The number of transactions that have been created by the applications user (range 1 - Maximum Transaction ID).

Active Root Link : A record containing the channel number and the channel identifier of the FTP channel which is currently being used to access a given network via an IOS connected to that channel.

Active Root Link Flag: A boolean valued flag indicating whether or not a working connection from a GPC to a root node exists.

Available I/O Services : An array of booleans indexed by I/O service identifiers, one for each GPC in the system. When the boolean is true, the given service is available to the GPC.

Channel Identifier : An identifier which designates a particular physical channel of an IOP.

Channel Number : A logical identifier for a channel of an IOP which contains the IOS connected to a given network.

Chain Record : A record that retains all of the information associated with a chain created by an applications user. The fields of a chain record are as follows:

a) Visible chain ID.
b) Private chain ID.
c) Network ID.
d) Array of transaction IDs (transactions in the chain).
e) Number of transactions.
f) Boolean flag which, when set, indicates that the chain has been assigned to an I/O request.
g) Type of data in the chain (static, dynamic or mixed).
h) Address in the IOS DPM of the first transaction in the chain.
i) Pointer to the next chain in the I/O request.
j) Pointer to the associated I/O request record.

Channel Selection : An array indicating which FTP channels interface to a particular network.

Connected Networks : An array of booleans indexed by GPC identifier. When the boolean is true, the given network is physically connected to the GPC.

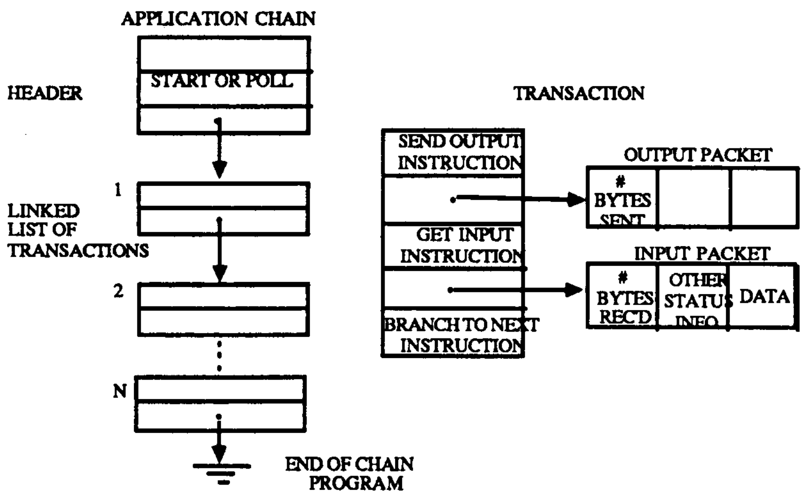
Connection Indicator : A boolean valued object which when true indicates that a given network is physically connected to a given GPC.

Current Root Link : A record containing the channel number and the channel identifier of the FTP channel which is currently being used to access a given network via an IOS connected to that channel.

DIU Chain Header Record : A record that specifies the initial sequence of instructions that are executed by an IOS when each chain is executed. The sequence of instructions are as follows:

a) Read the local time.
b) Read the HDLC Interrupt Register.
c) Read the HDLC Status Register.
d) Set the number of residual bits.
e) Set the polling priority.
f) Enable the receiver.
g) Start the poll.
h) Read the local time.
i) Branch to first transaction in the chain.

DIU Chain Program Record : A record that consists of a DIU chain header record and array of DIU transaction records. The record is used as a template to initialize the user program region of the DPM of the IOS. The organization of the DPM after the initialization process is illustrated in the following figure.



DIU Transaction Record : A record that specifies the sequence of instructions that are executed by an IOS when each transaction is executed. The sequence of instructions are as follows:

a) Disable the HDLC transmitter and receiver.
b) Stop the local timer.
c) Enable the IOS auto flag.
d) Enable the HDLC transmitter.
e) Send the output command (data) to the DIU.
f) Disable the IOS auto flag.
g) Enable the HDLC receiver and disable HDLC transmitter.
h) Read the Interrupt Register.
i) Write the desired time out to the timer and start timer.
j) Receive solicited input.
k) Branch to next transaction in the chain or end of chain program.

DPM Pointer : A pointer whose value is the address of the first addressable byte of one DPM or set of DPMs. When used to read from a DPM, the pointer value selects exactly one DPM. When used to write to a DPM, the pointer value may select a set of physical DPMs, at most one per channel, each occupying the same memory space within the channel. The pointer imposes an organization on the memory space which supports the execution of chains on an I/O network and the reading and writing of data used by those chains.

I/O Network Identifier : A logical identifier which is uniquely assigned to every physical network in the system.

I/O Network Manager Did Not Accept Repair : An array (indexed by network ID) of boolean flags which, when set, indicate that the I/O Network Manager was unable to accept a network repair request.

I/O Request Execution Count : The number of times of I/O request has been executed.

I/O Request Preempted Flag : A boolean flag which, when set, indicates that an I/O request was preempted by a priority 7 I/O request.

I/O Request Record : A record that retains all of the information associated with an I/O request created by an applications user. The fields of an I/O request record are as follows:

a) Visible I/O request ID.
b) Private I/O request ID.
c) Network ID.
d) Array of chain IDs (chains in the I/O request).
e) Number of chains.
f) Total number of transactions in the I/O request.

g) I/O request time out.
h) Frequency at which an event is used to notify the completion of the I/O request.
i) Priority of the I/O request.
j) Scheduling parameters - initiation, completion, and repetition.
k) Pointer into the I/O request priority queue.

I/O Service Descriptor : A record which states whether a given I/O service is local or regional. In the case of a local I/O network, it contains an array of network identifiers which specify the networks assigned to this service.

I/O Service Identifier : A logical identifier which is uniquely assigned to every I/O service in the system.

IOS Identifier : An logical identifier which designates a particular IOS which in turn maps to a specific address range within an FTP channel.

Local Chain Pointers : An array of pointers to chain records created by the applications user. The array is indexed by chain ID and allows direct access to the chain records. Local Chain Pointer arrays are constructed on the CP and IOP.

Local I/O Request Pointers : An array of pointers to I/O request records created by the applications user. The array is indexed by I/O request ID and allows direct access to the I/O request records. Local I/O Request Pointer arrays are constructed on the CP and IOP.

Local Transaction Pointers : An array of pointers to transaction records created by the applications user. The array is indexed by transaction ID and allows direct access to the transaction records. Local Transaction Pointer arrays are constructed on the CP and IOP.

Maximum Storage : An integer constant indicating the maximum number of bytes available for the I/O chain programs and data for the dual ported memory of an I/O Sequencer.

Memory Index Array : An array of pointers indexed by network ID that indicate the next available byte in the DPM of a particular IOS. The array is used to reserve memory in the DPM of the IOS(es) for the transaction I/O buffers.

Posting Task Pointers: An array of pointers indexed by I/O request ID to the tasks that post I/O service requests to the Queue Manager task (I/O Posting tasks). The array is used during the allocation and scheduling of the Posting tasks.

Preempt I/O Request Flag : A boolean flag which, when set, indicates that a priority 7 I/O request is pending.

Preempt Spare Link Test Flag : A boolean flag which, when set, indicates that an I/O request is pending.

Preempted I/O Request ID : The ID of the I/O request that was preempted by a priority 7 I/O request.

Preempted I/O Request State : The state (either "before writing output data to the DPM", "before I/O request execution" or "before uploading input data from the DPM") of the I/O request prior to being preempted by a priority 7 I/O request.

Relative DPM address : A record which is used to map the thirty-two bit address used by the FTP to access a location in an IOS/DPM into a sixteen bit value which the IOS will use to access the same location. Since the address space of the IOS is 8K bytes, only the lower thirteen bits are used in the mapping, the three highest order bits are assigned a value of zero. The mapping is defined below, where f is the value of the ith bit in the sixteen bit address:

$$f(i)= \begin{cases} 0, \text{ if } 15 <= i <= 13 \\ \text{value of the ith bit in the thirty-two bit address if } 0 <= i <= 11 \\ \text{value of the 15th bit in the thirty-two bit address if } i= 12 \end{cases}$$

Restoration Record: A record containing information about the repaired network component which the operator wishes to be returned to service. If a node is to be restored, the node number is provided. If a link is to be restored, a node number and a port number adjacent to that link is provided.

Results of Channel OK Test : A boolean value which is true if the test indicates that an FTP channel is not desynchronized and false if it is desynchronized.

Spare Link Test Preempted Flag : A boolean flag which, when set, indicates that a spare link cycling request was preempted by an I/O request.

System Address : A thirty-two bit value which maps to some physical location in the system. By which the M680X0 microprocessor accesses those physical locations

Unreachable DIUs : A list of DIUs which are attached to failed nodes and which therefore cannot send or receive messages on the I/O network.

Transaction Record : A record that retains all of the information associated with a transaction created by an applications user. The fields of a transaction record are as follows:

a) Visible transaction ID.
b) Private transaction ID.
c) DIU address.

d) Transaction type (input or output).
e) Number of input data bytes.
f) Number of output data bytes.
g) Maximum number of errors before system bypass.
h) Number of errors that have occurred during the execution of the transaction.
i) A boolean flag which, when set, indicates that the transaction is selected.
j) Time out.
k) Local address (CP or IOP) of the input data buffer.
l) Local address (CP or IOP) of the output data buffer.
m) Location of input data buffer in shared memory.
n) Location of output data buffer in shared memory.
o) Pointer to next transaction in chain.
p) Pointer to associated chain record.
q) Location of input data buffer in DPM of the IOS
r) Location of output data buffer in DPM of IOS.
s) Boolean flag indicating if the transaction has been assigned to a chain.

User Pointer : A pointer whose value is the address of the first addressable byte of the upper 4K of one DPM or set of DPMs. When used to read from a DPM, the pointer value selects exactly one DPM. When used to write to a DPM, the pointer value may select a set of physical DPMs, at most one per channel, each occupying the same memory space within the channel. The pointer imposes an organization on the memory space which supports the execution of application chains on an I/O network and the reading and writing of data used by those chains.

4.0 I/O SYSTEM SERVICES USER EXAMPLE

The I/O User Interface provides a user friendly environment allowing the applications programmer to create and schedule I/O requests, develop application tasks and processes, and access DIUs. This section provides examples of the construction of an I/O network, creation of an I/O request and development of an application task to illustrate the use of the I/O System Services.

4.1 Overview

The I/O System Services are written in Ada®, a design language primarily developed for the US Department of Defense for use in embedded systems. Ada is structured around a package framework allowing the development of structured modules. Accordingly, the I/O networks, I/O requests and application tasks are constructed in a package oriented environment.

The construction of I/O networks, I/O requests, and application tasks require the modification/creation of five Ada packages: I/O System Services Network Data Types, I/O System Services Central Database, I/O requests, applications tasks, and I/O System Services CP Main Initialization (illustrated in the Booch diagrams of Figures 12- 16). The I/O System Services Central Database and Network Data Types packages are the packages in which the networks are defined, and they must be modified to construct the network topologies required by the application. The I/O Requests package is a module in which the transactions, chains, and I/O requests are created. Furthermore, the Application Tasks package is the module in which the user develops the application tasks that interact with the I/O requests. Finally, the CP Main Initialization package is the means by which the user transfers control from the I/O User Interface to the application tasks.

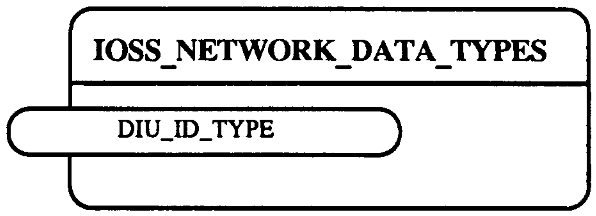


Figure 12: I/O System Services Network Data Types Package

PRECEDING PAGE BLANK NOT FILMED

PAGE 128 INTENTIONALLY BLANK

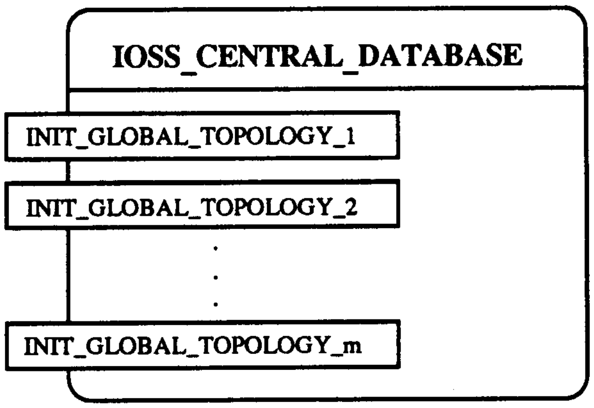


Figure 13: I/O System Services Central Database Package

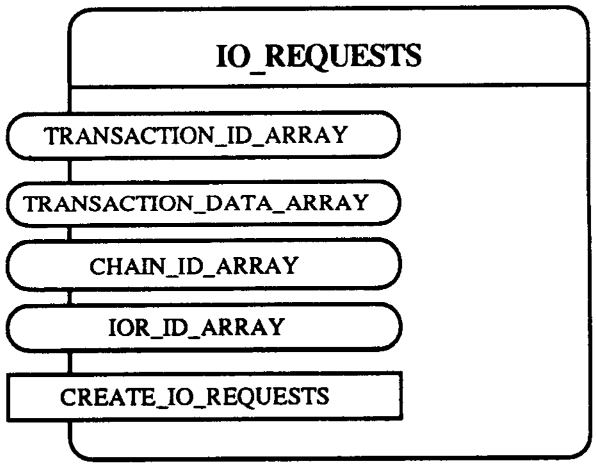


Figure 14: I/O Request Package

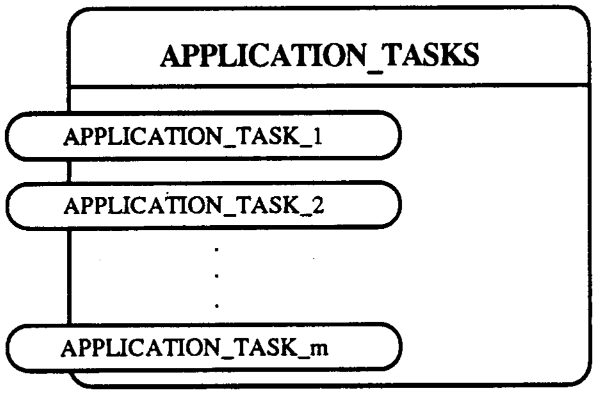


Figure 15: Application Tasks Package

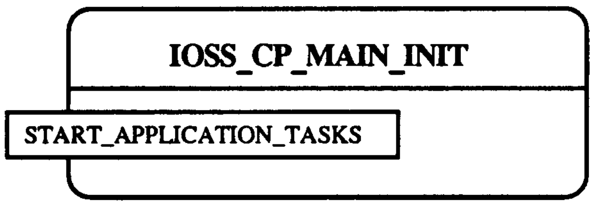


Figure 16: CP Main Initialization Package

Section 4.2 presents a template to illustrate the construction of an I/O network, while Section 4.3 uses an example to illustrate the creation of a single chain, on demand I/O request and a two chain, periodic I/O request. Section 4.4 provides an example of a on demand application task which is synchronized with the I/O requests created in Section 4.3. Finally, Section 4.5 provides an example of the use of the CP Main Initialization package to transfer control from the I/O User Interface to the application tasks.

4.2 Construction of an I/O Network Topology

The discussion of the I/O User Services and I/O Communications Management function of the I/O System Services has continually referred to I/O networks. The I/O networks are defined in the INIT_GLOBAL_TOPOLOGY procedures of the package IOSS_CENTRAL_DATABASE. These procedures must be modified to create the I/O network topologies required by the application. In addition, the type DIU_ID_TYPE in the IOSS_NETYPES package must be modified to provide logical names for the DIUs that are connected to the network. In this section, the method to construct a specific network topology for the AIPS system is illustrated.

```
procedure body INIT_GLOBAL_TOPOLOGY_2 is
begin

N_P_N(2) := 10;
-- Network 2 consists of 10 nodes.

GLOBAL_TOPOLOGY(2) := new NODE_ARRAY_TYPE(1.. N_P_N(2));
-- Define the range of nodes numbers in network 2 and allocate the required memory.

-- Node 1
GLOBAL_TOPOLOGY(2)(1).NODE_ADDRESS := 1;
-- The physical node address for logical node 1 of network 2 is 1;
-- Port 0 of Node 1 is a Root Node, connected to an IOS (logical ID of 3) in
-- channel A of a GPC, and the network is connected to GPC 2 (GPC_ADDR).
GLOBAL_TOPOLOGY(2)(1).PORT_ARRAY(0) :=
                    (ADJACENT_ELEMENT  => GPC,
                    GPC_ADDR           => 2,
                    CHANNEL            => A,
                    IOS                => 3);

-- Port 1 of Node 1 is connected to Port 1 of logical Node 3 (Node_Number is
-- logical ID) The physical address (Node_Address) of Node 3 is also 3.
GLOBAL_TOPOLOGY(2)(1).PORT_ARRAY(1) :=
                    (ADJACENT_ELEMENT  => NODE,
                    NODE_NUMBER        => 3,
                    NODE_ADDRESS       => 3,
                    PORT_NUMBER        => 1);

-- Port 2 of Node 1 is connected to Port 2 of Node 2.
GLOBAL_TOPOLOGY(2)(1).PORT_ARRAY(2) :=
                    (ADJACENT_ELEMENT  => NODE,
                    NODE_NUMBER        => 2,
                    NODE_ADDRESS       => 2,
                    PORT_NUMBER        => 2);

-- Port 3 of Node 1 is connected to a DIU with a logical name of S1 and physical
-- address of 11. The logical name S1 must be included in the type DIU_ID_TYPE
-- in the IOSS_NETYPES package.
GLOBAL_TOPOLOGY(2)(1).PORT_ARRAY(3) :=
                    (ADJACENT_ELEMENT  => DIU,
                    DIU_ADDR           => 11,
                    DIU_ID             => S1);

-- Port 4 of Node 1 is connected to Port 4 of Node 6.
GLOBAL_TOPOLOGY(2)(1).PORT_ARRAY(4) :=
                    (ADJACENT_ELEMENT  => NODE,
                    NODE_NUMBER        => 6,
                    NODE_ADDRESS       => 6,
                    PORT_NUMBER        => 4);

-- Node 2
GLOBAL_TOPOLOGY(2)(2).NODE_ADDRESS := 2;
-- The physical node address for logical node 2 of network 2 is 2;
-- Port 0 of Node 2 is connected to Port 0 of Node 8.
GLOBAL_TOPOLOGY(2)(2).PORT_ARRAY(0) :=
                (ADJACENT_ELEMENT   => NODE,
                 NODE_NUMBER        => 8,
                 NODE_ADDRESS       => 8,
                 PORT_NUMBER        => 0);

-- Port 1 of Node 2 is not connected.
GLOBAL_TOPOLOGY(2)(2).PORT_ARRAY(1) :=
                (ADJACENT_ELEMENT   => NONE);

-- Port 2 of Node 2 is connected to Port 2 of Node 1.  Notice the corresponding
-- connections with Port 2 of Node 1 in the database.
GLOBAL_TOPOLOGY(2)(2).PORT_ARRAY(2) :=
                (ADJACENT_ELEMENT   => NODE,
                 NODE_NUMBER        => 1,
                 NODE_ADDRESS       => 1,
                 PORT_NUMBER        => 2);

-- Port 3 of Node 2 is not connected.
GLOBAL_TOPOLOGY(2)(2).PORT_ARRAY(3) :=
                (ADJACENT_ELEMENT   => NONE);

-- Port 4 of Node 2 is connected to Port 4 of Node 4.
GLOBAL_TOPOLOGY(2)(2).PORT_ARRAY(4) :=
                (ADJACENT_ELEMENT   => NODE,
                 NODE_NUMBER        => 4,
                 NODE_ADDRESS       => 4,
                 PORT_NUMBER        => 4);

                 .
                 .
                 .
                 .
```

```
-- Node 10
GLOBAL_TOPOLOGY(2)(10).NODE_ADDRESS := 10;
-- The physical node address for logical node 10 of network 2 is 10;

        -- Port 0 of Node 10 is connected to Port 0 of Node 4.
        GLOBAL_TOPOLOGY(2)(10).PORT_ARRAY(0) :=
                        (ADJACENT_ELEMENT   => NODE,
                        NODE_NUMBER         => 4,
                        NODE_ADDRESS        => 4,
                        PORT_NUMBER         => 0);

        -- Port 1 of Node 10 is not connected.
        GLOBAL_TOPOLOGY(2)(10).PORT_ARRAY(1) :=
                        (ADJACENT_ELEMENT   => NONE);
        -- Port 2 of Node 10 is connected to Port 2 of Node 9.
        GLOBAL_TOPOLOGY(2)(10).PORT_ARRAY(2) :=
                        (ADJACENT_ELEMENT   => NODE,
                        NODE_NUMBER         => 9,
                        NODE_ADDRESS        => 9,
                        PORT_NUMBER         => 2);

        -- Port 3 of Node 10 is connected to Port 3 of Node 8.
        GLOBAL_TOPOLOGY(2)(10).PORT_ARRAY(3) :=
                        (ADJACENT_ELEMENT   => NODE,
                        NODE_NUMBER         => 8,
                        NODE_ADDRESS        => 8,
                        PORT_NUMBER         => 3);

        -- Port 4 of Node 10 is not connected.
        GLOBAL_TOPOLOGY(2)(10).PORT_ARRAY(4) :=
                        (ADJACENT_ELEMENT   => NONE);

end INIT_GLOBAL_TOPOLOGY_2;
```

4.3 Creation of an I/O Request

The creation of I/O requests was discussed in detail in Section 2.2 - I/O User Interface Software Specifications. In this section, the creation of a periodic and on demand I/O request is provided to illustrate the use of the I/O User Interface.

```
with IOSS_IOR_SPEC; use IOSS_IOR_SPEC;
-- include the I/O Request Specification package to make the I/O User Interface calls
-- visible to the IO_REQUESTS package;

package IO_REQUESTS is
-- An Ada package consists of a package specification and a package body. The package
-- specification contains the types, objects, and subprograms declarations that are visible to
-- other packages. The body contains transparent types, objects, and subprograms and
-- provides the instructions that implement the visible subprograms.

     type BYTE is range 0 .. 255;  -- define a memory byte type.
     for BYTE'SIZE use 8;          -- set the size of the type BYTE to be 8 bits.
     TRANSACTION_INPUT_DATA_1 : array (0 .. 6) of BYTE;
     TRANSACTION_INPUT_DATA_2 : array (0 .. 6) of BYTE;
     TRANSACTION_OUTPUT_DATA_1 : array (0 .. 1) of BYTE;
     TRANSACTION_OUTPUT_DATA_2 : array (0 .. 1) of BYTE;
     TRANSACTION_OUTPUT_DATA_3 : array (0 .. 1) of BYTE;
     TRANSACTION_OUTPUT_DATA_4 : array (0 .. 1) of BYTE;
       -- Each output transaction (3,4) has 2 bytes of output data. Each input
       -- transaction (1,2) has 2 bytes of output data and 7 bytes of input data.

     TRANSACTION_ID_ARRAY_1 : ID_ARRAY(1);
     TRANSACTION_ID_ARRAY_2 : ID_ARRAY(1);
     TRANSACTION_ID_ARRAY_3_4 : ID_ARRAY(2);
       -- Definition of the transaction arrays that will be passed into the Create_Chain
       -- procedure. The number in parentheses depicts the number of transactions that
       -- will be in the chain.

     CHAIN_ID_ARRAY_1_2 : ID_ARRAY(2);
     CHAIN_ID_ARRAY_3 : ID_ARRAY(1);
       -- Chains 1 and 2 will be executed simultaneously on two redundant networks
       -- (the chains of multiple chain I/O requests must be executed on different
       -- networks). Chain 3 is only executed on one network. The number
       -- in parentheses depicts the number of chains that will be in the I/O request.

     IOR_1, IOR_2 : IOR_ID_TYPE;
       --The definition of the I/O request IDs.

     procedure CREATE_IO_REQUESTS;
       -- The procedure that creates the I/O requests. It will be called from the CP Main
       -- Initialization package.

end IO_REQUESTS;
```

```
package body IO_REQUESTS is

SCHED_INFO : IOR_SCHED_RECORD;
PERIODIC_SCHED_INFO : PERIODIC_SCHED_RECORD;
TRANS_INFO : TRANSACTION_INFO_RECORD;
-- Definition of types that are used to create the transactions and I/O requests.

procedure CREATE_IOR_1 is

begin

        TRANSACTION_OUTPUT_DATA_1 := (16#F3#, 16#B0#);
        TRANSACTION_OUTPUT_DATA_2 := (16#F3#, 16#B0#);
        -- Initialize the output data buffers for transactions 1 and 2.

        TRANS_INFO := ( IO                  =>   INPUT;
                        DIU_ID              =>   12;
                        NUM_DATA_BYTES_IN   =>   7;
                        NUM_DATA_BYTES_OUT  =>   2;
                        DYNAMIC_OR_STATIC   =>   STATIC;
                        MAX_BEFORE_BYPASS   =>   0;
                        TIME_OUT            =>   255;
                        DATA_BUFFER_INPUT   =>
                              TRANSACTION_INPUT_DATA_1'ADDRESS;
                        DATA_BUFFER_OUTPUT  =>
                              TRANSACTION_OUTPUT_DATA_1'ADDRESS);
        -- Definition of I/O parameters for transaction #1

        CREATE_TRANSACTION( TRANSACTION_ID_ARRAY_1.IDS(1),
                            TRANS_INFO);
        -- Create transaction #1;
```

```
TRANS_INFO := ( IO                      =>    INPUT;
                DIU_ID                  =>    13;
                NUM_DATA_BYTES_IN       =>    7;
                NUM_DATA_BYTES_OUT      =>    2;
                DYNAMIC_OR_STATIC       =>    STATIC;
                MAX_BEFORE_BYPASS       =>    50;
                TIME_OUT                =>    128;
                DATA_BUFFER_INPUT       =>
                       TRANSACTION_INPUT_DATA_2'ADDRESS;
                DATA_BUFFER_OUTPUT      =>
                       TRANSACTION_OUTPUT_DATA_2'ADDRESS);
-- Definition of I/O parameters for transaction #2

CREATE_TRANSACTION( TRANSACTION_ID_ARRAY_2.IDS(1),
                    TRANS_INFO);
-- Create transaction #2;

CREATE_CHAIN(   CHAIN_ID_ARRAY_1_2.IDS(1),
                NETWORK_ID => 1,
                TRANSACTION_ID_ARRAY_1);
-- Create Chain #1 which will execute transaction #1 on network #1.

CREATE_CHAIN(   CHAIN_ID_ARRAY_1_2.IDS(2),
                NETWORK_ID => 3,
                TRANSACTION_ID_ARRAY_2);
-- Create Chain #2 which will execute transaction #2 on network #3.

PERIODIC_SCHED_INFO := ( HOW_STARTED =>START_AFTER_DELAY;
                         WAIT_FOR    => DURATION(30));
-- Start the periodic application task after a 30 second delay.

SCHED_INFO := ( PRIORITY            =>  6;
                COMPLETION_EVENT    =>  ALWAYS;
                IOR_TIME_OUT        =>  1000;
                HOW_SCHEDULED       =>  PERIODIC;
                START               =>  PERIODIC_SCHED_INFO;
                REPETITION_PERIOD   =>  DURATION(.01);
                WHEN_TO_STOP        =>  NEVER_STOP);
-- Schedule the I/O request to be periodic at 100 hz., have a priority of 6, always
-- use an event to signal its completion, start after a 30 second delay, require 1 ms.
-- on the networks and never stop.
```

```
        CREATE_IOR(IOR_1, CHAIN_ID_ARRAY_1_2, SCHED_INFO);
        -- Create I/O request #1

end CREATE_IOR_1;

procedure CREATE_IOR_2 is

begin

        TRANSACTION_OUTPUT_DATA_3 := (16#F3#, 16#B1#);
        TRANSACTION_OUTPUT_DATA_4 := (16#F3#, 16#B2#);
        -- Initialize the output data buffers for transactions 3 and 4.

        TRANS_INFO := ( IO                      =>    OUTPUT;
                        DIU_ID                  =>    14;
                        NUM_DATA_BYTES_OUT      =>    2;
                        DYNAMIC_OR_STATIC       =>    DYNAMIC;
                        DATA_BUFFER_OUTPUT      =>
                              TRANSACTION_OUTPUT_DATA_3'ADDRESS);
        -- Definition of output parameters for transaction #3

        CREATE_TRANSACTION( TRANSACTION_ID_ARRAY_3_4.IDS(1),
                            TRANS_INFO);
        -- Create transaction #3;


        TRANS_INFO := ( IO                      =>    OUTPUT;
                        DIU_ID                  =>    13;
                        NUM_DATA_BYTES_OUT      =>    2;
                        DYNAMIC_OR_STATIC       =>    DYNAMIC;
                        DATA_BUFFER_OUTPUT      =>
                              TRANSACTION_OUTPUT_DATA_4'ADDRESS);
        -- Definition of output parameters for transaction #4

        CREATE_TRANSACTION( TRANSACTION_ID_ARRAY_3_4.IDS(2),
                            TRANS_INFO);
        -- Create transaction #4;

        CREATE_CHAIN(   CHAIN_ID_ARRAY_3.IDS(1),
                        NETWORK_ID => 2,
                        TRANSACTION_ID_ARRAY_3_4);
        -- Create Chain #3 which will execute transactions #3 and #4 on network #2.
        SCHED_INFO := ( PRIORITY                => 7;
                        COMPLETION_EVENT        => NEVER;
                        IOR_TIME_OUT            => 2000;
                        HOW_SCHEDULED           => ON_DEMAND);
        -- Schedule the I/O request to be on demand, have a priority of 7, require 2 ms. on
        -- the network and never use an event to signal its completion.

        CREATE_IOR(IOR_2, CHAIN_ID_ARRAY_3, SCHED_INFO);
        -- Create I/O request #2

end CREATE_IOR_2;

procedure CREATE_IO_REQUESTS is

begin
        CREATE_IOR_1;
        CREATE_IOR_2;

end CREATE_IO_REQUESTS;

end IO_REQUESTS;
```

4.4 Creation of an Application Task

The application tasks that are created by the user may be synchronized with specific I/O requests or "free running" processes. In the simple synchronized case, either the I/O request is periodically triggered by the application (on demand I/O request; periodic application) or the application task is periodically triggered by the I/O request (on demand application; periodic I/O request). In the simple free running case, the I/O requests and application tasks are both periodic. Alternatively, combinations of the synchronization/free running schemes can be incorporated to handle more complex control applications.

This section presents an example to illustrate some of the features of the I/O User Interface. The example constructs an application task which is synchronized with the I/O requests created in Section 4.3. The control flow of the application task is as follows:

1) I/O request #1 periodically reads sensor information.
2) The application task is started by the completion event of I/O request #1.
3) After the application task is started, the input data is read. The application task uses the input data to generate output commands.
4) After the output commands have been calculated, the application task writes the output data and uses an event to start I/O request #2.
5) After I/O request #2 is started, the application task loops back to repeat the process (2 - 4).

```
with    SCHEDULER, EVENT_CONTROL, IO_REQUESTS, IOSS_IOR_SPEC,
        TASKS_IDS, APPLICATION_LOG;
use     SCHEDULER, EVENT_CONTROL, IO_REQUESTS, IOSS_IOR_SPEC,
        TASKS_IDS, APPLICATION_LOG;

package APPLICATION_TASKS is

        task type APPLICATION_TASK_TYPE is
                entry START_APPLICATION_TASK;
        end APPLICATION_TASK_TYPE;

APPLICATION_TASK : APPLICATION_TASK_TYPE;
-- Define an object of APPLICATION_TASK_TYPE.

end APPLICATION_TASKS;

package body APPLICATION_TASKS is

function GET_APPLICATION_TASK_ID is new TASK_IDS.ID_OF
                                        (APPLICATION_TASK_TYPE);
APPLICATION_TASK_ID : TASK_IDS.TASK_ID :=
                        GET_APPLICATION_TASK_ID(APPLICATION_TASK);
-- Determine an ID for the application task to schedule it.

task body APPLICATION_TASK_TYPE is

COMPLETION_EVENT : a_EVENT;
LOCKED, ERROR, OLD_DATA : BOOLEAN := FALSE;
OVERRUN_CNT : INTEGER := 0;
        -- Define some objects that will be used for error checking.
APPLICATION_LOG_ID : LOG_RANGE := 1;
        -- Select the application Log to be used.

begin

        accept START_APPLICATION_TASK;
        -- Wait until control has been passed to the application task from the CP Main
        -- Initialization procedure.

        WAIT_UNTIL_IOP_COMPLETED;
        --Wait until the I/O Services have been initialized.
```

```
COMPLETION_EVENT := IOR_COMPLETION_EVENT(IOR_1);
-- Obtain the pointer to the completion event associated with I/O request #1.

SCHEDULE( APPLICATION_TASK_ID,
          FALSE,
          SAME_PRIORITY,
          (ON_EVENT_SET, COMPLETION_EVENT),
          NO_REPETITION,
          NO_COMPLETION);
--Schedule the application task to be triggered by the completion event from I/O
-- request #1.

loop

     WAIT_FOR_SCHEDULE;
     -- Wait for the scheduling requirements to be met. Specifically, wait for I/O
     -- request #1 to be completed.

     READ_IOR( IOR_1, LOCKED, ERROR, OLD_DATA);
     -- Read the input data returned by I/O request #1. The input data is located
     -- in the buffers defined in IO_REQUESTS for transactions 1 and 2.

     if not ERROR and then not LOCKED then

          --
          -- Follow appropriate control laws to determine output data.
          -- Store the output data in the output buffers allocated for
          -- transactions 3 and 4 in the package IO_REQUESTS.
          --

          WRITE_IOR(IOR_2, LOCKED);
          -- Write the output data.

          START_IOR(IOR_2);
          -- Start I/O request #2.

     else

          LOG_ERROR(APPLICATION_LOG_ID,
                    INTEGER(IOR_1);
                    "APPLICATION TASK",
                    "LOCKED OR ERROR DURING READ");
          -- Log error in application log.
```

```
                end if;

            end loop;

        end APPLICATION_TASK_TYPE;

        end APPLICATION_TASKS;
```

4.5 Passing Control to an Application Task

The Local System Services performs the function of GPC Initialization, as discussed in [1]. After the GPC Initialization process is complete, the I/O requests can be created and the application tasks started. The START_APPLICATION_TASKS procedure allows the user to invoke the CREATE_IO_REQUESTS procedure and subsequently start the predefined application tasks. Unlike the IO_REQUESTS and APPLICATION_TASKS packages which can be arbitrarily named by the user, the IOSS_CP_MAIN_INIT package and START_APPLICATION_TASKS procedure names *can not* be changed. The main program on the CP expects these names to exist.

The following example illustrates the invocation of the user defined Create I/O Requests procedure and starting the associated application tasks.

```
with    IO_REQUESTS, IOSS_IOR_SPEC, APPLICATION_TASKS;
use     IO_REQUESTS, IOSS_IOR_SPEC, APPLICATION_TASKS;
package IOSS_CP_MAIN_INIT is

        procedure START_APPLICATION_TASKS;
        -- Make the Start Application Task procedure visible to the CP Main Program

end IOSS_CP_MAIN_INIT;

package body IOSS_CP_MAIN_INIT is

procedure body START_APPLICATION_TASKS is
begin

        CREATE_IO_REQUESTS;
        -- Invoke Create I/O Requests Procedure.

        CP_COMPLETED;
        -- Signal the IOP that the I/O requests have been created.
```

```
          APPLICATION_TASK.START_APPLICATION_TASK;
          -- Start the application tasks.

      end START_APPLICATION_TASKS;

      end IOSS_CP_MAIN_INIT;
```

5.0 CONCLUSIONS AND RECOMMENDATIONS

The Advanced Information Processing System (AIPS) I/O System Services have been designed, implemented, and tested on the centralized configuration of the AIPS engineering model. The I/O User Interface provides a robust, user friendly environment that allows an applications programmer to create I/O requests and applications tasks to directly reference DIUs (to the user, the DIUs appear to be memory mapped). The I/O Communications Management system interacts with the I/O User Interface to schedule, execute, and process the I/O requests. The I/O Network Management function initializes its associated network and performs network FDIR when required. The responsibilities of this software include the following: communicating I/O request specifications from the CP to the IOP, scheduling application tasks and I/O requests, communicating I/O data between the CP and IOP, executing I/O requests on single and redundant networks, performing I/O request error processing, communicating error and status information to application tasks, and performing network FDIR.

5.1 Preliminary Testing of the I/O System Services

The preliminary testing of the I/O System Services was performed by creating a sample set of I/O requests and application tasks. The I/O requests exercised all of the I/O communication, scheduling and synchronization features of the I/O User Interface. The application tasks tested the interprocessor communication of I/O data and status information. This testing process involved:

1) single and redundant networks of varying topologies
2) network nodes simulating DIUs
3) synchronous and asynchronous I/O requests of varying priorities and sizes
4) synchronous and asynchronous application tasks of varying complexities
5) interprocessor contention for shared resources
6) channel, data exchange, IOS, network and DIU fault injection
7) channel and network node/link restoration
8) spare link cycling

The testing of I/O System Services focused on the near simultaneous execution of I/O chains on redundant I/O networks (a redundant network is a set of I/O networks connected to the same GPC, where each network consists of a set of redundant DIUs). The tests primarily examined the performance of the I/O System Services when faults were injected in a redundant network consisting of two I/O networks and six DIUs. It was crucial to verify that the I/O requests were executed without interuption on the fault-free network while I/O network FDIR was performed on the faultly network. The types of network faults that were injected were IOS, link, and node faults. In all test cases, the faults were identifed (and bypassed if possible), and the I/O requests were executed on schedule.

5.2 Performance Metrics

Performance metrics were recorded and evaluated for a sample I/O application. This application involved:

1) a redundant I/O service - two 10 node networks
2) network nodes simulating DIUs
3) three application tasks
    a) a periodic task scheduled to execute every 100 ms. (10 Hz.)
    b) an on_demand task scheduled to execute every 200 ms. (5 Hz.)
    c) an on_demand task scheduled to execute every 400 ms. (2.5 Hz.)
    d) the periodic task started the on_demand tasks using local events
4) three on_demand I/O requests
5) synchronized application tasks and I/O requests
    a) the 10 Hz. task starts a two chain I/O request which has 8 input transactions per chain
    b) the 5 Hz. task starts a two chain I/O request which has 10 input transactions per chain
    c) the 2.5 Hz. task starts a two chain I/O request which has 2 input transactions per chain

The performance metrics were measured on the centralized configuration of the AIPS engineering model which incorporates the following components:

1) 68010 processors with 7.9 MHz. clocks (CP and IOP)
2) custom I/O Sequencers with 7.9 MHz. clocks
3) 2 MBit/second I/O buses
4) custom network nodes
    a) 68701 processor with a 2 MHz. clock
    b) requires a 256 microsecond "quiet time" between successive node transmissions to check input buffers (affects network FDIR)
5) Verdix 5.4 Run Time System

The I/O request processing times for the sample application are illustrated in Figures 17 and 18. The timings of the I/O System Services procedure and function calls are presented in Figure 19.

FUNCTION	REQUIRED TIME
I/O Request (2 Chains - 2 Transaction per Chain) (2.5 Hz. I/O request)	25 ms.
a) Overhead of event processing	1.0 ms.
b) Overhead of accepting I/O request for processing	2.0 ms.
c) Overhead of transaction select/deselect check	0.1 ms.
d) Determine SM Buffer	0.3 ms.
e) Write output data and setup chain	9.6 ms.
f) Execute Chain	3.2 ms.
g) Calculate SM Buffer	0.4 ms.
h) Read input data and error processing	8.2 ms.
i) Required IOS header time	150 µs.
j) Required IOS processing time for input transaction	235 µs.
k) Required network time	334 µs.
l) Approximate IOR Time Out	1 ms.
I/O Request (2 Chains - 8 Transactions per Chain) (10 Hz. I/O request)	41.1 ms.
a) Overhead of event processing	1.0 ms.
b) Overhead of accepting I/O request for processing	2.0 ms.
c) Overhead of transaction select/deselect check	0.1 ms.
d) Determine SM Buffer	0.3 ms.
e) Write output data and setup chain	12.9 ms.
f) Execute Chain	6.0 ms.
g) Calculate SM Buffer	0.4 ms.
h) Read input data and error processing	18.2 ms.
i) Required IOS header time	150 µs.
j) Required IOS processing time for input transaction	235 µs.
k) Required network time	1632 µs.
l) Approximate IOR Time Out	3.8 ms.

Figure 17: I/O Request Processing Times for the Sample Application

FUNCTION	REQUIRED TIME
I/O Request (2 Chains - 10 Transactions per Chain) (5 Hz. I/O request)	52.6 ms.
a) Overhead of event processing	1.0 ms.
b) Overhead of accepting I/O request for processing	2.0 ms.
c) Overhead of transaction select/deselect check	0.1 ms.
d) Determine SM Buffer	0.3 ms.
e) Write output data and setup chain	15.9 ms.
f) Execute Chain	6.8 ms.
g) Calculate SM Buffer	0.4 ms.
h) Read input data and error processing	25.8 ms.
i) Required IOS header time	150 µs.
j) Required IOS processing time for input transaction	235 µs.
k) Required network time	1916 µs.
l) Approximate IOR Time Out	4.6 ms.
Latency between execution of 2 simultaneous chains	4.1 µs.

Figure 18: I/O Request Processing Times for the Sample Application (Continued)

5.3 Future Work

As mentioned previously, the design and implementation of the I/O System Services was completed on the centralized configuration of the AIPS engineering model. With respect to a centralized system, the I/O System Services is finished. Yet, with respect to the distributed configuration of the engineering model, the I/O System Services must be modified. These modifications are required because the I/O Communications Manager and its associated I/O Network Managers could reside on different remote sites. Accordingly, the interface between the I/O Communications Manager and Network Manager must be extended to allow for both intracomputer and intercomputer communication of FDIR, spare link cycling, and restoration commands and network status information.

FUNCTION	REQUIRED TIME
Write_IOR	
a) 2 Chains, 2 Transactions per Chain, 2 Bytes	2.4 ms.
b) 2 Chains, 8 Transactions per Chain, 2 Bytes	6.7 ms.
c) 2 Chains, 10 Transactions per Chain, 2 Bytes	8.2 ms.
Read_IOR	
a) 2 Chains, 2 Transactions per Chain, 7 Bytes	3.2 ms.
b) 2 Chains, 8 Transactions per Chain, 7 Bytes	8.4 ms.
c) 2 Chains, 10 Transactions per Chain, 7 Bytes	10.1 ms.
Start_IOR	0.5 ms.
Stop_IOR	0.5 ms.
Transaction_Error	0.2 ms.
Transaction_Is_Bypassed	0.2 ms.
Chain_Error	0.6 ms.
IOR_Has_Overrun	0.1 ms.
Select_Transaction	0.5 ms.
Deselect_Transaction	0.5 ms.
Write_Transaction	1.2 ms.
IOR_Ready	0.1 ms.
Clear_IOR_Ready	0.1 ms.
IOR_Ready_and_Clear	0.2 ms.

Figure 19: I/O System Services Timings for the Sample Application

6.0 REFERENCES

1. L. Burkhardt, L. Alger, R. Whittredge, and P. Stasiowski, "Advanced Information Processing System: Local System Services", NASA Contractor Report 181767, March, 1989.

2. G. Nagle, L. Alger, and A. Kemp, "Advanced Information Processing System: Input/Output Network Management Software", NASA Contractor Report 181678, May, 1988.

3. G. Booch, Software Engineering with Ada®, Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA, 1983.

APPENDIX A: GLOSSARY OF I/O NETWORK TERMS

Network Hardware

Node: a circuit switching device with 5 ports which can each be independently enabled or disabled. An enabled port retransmits the logical OR of all data which has been received by any other enabled port. The retransmission is carried out with minimal delay, nominally one half the period of the transmission clock.

Device Interface Unit (DIU): The smallest unit addressable by an application on an I/O network. DIUs may be single devices (such as sensors or actuators) or collections of such devices.

IOP: I/O Processor.

CP: Computational Processor.

GPC: General Purpose Computer consisting of an IOP, a CP, and their interfaces to I/O and IC networks.

I/O Sequencer (IOS): A state machine whose function is to conduct the physical aspects of communication on an I/O network for a GPC. The IOS communicates with one channel of a GPC by means of a Dual Ported Memory (DPM). The IOS executes a program which has been stored in DPM by the IOP. Part of the DPM behaves as a set of control and status registers for the IOS. Once a program has been stored in the DPM, communication between the IOS and GPC can be conducted by means of the control and status registers. Once the programs have been stored in the DPM, it is only necessary to update the data required by these programs. Input data must be exchanged across redundant channels for source congruency and output data must be voted to provide fault masking. Each IOS is a simplex device which performs its function asynchronously from other IOSes and from the GPC to which it is connected.

Network Interface: the hardware involved in the connection between a GPC and a network. It consists of an IOS, 8 K bytes of dual ported memory, and a link (called the root link) connecting the IOS to a network node (called the root node).

I/O Network: a set of nodes and DIUs which are physically interconnected.

I/O Network Topology: The specific interconnections among the nodes, GPCs and DIUs in an I/O network.

Virtual Bus: A network whose nodes have been configured to allow communication between a GPC and DIUs or nodes on that network to emulate communication on a serial bus.

Network Classification

I/O Service: A logical organization imposed on I/O network use. A service may be designated as Regional or Local.

Regional I/O Service: I/O activity conducted on a single I/O network which is shared among several GPCs. Since only one GPC may use the network at any given time, GPCs must contend for use of the network.

Local I/O Service: I/O activity conducted on an I/O network which is used exclusively by one GPC. If an I/O network which is part of a Local I/O Service is physically connected to more than one GPC, only one of those GPCs will be included in the service at any given time. A change in the GPC included in the service constitutes a function migration.

Redundant I/O Network: a set of I/O networks connected to the same GPC. Each network in the set consists of a set of corresponding, redundant devices (sensors and effectors). It is not required that these devices be interconnected by the same topology. To support function migration, each network in the set may have corresponding connections to more than one GPC. However, during normal operation, access to this set of networks is reserved exclusively for one GPC.

Redundant I/O Service: A special form of Local I/O Service where I/O activity is conducted on a set of redundant I/O networks. This is the only type of service supported on redundant I/O networks. The intent of this service is threefold:

1) to provide simultaneous access to redundant devices on redundant networks during no fault conditions
2) to increase the bandwidth of the physical I/O networks communicating with redundant external devices.
3) to provide applications an uninterrupted flow of I/O data during periods of network reconfiguration activity.

Network Protocols

HDLC Protocol: The bit oriented protocol conducted on the data link of the communication hierarchy.

General I/O Protocol: The protocol followed between the IOS and nodes/DIUs for the purpose of conducting I/O transactions. All transactions begin with a command frame sent from the IOS to a node or DIU. A node always returns a response frame. A DIU is not required to return a response frame.

Network Traffic

Frame: An HDLC frame. The smallest unit of communication between an IOS and an external device (node or DIU) on an I/O network.

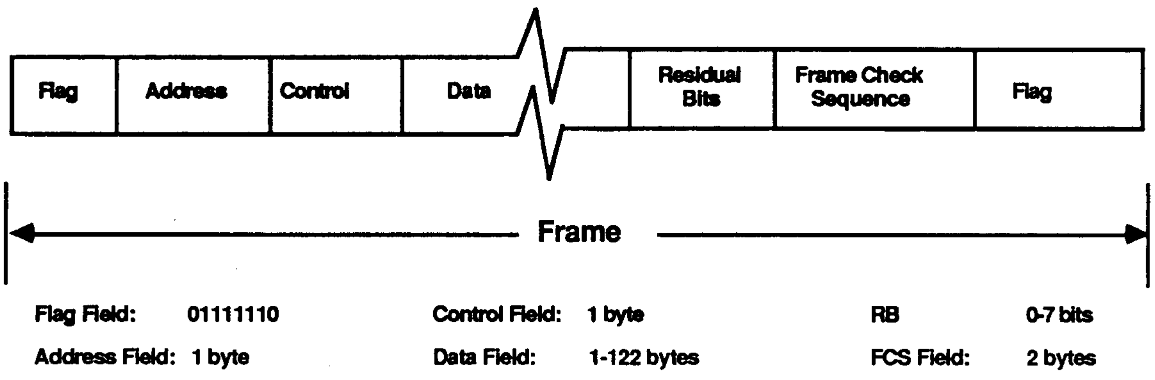


Figure 17: HDLC Frame Format

Command Frame: A frame sent to a single node or DIU from an IOS using the HDLC protocol. See Figure 17. A command frame to a node is distinguished from a command frame to a DIU by the number of residual bits which are transmitted. A node command frame has three residual bits while a DIU command frame has five residual bits.

Response Frame: A frame sent to a GPC from a node or DIU using the HDLC protocol. See Figure 17. A response frame from a node is distinguished from a response frame from a DIU by the number of residual bits which are transmitted. A node response frame has three residual bits while a DIU response frame has five residual bits.

I/O Transaction: A command frame which may be followed by a response frame. A node always returns a response frame. A DIU is not required to return a response frame.

I/O Chain: An ordered set of one or more transactions addressed to devices on one I/O network. A chain consists exclusively of either node transactions or DIU transactions. A chain is the smallest unit of I/O activity conducted by an IOS for a GPC.

Redundant Chains: A set of I/O chains designed to execute in loose simultaneity on a set of redundant I/O networks. The transactions within each chain are in a one to one correspondence with the transactions in the other chains. This reflects the one to one correspondence of redundant DIUs among the networks.

I/O Request: A set of one or more I/O chains each of which executes on a different I/O network. An I/O request consists exclusively of one set of redundant or non-redundant chains. An I/O request is the smallest unit of I/O activity conducted by I/O System Services for a user.

Chain Execution: The activity carried out by an IOS which results in the transmission of command frames and the reception of response frames on an I/O network. The program which an IOS executes is under the direct control of I/O System Services and the indirect control of the user specifying the chain. When a user creates a chain of transactions, certain parameters must be specified which control the execution of the chain. I/O System Services then translates these specifications into a program which is stored in DPM and which executes when I/O System Services starts that chain. The activity is initiated by I/O System Services but executes independently of the program running in the GPC.

Interprocessor Communications

Companion I/O Record: A transaction, chain or I/O request record residing on the IOP that is the dual of a transaction, chain, or I/O request record created by the user on the CP. These records are communicated to the IOP from the CP and are required for the execution and processing of the I/O requests.

Event: A signal which is observed by the GPC Real Time Operating System. Events interrupt the co-processor and are used to activate/deactivate tasks on the co-processor.

Flag: A passive signal that does not interrupt the co-processor. Flags may be observed or ignored by tasks on the co-processor.

Queue Management

Queue Manager Task: A process which initializes and controls all access to an I/O service.

Service Request: A request to process an I/O request, cycle a spare link, or restore a network element.

I/O Request Priority Queue: A prioritized queue which is constructed to control the processing of I/O requests. The queue is a linked list and is ordered based on the priorities of the I/O requests. The queue is used by the Priority Queue Manager to determine the next I/O request to be processed.

Posting Task: A task that is scheduled through the GPC Real Time Operating System based on the scheduling requirements of its associated I/O request. When scheduling requirements of the I/O request have been met, the task posts a service request to I/O request priority queue.

APPENDIX B: I/O SERVICE OPERATING RULES: NETWORK TOPOLOGY, GPC CONNECTIVITY AND I/O REQUEST DEFINITION

I/O Service: Definition and Operation

1) An I/O Service provides access either to one regional network or to a set of one or more local networks.

2) An I/O Service to a set of local networks operates those networks in parallel.

3) I/O Requests are specific to one I/O Service. It consists of a set of chains, at most one per network within the service.

4) All chains in an I/O Request are started at the same time. The I/O Request is completed and data becomes available to a user, when all chains within the request are completed.

5) Chains on parallel networks can be used to allow corresponding devices on each network to be accessed at approximately the same time. The degree of simultaneity which can be achieved is determined by three factors: the rate at which the IOS samples its Interface Command Register, the amount of time required to issue a Start Chain command, and the reproducibility of the response time for corresponding external devices.

6) A network is out of service from the time errors are detected on that network until a reconfiguration has been effected. In this context, a reconfiguration consists of either a network interface switch or a network reconfiguration. When a network is out of service, no user chains are executed on that network, however, service to other networks in that I/O Service is not affected.

7) Node status collection and spare link testing will be conducted simultaneously on all parallel networks within an I/O Service.

Network Topology Rules

1) Nodes will be connected in a way which would require at least 3 port failures to isolate any node or set of nodes from the rest of the network. This is the so called "minimum cut set".

2) At most one DIU will be connected to a node.

3) At most one GPC will be connected to a node.

4) A node may be connected to both a GPC and a DIU.

5) Parallel networks need not be connected in identical ways nor do they need to contain the same number of nodes or the same number of DIUs. In this way, user can trade throughput for reliability.

GPC Connectivity and Network Interfaces

1) A network has at most one interface per GPC channel, i.e. redundant root links to a network from a GPC come from distinct channels. Thus the maximum number of network interfaces connecting a GPC to a network is equal to the number of channels in the GPC.

2) Parallel networks are local networks in that they are used exclusively by one GPC for normal operations for long periods of time. However, more than one GPC may be physically connected to these networks and are therefore capable of taking over control and use of these networks in response to failure conditions. These spare connections are made ready and initialized as if they were to be used but remain dormant until activated by some higher controlling process such as the system manager.

3) Redundant network interfaces (i.e. root links to the same network) must have their IOSes occupy corresponding address spaces within their respective channels. This facilitates dual ported memory testing and allows modifications to chain programs and chain data to be made simultaneously to all redundant interface to the network.

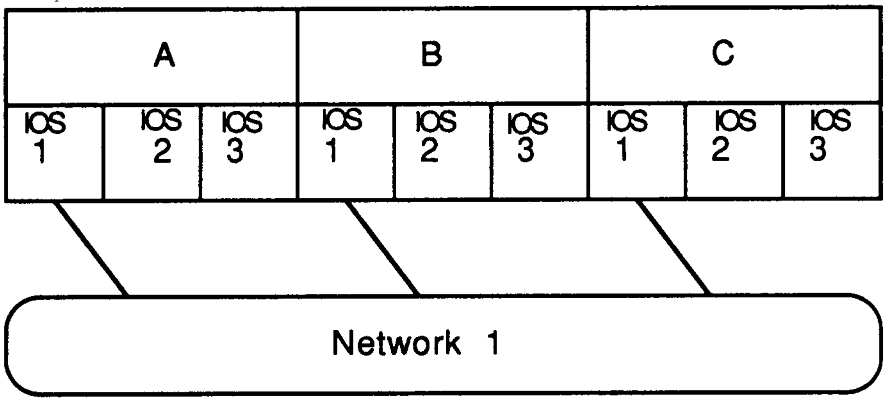


Figure B-1: Correct Redundant Root Link Connection of a GPC to a Network

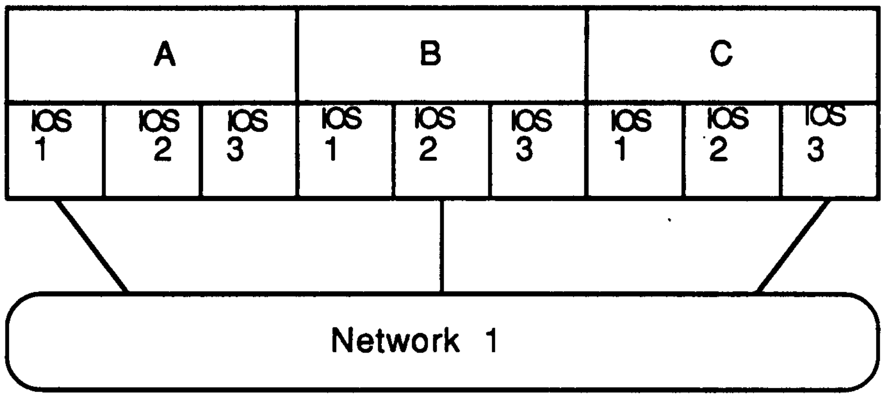


Figure B-2: Incorrect Redundant Root Link Connection of a GPC to a Network

I/O Request Definition

1) I/O Request Definitions determine whether an I/O Service is being used for reliability or throughput. They may access redundant devices simultaneously for greater reliability or they may access non-redundant devices for greater throughput.

2) An I/O Request may run chains on a subset of networks in an I/O Service, however, unused networks in the service remain idle during the execution of the request.

I/O Request Construction

1) A transaction ID returned by the Create Transaction procedure may only be used in one Create Chain procedure call.

2) A chain ID returned by the Create Chain procedure may only be used in one Create I/O Request procedure call.

APPENDIX C: INPUT OUTPUT SEQUENCER (IOS)

1.0 OVERVIEW

The Input Output Sequencer (IOS) is an autonomous asynchronous interface between an AIPS General Purpose Computer (GPC) and an I/O network. It resides on the shared bus of the GPC and can be accessed by either the Computational Processor (CP) or the I/O Processor (IOP). A major function of the IOS is to carry out detailed communication with I/O devices on the network as well as with the network nodes, off-loading the GPC from lower level I/O functions. A simplified block diagram of the AIPS I/O organization is shown in Figure C-1.

The IOS is connected to a node of the I/O network via a bidirectional connection, which is called a root link. When activated by the GPC, the IOS can transmit on the network via its root link. The IOS is a memory mapped device that can be accessed and/or programmed by the CP or the IOP to perform a sequence of instructions which is called a chain. The memory locations within an IOS form a dual port memory that can be accessed by processors one side and the IOS on the other. GPCs preload data into the memory for transmission to Device Interface Units (DIUs) or nodes. The IOS interfaces with the DIUs and nodes in a command response mode, which is referred to as a transaction. During a transaction the IOS transmits a command to a DIU or node and then, if required, waits a predetermined time for a response. The IOS writes response data into the locations of memory specified by the GPC in the chain. An IOS executes a chain only when it is enabled by its GPC.

Each channel of a redundant GPC may contain an IOS. These IOSs, if connected to individual networks, can all be active simultaneously. However, if all of the IOSs are connected to the same I/O network, then only one should be enabled to transmit at a time. A GPC channel may also contain more than one IOS for redundancy. When an IOS is commanded to start, it first contends (polls) with all other active IOSs for the use of the network if the network is shared among several GPCs. If it wins, it then has exclusive use of the network and can send and receive messages to DIUs and nodes. If an IOS loses, it waits for the network to be quiet (no data traffic) for a fixed amount of time or for another poll to start before contending again. Provision is made within the IOS for starting a poll without waiting if a failure is perceived on the network.

A detailed explanation of the components of the IOS follows. It includes a description of the instruction format, memory assignments, register definition as well as chain examples for the IOS. For the purpose of this document a chain is defined as the instructions that are executed as a unit. All instructions within the IOS are 4 bytes long. All values are given in hexadecimal units unless otherwise specified.

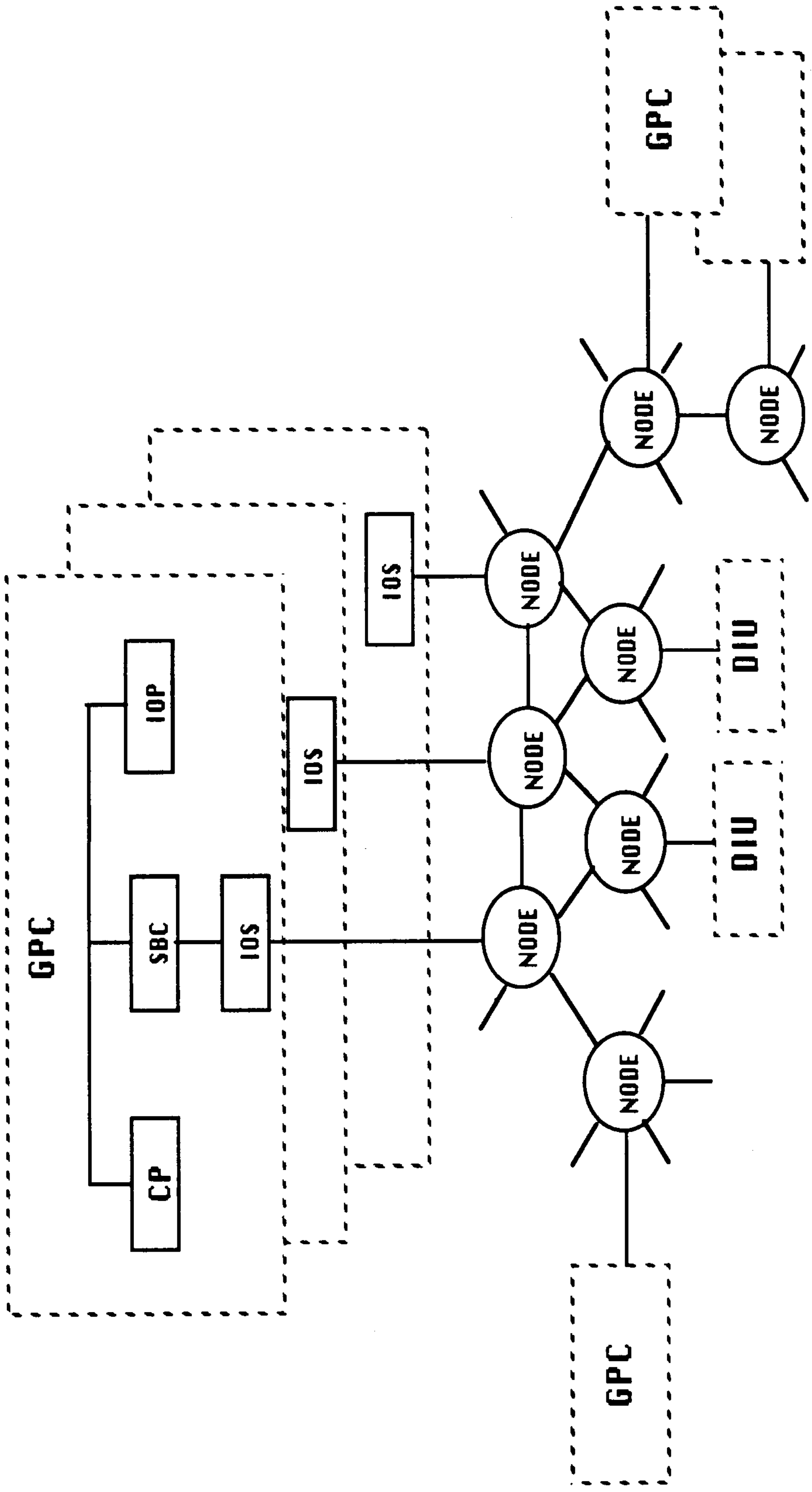


Figure C-1. AIPS I/O Organization

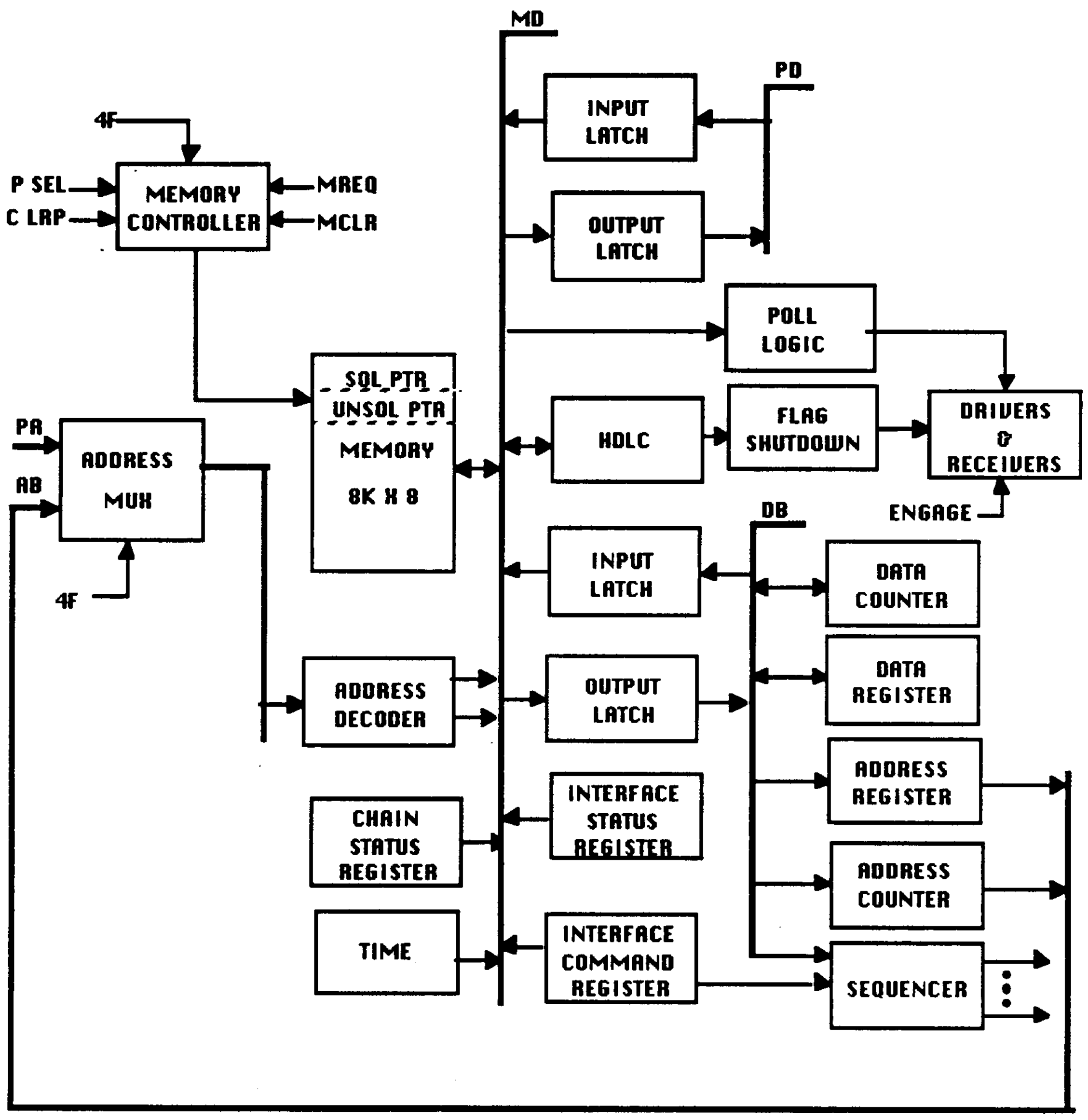


Figure C-2. IOS Block Diagram

2.0. IOS ORGANIZATION

A block diagram of the IOS is shown in Figure C-2. The IOS is programmed from a GPC which has access to the dual port memory and hardware registers. After loading the memory with the required chains the GPC then starts the IOS. The IOS can poll and run the chains without GPC intervention. An overview of the major components of the IOS is given below.

2.1 MEMORY CONTROLLER - The memory controller arbitrates memory accesses from the GPC and the IOS. The memory is time shared between them by the use of processor signal 4F. When 4F is high the processor can access the memory and when 4F is low the IOS can have access. The memory controller generates chip select, read-write and output enable at the appropriate times.

2.2 ADDRESS MULTIPLEXER - The address multiplexer selects between the GPC and IOS address buses. The output of the multiplexer is the memory address bus (MA). When 4F is high the processor address bus is connected to memory and when 4F is low the IOS memory bus is connected to the memory.

2.3 MEMORY - The IOS memory is a byte addressed memory containing 8192 bytes. It is used to store the chains, input packets and output packets. The first two bytes of memory are used as the solicited chain pointer and the second two bytes are used as the unsolicited chain pointer.

2.4 GPC INPUT LATCH - The GPC input latch is a buffer driver used to input byte wide data from the GPC data bus (PD) to the memory data bus (MD).

2.5 GPC OUTPUT LATCH - The GPC output latch is a buffer driver used to output data from the memory bus (MD) to the GPC data bus (PD).

2.6 IOS INPUT LATCH - The IOS input latch is a buffer driver used to input byte wide data from the internal IOS data bus (DB) to the memory data bus (MD).

2.7 IOS OUTPUT LATCH - The IOS output latch is a buffer driver used to output data from the memory bus (MD) to the internal IOS data bus (DB).

2.8 ADDRESS DECODER - The address decoder, decodes the individual hardware registers which are located in the memory space between 10 and 1F. The addresses of the hardware registers is given in section 4.

2.9 INTERFACE COMMAND REGISTER - The interface command register is a write only register that contains the commanded mode. See section 5.3.1 for a detailed description of the possible modes.

2.10 SEQUENCER - The sequencer is the main control element of the IOS. When started the sequencer fetches the instructions from memory, stores them internally, decodes and executes the microcycles by generating the appropriate control signals.

2.11 CHAIN STATUS REGISTER - The chain status register is a read only register that contains the chain and contention logic status within the IOS. See section 5.2.1.2 for a detailed description of the status reported.

2.12 INTERFACE STATUS REGISTER - The interface status register is a read only register that contains the status of the IOS. See section 5.2.1.1 for a detailed description of the status reported.

2.13 ADDRESS COUNTER - The address counter stores the current memory address that the IOS is using. It is used to point to where in memory the chain instructions are located. During an input instruction it points to the location where the incoming data byte is to be stored. In an output instruction it points to the byte to be output when the HDLC chip requests a byte. It is loaded during instruction fetches and incremented during the instruction microcycles.

2.14 ADDRESS REGISTER - The address register contain the fixed addresses used in the instructions. During an input instruction it contains the address of the HDLC input register. During an output instruction it contains the address of the HDLC transmitter holding register.

2.15 DATA COUNTER - The data counter contains data that is incremented during an instruction. During an input instruction it accumulates the byte count of the incoming data. During an output instruction it counts the number of bytes outputted until the message is complete at which time it signals the sequencer to terminate the instruction.

2.16 DATA REGISTER - The data register is used to temporarily store data within an instruction. During an input instruction it holds the incoming byte from the HDLC receiver register until a memory cycle can be performed to store it. During an output instruction it holds the next byte to be outputted until the HDLC transmit holding register requests it.

2.17 HDLC - The HDLC device contains independent transmitter and receiver sections. The HDLC transmitter section receives the data bytes, appends opening and closing flags, encodes, and transmits the data. The receiver section searches the data stream for an opening flag. When it detects one, it synchronizes with the data fields and decodes the data stream into bytes for storage. In both modes the device generates the handshaking signals necessary to run the interface. See Sections 5 for details on the operation and control of the HDLC registers.

2.18 FLAG SHUTDOWN - The flag shutdown logic guarantees that the external IO network transmissions lines are always left in the same state after use. This allows the data and poll logic to utilize the same transmission lines. See section 6 for details.

2.19 DRIVERS and RECEIVERS - These drivers and receivers allow the IOS to interface to the IO network. The drivers are enabled by an engage line from the GPC. The receivers are always enabled but the input is controlled by the HDLC device.

2.20 POLL LOGIC - The poll logic which allows the IOS to contend with other IOSs to gain control of the IO network. When enabled the poll logic monitors the IO network waiting for a quiet time and then starts a poll. If it wins it starts a solicited chain, and if it loses it waits for the next poll or quiet and tries again. See section 7 for additional details.

2.21 TIME DRIVER - The time driver allows the chain to read a time byte that appears on the shared bus.

3.0. INSTRUCTION FORMATS

The IOS can execute a limited number of instructions to perform its functions. The following paragraphs detail the form and function of the IOS instructions.

3.1. NOP (0000 0000)- This instruction updates the chain pointer to the address of the next sequential instruction. At the end of the NOP it will fetch that instruction.

3.2. BRANCH (2000 dddd)- This instruction will fetch the instruction contained at location 'dddd' and begin its execution. The Chain Pointer will be updated to point to the next instruction (dddd+4).

3.3. MOVE (40ss dddd)- This instruction will move a byte, located at any location 'ss' within the first 256_{10} bytes of IOS memory, to the byte location specified by 'dddd'. MOVE can be used to store the current value of a hardware register or store a preset value into a register.

3.4. MOVE IMMEDIATE (60xx dddd)- This instruction allows a constant, xx, to be stored into the destination dddd.

3.5. INPUT (801B dddd)- This instruction will store incoming HDLC bytes in the buffer area specified by 'dddd'. At the start of execution of this instruction the byte reserved for the input byte count is set to zero and the current value of the contention status is also stored within the buffer. As bytes are received they are stored at the specified buffer locations and an internal byte count incremented. A valid message always ends with a closing flag, which causes the IOS to then store the byte count, HDLC status registers and the TIME byte within the incoming packet buffer area. The INPUT instruction has now completed and the next sequential instruction is fetched and executed. The maximum number of data bytes that a single instruction can store is 122_{10}. If the INPUT contains more than 122_{10} data bytes, data will be lost. However, the buffer will never exceed the 128_{10} bytes allotted to it. The byte count which includes the status bytes, will also never exceed 80 (128_{10}). This instruction can also end if the time allotted for response is exceeded (the value programmed into the timer is reached without a data byte being received). However, in this situation none of the status information (HDLC IR & SR registers, time and byte count) is saved. An incoming data packet will always have the following format.

Byte count

HDLC IR register

HDLC SR register

TIME byte

contents of Chain Status Register

data (first byte)

data (last byte received)

3.6. OUTPUT (E01C ssss)- This instruction will transmit the bytes specified in the buffer starting at location 'ssss + 1'. The first byte at location 'ssss' contains the value of the expression, 80 - NB, where NB is the number of bytes to be transmitted. This instruction terminates when all the bytes have been transmitted.

4.0. MEMORY MAP

The following is the assigned memory locations in the dual port memory space of the IOS. Addresses 10 - 1F are hardware registers, however they are addressed the same as the RAM locations. All memory addresses, including the hardware registers are accessible from the CPU.

ADDRESS		FUNCTION
0	R/W	Solicited Chain Pointer - High Byte (RAM)
1	R/W	Solicited Chain Pointer - Low Byte (RAM)
2	R/W	Unsolicited Chain Pointer - High Byte (RAM)
3	R/W	Unsolicited Chain Pointer - Low Byte (RAM)

10	R	Chain Status Register
11	W	Interface Command Register
11	R	Interface Status Register
12	W	Timer Limit Register
13	W	Poll ID Register - 6 bit polling address
14	W	Poll Prio Register-3 bit prio & polling level
15	R	Time
16		Reserved
17		Reserved
18	R/W	HDLC Control Register 1 (CR1)
19	R/W	HDLC Control Register 2 (CR2)
1A	R/W	HDLC Control Register 3 (CR3)
1B	R	HDLC Receiver Holding Register (RHR)
1B	W	Address Register (AR)
1C	R	HDLC Interrupt Register (IR)
1C	W	Transmit Holding Register(THR)
1D	R	HDLC Status Register (SR)
1E		Reserved
1F		Reserved

With the exception of the addresses specified above, the rest of the dual port memory space can be used for any desired function. However, it should be noted that the MOVE instruction can only use the first 256_{10} addresses for the source byte.

5.0. REGISTERS

A description of the hardware registers and their use is contained in the following paragraphs. The hardware can execute two types of chains, solicited and unsolicited. Solicited chains are defined as command response chains and are meant to be executed when the GPC has control of the network. Unsolicited chains are defined as those that are performed when the GPC does not have control of the network but must accept all frames addressed to it. Unsolicited chains are not defined on the IO network, however, they are used as a vehicle while waiting for a poll to be won. On the IC network using the ICIS, unsolicited chains are executed whenever the GPC does not have the network, including while waiting for a poll to be won.

5.1. READ/WRITE REGISTERS

5.1.1. CHAIN POINTER REGISTERS

5.1.1.1. SOLICITED CHAIN POINTER (ADDR = 00 & 01) - The solicited chain pointer is used by the IOS to indicate where the next instruction of a solicited chain is located. When a new chain is to be started this location is loaded with the address of the first instruction to be executed. It must be loaded before an execute chain command is issued. As each chain instruction is fetched, this location is updated to point to the next sequential instruction. The GPC can read this location at any time. However, since the IOS writes the locations a byte at a time and the GPC can read them as a word, the value read by the GPC may be incorrect if a chain is executing. The GPC should not attempt to write these bytes while a chain is executing, since it cannot be guaranteed that the IOS is not presently also modifying them.

5.1.1.2. UNSOLICITED CHAIN POINTER (ADDR = 02 & 03) - The unsolicited chain pointer is used by the IOS to indicate where the next instruction of an unsolicited chain is located. When a new chain is to be started this location is loaded with the address of the first instruction of the unsolicited chain to be executed. It must be loaded before an execute chain command is issued. As each chain instruction is fetched, this location is updated to point to the next sequential instruction. The GPC can read this location at any time. However, since the IOS writes the locations a byte at a time and the GPC can read them as a word, the value read by the GPC may be incorrect if a chain is executing. The GPC should not attempt to write these bytes while a chain is executing, since it cannot be guaranteed that the IOS is not presently also modifying them. Unsolicited chains are identical to solicited chains and can execute any mix of instructions.

5.1.2. HDLC READ/WRITE REGISTERS

The following is extracted from the Western Digital data sheets on the HDLC chip (WD 1935). Definitions of bit polarity and sense have been modified to reflect what is seen by the AIPS system.

5.1.2.1. CONTROL REGISTER #1 (CR1) (ADDR = 18) - Control register 1 is used to specify the transmitter parameters and the transmitter and receiver enables. It can be loaded by a GPC or by a MOVE instruction in the chain.

<u>NOTE</u>: This register must always be loaded after CR2 and/or CR3. If CR2 and/or CR3 are ever changed, CR1 must again be reloaded after the change even if there are <u>no</u> changes being made to CR1.

7	6	5	4	3	2	1	0
ACT	ACT	TC	TC	TCL	TCL	DTR	MISC
REC	TRAN	1	0	1	0		OUT

5.1.2.1.1. ACT REC (bit 7) - Activate receiver bit when set to a ZERO (0), the receiver is enabled to accept a data stream. When set to a ONE (1), the receiver will ignore any frames on the network.

5.1.2.1.2. ACT TRAN (bit 6) - Activate transmitter bit when set to a ZERO (0), the encoder and transmitter are enabled to output onto the network. When set to a ONE (1) the HDLC device will not transmit data.

5.1.2.1.3. TC1 and TC0 (bits 5 and 4) - The transmit command bits program the device into the requested mode. In AIPS, the OUTPUT instruction will function properly only in the data mode. These bits and the modes that they generate are as follows:

bit 5	bit 4	MODE	FUNCTION
1	1	data	Outputs the contents of the transmitter holding register
1	0	abort	Generates an abort message (not used on AIPS)
0	1	flag	Transmits one flag character (not used on AIPS)
0	0	FCS	Generates the two CRC bytes and a closing flag (not used on AIPS)

5.1.2.1.4. TCL1 and TCL0 (bits 3 and 2) - These bits control the number of bits per character from the transmitter. In AIPS this has been defined as 8 bit bytes. The definition of these bits follows:

bit 3	bit 2	BITS PER CHARACTER
1	1	8
1	0	7
0	1	6
0	0	5

5.1.2.1.5. DTR (bit 1) - Data Terminal Ready, a modem signal that is not used in this design and should be programmed to a ONE (1).

5.1.2.1.6 MISC OUT (bit 0) - Miscellaneous Output, a control signal not implemented in this design and should be programmed to a ONE (1).

5.1.2.2. CONTROL REGISTER #2 (CR2) (ADDR = 19) - Control register #2 specifies the receiver parameters and other control functions as defined below. It can be loaded by a GPC or by a MOVE instruction in the chain.

7	6	5	4	3	2	1	0
EXT CONT	ADDR COMP	EXT ADDR	RCL 1	RCL 0	LOOP	SELF TEST	AUTO FLAG

5.1.2.2.1. EXT CONT (bit 7) - This bit extends the HDLC control field. It is not used on AIPS and must be programmed to a ONE (1).

5.1.2.2.2. ADDR COMP (bit 6) - This bit enables the on-chip address comparator. If set to a ZERO (0), the first byte after the opening flag will be compared to the byte stored in the AR register. If equal, the data bytes that follow will be output. If address compare is enabled, and the address does not compare, all data bytes following will be ignored. If bit six is set to a ONE (1) then address comparison is not performed in the chip and all bytes between the opening and closing flag are presented to the interface. In AIPS, the IOS and the NODES do not use the address compare function but the ICIS does.

5.1.2.2.3. EXT ADDR (bit 5) - This bit extends the HDLC address field. It is not used on AIPS and must be programmed to a ONE (1).

5.1.2.2.4. RCL1 and RCL0 (bits 4 and 3) - These bits specify the receiver character length. In AIPS this has been defined as 8 bit characters. The definition of these bits is as follows:

bit 4	bit 3	BITS PER CHARACTER
1	1	8
1	0	7
0	1	6
0	0	5

5.1.2.2.5. LOOP (bit 2) - Specifies HDLC loop mode, a test function, not implemented in the IOS. This bit should always be programmed to a ONE (1).

5.1.2.2.6. SELF TEST (bit 1) - Chip diagnostic mode, not implemented through the IOS. This bit should always be programmed to a ONE (1).

5.1.2.2.7. AUTO FLAG (bit 0) - When this bit is set to a ZERO (0) and the transmitter is enabled, the chip will issue constant flag characters between frames. The IOS design utilizes this function and therefore must be set to a zero during an output instruction.

5.1.2.3. CONTROL REGISTER #3 (CR3) (ADDR = 1A) This register is used to control the number of residual bits in a transmission. It can be loaded by a GPC or by a MOVE instruction in the chain. The definitions of these bits are as follows:

7	6	5	4	3	2	1	0
X	X	X	X	X	TRES 2	TRES 1	TRES 0

5.1.2.3.1. BITS 7 through 3 - Unused

5.1.2.3.2. TRES 2 - 0 (bits 2, 1 and 0) - These bits define the number of residual bits to be sent as the last character in a transmission. Messages sent to and from a NODE contain three (3) residual bits. Messages to and from DIUs contain five residual bits. The definition of these bits are as follows:

bit 2	bit 1	bit 0	RESIDUAL BITS/FRAME
1	1	1	No residual bits sent
1	1	0	1
1	0	1	2
1	0	0	3
0	1	1	4
0	1	0	5
0	0	1	6
0	0	0	7

5.2. READ ONLY REGISTERS

5.2.1 STATUS REGISTERS

5.2.1.1. INTERFACE STATUS REGISTER (Read Only) (ADDR = 11) - This register contains status of the interface.

5.2.1.1.1. UNSOLICITED INPUT RCVD (bit 0) - Set if any unsolicited input is received. This bit is set when the end of that message is detected, and reset whenever the IOP reads this register.

5.2.1.1.2 STUCK ON HIGH (bit 1) - This bit is set to a one, if the receiver has been high for the IDLE time. It is cleared whenever a transition is detected.

5.2.1.2. CHAIN STATUS REGISTER (ADDR = 10) - This register contains status of both the chain and the contention logic.

CHAIN STATUS REGISTER (Read Only)

7	6	5	4	3	2	1	0
Chain Comp	Contention State		Possession Default	Data Tx Fail	Poll Tx Fail	Any Rcv Fail	Any Rcv Good

5.2.1.2.1. CHAIN COMPLETE (bit 7) - This bit is set whenever the current chain has completed. Chain complete is defined as an IOS transition from solicited mode to unsolicited mode without the POLL bit in the command register set. It is reset whenever the poll bit is changed to a one in the interface command register or the IOS transitions from the unsolicited mode to the solicited mode.

5.2.1.2.2. CONTENTION STATE (bits 6 and 5) - This is the present state of the poll logic only. The following are the possible states that can be indicated.

5.2.1.2.2.1. INACTIVE, BUS RELEASED (00) Both bits are zero whenever the IOS is not attempting to gain control of the network.

5.2.1.2.2.2. WAIT (01) This IOS has been instructed to acquire the network, however no POLL has completed since the request occurred.

5.2.1.2.2.3. ATTEMPTED (10) This IOS has entered and lost at least one POLL sequence since being commanded to acquire the network.

5.2.1.2.2.4. POSSESSES (11) This IOS presently has possession of the network.

5.2.1.2.3. POSSESSION DEFAULT (bit 4) - Indicates that the IOS possesses the network and detected an incoming POLL length bit on the network. If a chain is in progress when this happens, it will continue to completion. This bit is reset whenever the POLL bit in the Command Register is set to zero.

5.2.1.2.4. DATA TX FAIL (bit 3) - Indicates that a data bit was detected at the receiver during a command frame transmission. The chain will continue to completion. This bit is reset whenever the POLL bit in the Command Register is set to zero. This bit can only be set during a network possession.

5.2.1.2.5. POLL TX FAIL (bit 2) - Indicates that a data length bit was detected during a Poll Sequence. This bit is reset whenever the POLL bit in the Command Register is set to zero.

5.2.1.2.6. ANY RCV FAIL (bit 1) - Indicates that at least one response frame has been received with a protocol error in it. It is reset whenever a new poll begins or the IOS transitions from the unsolicited mode to the solicited mode.

5.2.1.2.7. ANY RCV GOOD (bit 0) - Indicates that at least one response frame has been received without a protocol error. It is reset whenever a new poll begins or the IOS transitions from the unsolicited mode to the solicited mode.

5.2.2. HDLC READ ONLY REGISTERS

5.2.2.1. RECEIVER HOLDING REGISTER (RHR) (ADDR = 1B) This read-only register contains the received bytes as they are decoded from the frame. When executing an input instruction the IOS automatically reads this location and stores the received characters into the specified location in the dual port memory.

5.2.2.2. INTERRUPT REGISTER (IR) (ADDR = 1C) This read-only register contains status information on the state of the HDLC operation. It can be read by the GPC or with a MOVE instruction within a chain. Bits 7 through 3 will accumulate information such that if the IR is read after several operations, it will have the "OR" of all those frames. The definition of the bits within this register is as follows:

7	6	5	4	3	2	1	0
REOM NO ERR	REOM WITH ERR	XMIT NO ERR	XMIT WITH URUN	DISC	DRQI	DRQO	INTRQ

5.2.2.2.1. REOM NO ERR (bit 7) - When equal to a ZERO, this bit indicates that the frame was received without errors. If this bit is read before the closing flag is detected, it will not have been updated from the last frame.

5.2.2.2.2. REOM WITH ERR (bit 6) - When equal to a ZERO, this bit indicates that the frame was received with errors. If this bit is read before the closing flag is detected, it will not have been updated from the last frame. The errors that are reported here are: CRC, overrun, invalid frame and aborted frame.

5.2.2.2.3. XMIT NO ERR (bit 5) - When equal to ZERO, this bit indicates that the transmitted frame had completed without underrun errors.

5.2.2.2.4. XMIT WITH URUN (bit 4) - When equal to ZERO, this bit indicates that the transmitted frame had extra bytes inserted by the chip because the data was not available to the transmitter in the allotted time.

5.2.2.2.5. DISC (bit 3) - This bit is used with modems and in this system has no meaning.

5.2.2.2.6. DRQI (bit 2) - When set to a ZERO, this bit indicates that there is a byte available in the Receiver Holding Register (RHR). Reading the RHR sets this bit to a ONE. The hardware uses a buffered copy of this bit when storing bytes into dual port memory during an input instruction.

5.2.2.2.7. DRQO (bit 1) - When set to a ZERO, this bit indicates that the Transmit Holding Register (THR) is empty and requires another character to prevent an underrun error. Storing a byte into the THR sets this bit to a ONE. The hardware uses a buffered copy of this bit during an output instruction to read a byte from the dual port memory and store it into the THR.

5.2.2.2.8. INTRQ (bit 0) - This bit is set to a ZERO whenever at least one of the other bits in the IR register is set to a ZERO. This bit is set to a ONE whenever the IR is read. A buffered copy of this bit is used to terminate a normally completing input or output instruction.

5.2.2.3. STATUS REGISTER (SR) (ADDR = 1D) -` This read-only register contains status information that, when used in conjunction with the contents of the Interrupt Register, define the cause of the error.

7	6	5	4	3	2	1	0
RI	CD	DSR	MISC IN	RCVR IDLE	RRES 2 /ERR	RRES 1 /ERR	RRES 0 /ERR

5.2.2.3.1. RI (bit 7) - A modem signal not implemented in this interface.

5.2.2.3.2. CD (bit 6) - A modem signal not implemented in this interface.

5.2.2.3.3. DSR (bit 5) - A modem signal not implemented in this interface.

5.2.2.3.4. MISC IN (bit 4) - An input discrete not used in this interface.

5.2.2.3.5. RCVR IDLE (bit 3) - When set to a ZERO, the receiver is idle, i.e. a frame is not in process.

5.2.2.3.6. RRES2 /ERR (bit 2) - This bit has a dual role. If bit 7 in the Interrupt Register is a ZERO, then this bit is part of a binary number (see section 5.2.2.3.8) representing the number of residual bits received. If bit 6 in the Interrupt register is set to a ZERO, and this bit is set to ZERO then an aborted or invalid frame was detected.

5.2.2.3.7. RRES1 /ERR (bit 1) - This bit has a dual role. If bit 7 in the Interrupt Register is a ZERO, then this bit is part of a binary number (see section 5.2.2.3.8) representing the number of residual bits received. If bit 6 in the Interrupt register is set to a ZERO, and this bit is set to ZERO then an overrun error was detected. An overrun error indicates that a received byte was not removed from the Receiver Holding Register before the next byte was received. That first byte will be lost.

5.2.2.3.8. RRES0 /ERR (bit 0) - This bit has a dual role. If bit 7 in the Interrupt Register is a ZERO, then this bit is part of a binary number (see below) representing the number of residual bits received. If bit 6 in the Interrupt register is set to a ZERO and this bit is set to ZERO, then a CRC error was detected.

bit	bit	bit	RESIDUAL BITS/FRAME
2	1	0	
1	1	1	No residual bits sent
1	1	0	1
1	0	1	2
1	0	0	3
0	1	1	4
0	1	0	5
0	0	1	6
0	0	0	7

5.2.3. TIME (read only) (ADDR = 15) - This byte contains a value that is slaved to the system timer, incremented by a 66_{10} microsecond clock and capable of measuring 16.830_{10} milliseconds. It can be read by the GPC or by a move instruction in the chain. It is automatically appended to all incoming frames that complete in a valid manner.

5.3. WRITE ONLY REGISTERS

5.3.1 INTERFACE COMMAND REGISTER (Write Only) (ADDR = 11) - This register contains the necessary control bits to operate the IOS. The following are valid commands used to control the IOS. The END CHAIN command transitions the IOS from solicited to unsolicited mode. The STOP CHAIN command turns the IOS off.

START CHAIN WITH POLL = 94

START CHAIN WITHOUT POLL = 80

END CHAIN = 84

STOP CHAIN = 20 (followed by a 00 command to prime the interface for the next command)

INTERFACE COMMAND REGISTER (Write Only)

7	6	5	4	3	2	1	0
EXECUTE CHAIN	X	STOP IMM	POLL	SPOLL	EXECUTE UNSOL CHAIN	X	X

5.3.1.1 EXECUTE CHAIN (bit 7) - When only the execute chain bit is set to a one (1), this commands the hardware to fetch and start executing instructions starting at the address stored in the Solicited Chain Pointer. The chain will start even if a Poll was neither started nor won. If however, a poll is to be won first before starting the chain, then bits 7, 4 and 2 must be set to a one. The hardware will then start the polling logic, start an unsolicited chain pointed to by the unsolicited chain pointer (usually an input instruction) and when a poll is won, automatically start the chain at the location pointed to by the solicited chain pointer.

5.3.1.2. Bit 6 - Not used by the IOS.

5.3.1.3. STOP IMM (bit 5) - When the stop immediately bit is set to a one (1) the hardware will turn off the IOS. Whatever function the IOS is now performing will be terminated. This allows the GPC to stop the hardware if it is caught in a loop or otherwise malfunctioning.

5.3.1.4. POLL (bit 4) - Whenever the poll bit is set to a one (1) the logic will attempt to gain control of the network by joining the next possible poll sequence. At the end of a chain this bit must be reset.

5.3.1.5. SPOLL (bit 3) - Whenever the spoll bit is set to a one (1), the hardware will immediately start to poll. The hardware will not wait for the start of a new poll from another site or an idle condition on the network. At the end of a chain this bit must be reset.

5.3.1.6. EXECUTE UNSOL CHAIN (bit 2) - This bit is only recognized by the hardware when set in conjunction with the execute chain bit, bit 7. If bits 7 and 2 are both set to a one (1), the hardware will execute the chain starting at the location pointed to by the unsolicited chain pointer. If a GPC desires to first gain control of a network, it sets bits 7, 4 and 2 to a one and all others to a zero (0). The hardware will enable the polling logic, start the unsolicited chain at the location pointed to by the unsolicited chain pointer (usually an input instruction) and when a poll is won, automatically start the chain at the location pointed to by the solicited chain pointer.

5.3.1.7. Bits 1 and 0 are not used.

5.3.2. HDLC WRITE ONLY REGISTERS

5.3.2.1. ADDRESS REGISTER (AR) (ADDR =1B) - This write-only register contains the address that the chip is to use for comparison if on-chip address recognition is being used. If on-chip address detection is not used, the contents of this register will be ignored.

5.3.2.2. TRANSMIT HOLDING REGISTER (THR) (ADDR = 1C) - This write-only register holds the next data byte to be transmitted. The hardware loads a byte into this register during an Output instruction whenever DRQO is set.

5.3.3. TIMER LIMIT REGISTER (Write only) (ADDR = 12) - The timer limit register contains the current value to be used to time out an instruction. A non-zero value written to the timer limit register allows the timer to function. The timer is initialized at the beginning of each instruction and as each incoming data byte is detected. If an instruction does not complete or an incoming data byte is not detected in the programmed number of microseconds, the current instruction is terminated and the next sequential instruction started. A new value stored in the timer limit register will be accepted when the next instruction is started or the next incoming byte is accepted during an input instruction.

5.3.3.1. TIMER LIMIT VALUE The timer limit is the number of periods of the clock 2F. 2F has a period of approximately 2 microseconds. The timer has a range of 2 to 512 microseconds.

5.3.4. POLL REGISTERS

5.3.4.1. POLL PRIORITY REGISTER (Write only) (ADDR = 14) - The Poll Priority Register contains the six high order polling bits. The three bits labeled PRIO, are used for the initial priority of this IOS. They will automatically increment after each poll sequence loss until they contain all ones, at which time incrementing is inhibited and the maximum priority held. Since the initial state of the PRIO bits are not saved, this register must be reloaded whenever the initial polling state is required. The three bits labeled LEVEL, are the high order bits of the poll sequence. For the IO network LEVEL 2 is set to a one, LEVEL 1 and LEVEL 0 are set to a zero. It can be loaded by a GPC or by a MOVE instruction in the chain.

7	6	5	4	3	2	1	0
X	LEVEL 2	LEVEL 1	LEVEL 0	X	PRIO 2	PRIO 1	PRIO 0

5.3.4.2. POLL ID REGISTER (Write only) (ADDR = 13) - The Poll ID register contains the six (6) low order bits used in the polling procedure. These bits normally contain the address that this IOS uses for polling. It can be written into by the GPC or by a MOVE instruction within the chain.

7	6	5	4	3	2	1	0
X	X	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
		(MSB)					(LSB)

6.0. FLAG SHUTOFF SYNC

The IOS uses the same IO network lines to communicate and to poll. In order to be able to perform both functions on the same lines all operations must leave the lines in a known state. The HDLC protocol allows the signalling lines to be left in either state, and in fact the device used to generate the HDLC protocol does leave the line in either state depending upon the data content of the message. The IOS contains logic which upon sensing the end of a message utilizes the closing flags to turn off with the line in a low state without generating any extraneous data. When the next output message is started, the first flags are used to turn the logic back on to the state that the HDLC device attempted to leave the line. Again this is done without generating any extraneous bits. The polling logic is fabricated so as to always end with the line low.

7.0. POLLING

The IOS contains logic which allows it to contend with the other IOSs for use of the IO network. If the IOS is to contend for the network, the bits in the interface command register must be set to execute, poll and execute unsolicited mode. The logic will start the chain pointed to by the unsolicited pointer and simultaneously prime the polling logic. The reason for having a unsolicited chain is to give the IOS a place to wait for the poll to complete. Therefore, there must be an input instruction without the timer running where the IOS will "hang" waiting for the poll to be won.

The polling logic waits for either a poll to begin or the bus going quiet for 512 microseconds. When either occurs, the logic will assert a start bit for 48 microseconds. This gives all other IOSs time to recognize the start of a poll and join if required. At the end of the poll bit the logic compares the state of its input line with the state of its output line. If another IOS is joining the poll, the input line will also be high and the IOS must continue to poll to determine who will win. It next asserts the fixed priority bits, one at a time for 28 microseconds, followed by the variable priority bits and its address bits. At the end of each 28 microsecond period it compares its output to what it perceives on the bus. If what it hears is the same as what it is transmitting it must continue to the next bit as no decision can be made. If it hears a zero while it is transmitting a one, then it knows it has won because it has a higher value than all others that are contending. If it hears a one while it is transmitting a zero, then it knows it has lost because it has a lower value than at least one other contender, and it will stop transmitting and wait for another poll to begin. When the IOS decides that it has won it will abort the unsolicited chain and do a context switch to the solicited chain and start to execute it.

The variable priority bits are incremented after each poll sequence loss until they reach the maximum value of 7. They will remain at this value until written into by the program or the chain. If an IOS detects a data bit during its polling it will terminate the poll and set an error bit.

8.0. DUAL-PORT OPERATION

The IOS utilizes a time shared 8k x 8 memory for program and input output buffer storage. This memory can be alternately accessed by the GPC and the IOS. Each site has independent access to the memory for four CPU clock periods.

8.1 TIMING The dual-port memory utilizes the CPU clock signal 4F, which has a period of eight CPU clocks. When 4F is high the GPC has access to the memory and when 4F is low the IOS has access to the memory. In the following discussion the IOS timing is discussed, the timing as it pertains to the GPC is identical but only happening on the opposite phase of 4F.

If the IOS requires the use of the memory it assert the signal MREQ. MREQ is recognized on the first rising edge of CPU clock after 4F falls, which causes a chip select to the memory to be asserted. (4F changes state on a falling edge of CPU clock.) Chip select is three clock periods wide. The memory cycle is terminated by MCLR being asserted for one CPU clock period and chip select being deasserted.

When a write is specified, the read/write line will be low. One clock period after chip select is asserted, a write enable signal to the memory is asserted. If a read is specified, the read/write line will be high and on the next rising edge of CPU clock after chip select is asserted, an output enable will be asserted. By delaying output enable, none of the memory switching transients are seen.

The operation of the dual-port memory from the GPC side is identical except the memory request is initiated with the falling edge of PSEL and terminated with the assertion of

CLRP. The GPC can only make memory requests during the time that 4F is high. The address multiplexers are also driven by 4F.

9.0. HDLC PROTOCOLS

The HDLC bit orientated protocol was chosen for use on AIPS. HDLC allows automatic address detection, control information and a cyclic redundancy error word to detect transmission errors. In the IOS automatic address detection and the control byte are not used. The IOS operates in a command response mode at all times. It sends a message to a site and then waits for a response only when it has control of the IO network.

An HDLC frame contains an opening flag, address byte, control byte, data bytes (in AIPS up to 119_{10}), FCS byte, FCS byte and a closing flag. The opening and closing flag are identical and consist of a zero, followed by six ones and a zero. It is not possible for a flag to look like data since the HDLC protocol specifies that within the data field after five continuous ones a zero is added.

10.0. ENGAGE

The AIPS GPCs generate a voted engage signal which is used to enable external functions. In a faulty GPC this signal will not be asserted. The IOS uses this signal to enable its bus driver that connects it to the IO network. Therefore, a faulty GPC and/or faulty IOS can be disconnected and prevented from bring down the IO network.

11.0. BUFFER FORMATS

A typical chain will contain both input and output instructions. Each of these instructions must have buffer areas within the IOS's memory. The input buffers contain the messages that the IOS receives from Nodes and DIUs. The output buffer areas contain the messages that the IOS sends to Nodes and DIUs. There are no restrictions on where in memory inputs or output are stored. The following is the format of the input and output messages.

11.1. INPUT BUFFER FORMAT: The third and fourth byte of the input instruction point to a location in memory where the IOS will store an incoming message. Each incoming message contains a five byte preamble before the data part of the message. The first byte contains the byte count, which is the number bytes received plus the four additional bytes of the preamble. This can be used as an offset to point to the last byte of the buffer. If the input instruction is terminated by the timer expiring, then this byte will contain zero even if a partial message had been received before the message stopped. The second and third bytes contains the HDLC IR and SR registers respectively. These bytes are used to check for HDLC protocol errors. The fourth byte contains a time tag as recorded at the end of the input instruction. The fifth byte contains the contents of the Chain Status Register. From the sixth byte on is the data content of the message. To recap, input buffers within the memory all have the following format:

Byte Count

HDLC IR Register

HDLC SR Register

Time

Content of Chain Status Register

data (first byte)

data (last byte)

In the case of a response from a Node the input format will be as follows:

Byte Count

HDLC IR Register

HDLC SR Register

Time

Content of Chain Status Register

Node Address

Port Activity Seen

Transmission Errors Seen

Valid Frame Seen

Error in Node Messages Seen

Node Port Configuration

Sum Check

Residue Bits (3 bits residue + 5 bits FCS)

FCS (next 8 bits of FCS)

FCS (last 5 bits of FCS + 3 bits of pad)

11.2. OUTPUT BUFFER FORMAT: The third and fourth bytes of the output instruction contain the address within the IOS memory of the output buffer for this instruction. The byte located at this location is 80 - NB, where NB is the number of bytes in this output message. Following the byte count is the rest of the message. Since the longest message that can be received has been defined as 128_{10} bytes, and each input message contains a 5 byte preamble and 2 FCS bytes, the maximum data part of an output message can only contain 121_{10} bytes. If more than 121_{10} bytes are specified, the receiving location will truncate the message. The format of the output buffer is as follows:

Byte Count (80 - NB)

data

last data byte

12.0. EXAMPLE CHAINS

The following are intended to show how a Chain is programmed in the IOS.

12.1 EXAMPLE #1 - This example shows a chain which programs the HDLC chip and then does an Output frame followed by an input frame. The GPC stores the following values into the IOSs memory. (The IOS is a byte oriented device. For simplicity, the columns value or contents are two bytes and the columns labeled IOS location or address show the address of the high order byte. i.e. 0100 @ 8CX000 means that a 01 is stored at location 8CX000 and a 00 is stored at location 8CX001. The value of X indicates in which channel of a GPC the IOS is located. i.e. X = 1 for channel A, X = 2 for channel B, X = 4 for channel C. The high order bit of X is the high order bit of the address of the dual-port memory.)

The GPC writes the following locations:

xx	8CX013	Value of low order polling bits
4y	8CX014	Value of high order polling bits
94	8CX011	Commands IOS to execute chain, execute unsolicited and poll

The last store writes into the interface command register which instructs the IOS to enter a POLL as soon as it detects one starting, or to start a POLL if it sees the bus go idle. The IOS meanwhile starts to execute the unsolicited instructions starting at location 200. As soon as this IOS thinks it won a POLL, it terminates the unsolicited chain and starts the solicited chain at location 100. (All values below are given in HEX.)

INST	ADDRESS	CONTENTS	DESCRIPTION
--	8CX000	0100	Solicited Chain Pointer
--	8CX002	0200	Unsolicited Chain Pointer
	.	.	
	.	.	
0005	8CX100	4015	MOVE the current value of
	8CX102	01F1	time to location 01F1 (This could be done to find out when the solicited part of the chain started)
0006	8CX104	401C	Read the IR register to clear
	8CX106	01F2	any prior status
0007	8CX108	60FC	Store an FC in CR3. Sets the
	8CX10A	001A	chip to send 3 residual bits.
0008	8CX10C	60FE	Store an FE in CR2. Puts the
	8CX10E	0019	chip in the auto flag mode. This is mandatory to guarantee that all receivers will see the flag character and no extraneous data.
0009	8CX110	60BF	Store a BF in CR1. This
	8CX112	0018	enables the chip in the data mode and turns on the transmitter. (CR1 must be loaded after CR2 and CR3) Flags will now be sent until data is loaded into the THR.

0010	8CX114 8CX116	E01C 1000	Enter the OUTPUT mode. The byte count is read from location 1000 and the data bytes starting at location 1001 are transmitted. When the byte count reaches 80 the output instruction ends.
0011	8CX118 8CX11A	0000 0000	There must be at least one instruction, that does not read or write the HDLC chip after an OUTPUT instruction, to allow the CRC bytes and closing flag time to be transmitted. In this example a NOP was used, but any non HDLC instruction could be used.
0012	8CX11C 8CX11E	607F 0018	Store a 7F in CR1. This instruction turns off the transmitter and turns on the receiver.
0013	8CX120 8CX122	401C 01F3	Read the IR register and store it in location 01F3. This will clear the status before the next use of the HDLC chip.
0014	8CX124 8CX126	401D 01F4	Read the SR register and store it in location 01F4.
0015	8CX128 8CX12A	60FF 0012	Store a FF in the timer limit register and enable it to run (Timer = 512 micro).
0016	8CX12C 8CX12E	801B 4000	Enter the INPUT mode. Location 4000 will be cleared to accept the incoming byte count. If no data is received the IOS will wait here for 512 microsecond before going on to instruction #17. If any data byte is received before a timeout, the timer will be restarted. The IOS will stay in this

			instruction until a closing flag is received or the timer expires or in the case of an infinite input message the GPC terminates the chain.
0017	8CX130 8CX132	6000 0012	Disable the timer.
0018	8CX134 8CX136	2000 0348	BRANCH to the next instruction to be executed in this chain at location 0348. (This is an example of how bypassing might be done. The next executable instruction will be at location 0348).
0001	8CX200 8CX202	4015 01F0	MOVE the current value of time to location 01F0. (This could be done to log the time that this device was first enabled.)
0002	8CX204 8CX206	6000 0000	Turn off the timer.
0003	8CX208 8CX20A	801B 1F00	Enter the INPUT mode and store the frame starting at location 1F00. In an IOS, the system will hang at this instruction for a poll to be won since there are no unsolicited messages on the I/O network.
0004	8CX20C 8CX20E	2000 0204	In an IOS there would be only these two instructions. This BRANCH allows the IOS to return to unsolicited mode without the need to restore pointers.

A possible way that a solicited chain could always end is the following. The last instruction in the chain does a branch to a location that performs the desired chain termination. The advantage of this is that the solicited chain pointer will always have a known value in it whenever a chain has gone to completion.

nnnn	xxxx	2000	This is the last instruction of the chain.
	xxxx+2	0FF0	It specifies BRANCH to 0FF0.
	8CXFF0	6084	This is an END CHAIN command.
	8CXFF2	0011	It turns off the POLL and places the IOS in the execute unsolicited mode. The solicited chain pointer will contain the value 0FF4, which can be used to verify that the chain has completed.

Report Documentation Page

National Aeronautics and Space Administration

1. Report No. NASA CR-181874	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Advanced Information Processing Systems: Input/Output System Services		5. Report Date August 1989
		6. Performing Organization Code
7. Author(s) Thomas Masotto and Linda Alger		8. Performing Organization Report No.
		10. Work Unit No. 506-46-21-05
9. Performing Organization Name and Address The Charles Stark Draper Laboratory, Inc. 555 Technology Square Mail Station 3B Cambridge, MA 02139		11. Contract or Grant No. NAS1-18565
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes
NASA Langley Technical Monitor: Felix L. Pitts

16. Abstract
The purpose of this report is to document the functional requirements and detailed specifications for the Input/Output (I/O) Systems Services of the Advanced Information Processing System (AIPS). This introductory section is provided to outline the overall architecture and functional requirements of the AIPS system. Section 1.1 gives a brief overview of the AIPS architecture as well as a detailed description of the AIPS fault tolerant network architecture, while section 1.2 provides an introduction to the AIPS systems software. Sections 2 and 3 describe the functional requirements and design and detailed specifications of the I/O User Interface and Communications Management modules of the I/O System Services, respectively. Section 4 illustrates the use of the I/O System Services, while Section 5 concludes with a summary of results and suggestions for future work in this area.

17. Key Words (Suggested by Author(s)) Fault Tolerant Processor, I/O Network, Advanced Information Processing System (AIPS), I/O Transaction, I/O Request, AIPS System Software, I/O User Interface, I/O Communications Management	18. Distribution Statement Unclassified - Unlimited Subject Category 62

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 193	22. Price