1.0 OVERVIEW

The KLIPA driver is the lowest level of support for the CI-20. The driver is responsible for coordinating all access to the device and for implementing the queued interface between the device and the operating system. It also must provide services for diagnostic functions as well as for "DMA" functions specified in the SCA layer.

The driver is part of the PHYSIO system. The decision to place the KLIPA within the bounds of PHYSIO was made in order to minimize the effort within TOPS-20 to interface to the KLIPA. The KLIPA is the first device in the PHYSIO system that is not an RH20 and therefore we will have to "teach" PHYSIO how to manage such devices. It seemed appropriate to make this alteration to PHYSIO so that any other additions to the KL device repertoire will be easily integrated into the monitor.

This document describes the structure of the KLIPA driver, the specific services it performs and its relationship to the other CI-related services and to the monitor in general.

1.1 References

1. PHYKLP functional specification (R60SPC:KLPFUN). 10-AUG-83 (Grant/Miller)

2. LCG CI Port Architecture Specification, 11-July-83 (Keenan)

3. IPA20-L Error Spec, 30-Sept-82 (Holewa)

4. Systems Communication Architecture, 20-July-82 (Strecker)

1.2 Purpose Of This Document

This document describes how PHYKLP implements the services described in the functional specification [1]. Ideally, this document is a template for the implementor to follow and therefore will serve as a "road map" to the code. Portions of this document will appear as commentary in the source module so that maintainers will benefit from this design plan.

1.3 Driver Functions

The KLIPA driver is a TOPS-20 device driver. It is generally a passive service that responds to requests from higher level components and simply performs the device-specific operations required by the requests. Because the KLIPA is a smart and somewhat independent device, the driver must perform some actions on its own without being directed by higher-level services. However, these operations are transparent to the rest of the monitor.

The driver is responsible for maintaining the port-to-port virtual circuits to other CI drops. That is, the driver must locate the other CI drops, and negotiate a "three-way handshake" initialization in order to establish a "reliable communication path" between the nodes. In support of this operation, the driver polls the CI drops from time-to-time to detect any changes to the CI configuration. It turns out that the other nodes are also polling the CI, and therefore nodes may recognize one another during either of the polling actions. Whereas some nodes will only allow the port-to-port circuit when its poller detects another node (a la the HSC-50), the TOPS-20 KLIPA driver will cooperate with any node that wishes to establish a port-to-port circuit.

The only active part of the driver is the polling operation and the concomitant "virtual circuit initialization" operation. All other data transfers, including datagrams, messages and DMA transfers are requested explicitly by SCA. The interfaces to PHYKLP and SCA are diagrammed below:

```
   +----------+                            +--------------+
   !  SCA     !                            !    PHYSIO    !
   +----------!                            +--------------+
        \  ^  config change                 / poll
         \  \ data received                / interrupt
 DMA      \  \                            /
 messages  \  \                          /
 datagrams  \  \--+----------------+</
            ---> !    PHYKLP      !
                 +-------+--------+
                         ! device interface
                 +-------+--------+
                 !    IPA20       !
                 +----------------+
```

As can be seen, the PHYSIO interface is used only to drive the interrupts and as a source of polling calls. Unlike other PHYSIO drivers, PHYKLP is not called by PHYSIO to perform register reads, register writes and other "housekeeping" functions. PHYKLP appears to PHYSIO as a device other than an RH20 and one that is relatively self-contained.

1.4 Driver Components And Data Structures

The driver, PHYKLP, maintains the following structures

. The port control block (PCB). This is a "communication" area between the KLIPA and the driver. The PCB contains interlock words, queue headers and control information necessary for the KLIPA to monitor protocol. It is described fully in [2].

. System blocks. A system block is a data structure describing a node on the CI. Each system block contains information about the node as well as the state of the connection between this node and the remote node. The system block is the fundamental data structure for maintaining the virtual circuit between two nodes. There is a unique system block for each of the nodes on the CI.

The fundamental pieces of the driver are:

. Poller. This is the code that attempts to recognize new nodes appearing on the CI and to detect and report path failures in the CI. It is called as part of the scheduler.

. Interrupt service. This code is called by PHYSIO when the KLIPA interrupts the monitor. Most of the real work in the driver happens in the interrupt service routine, including the crux of the port-to-port VC initialization. The interrupt service routine is responsible for detecting device errors, protocol violations and system errors that affect the KLIPA.

. Message and datagram transmission. This service is used by SCA to send messages and datagrams.

. DMA services. These routines are used to create and validate CI buffers and to provide the software required portions of the DMA message service.

. diagnostic services. These routines support the DIAG% and SCS% functions specified for the KLIPA.

The remaining sections of this document will explore each of the pieces described in the overview as well as provide an operational description of PHYKLP.

2.0 DATA DESCRIPTIONS

2.1 PCB

The PCB, port control block, is the interface communications region between PHYKLP and the KLIPA. PHYKLP initializes the PCB and then identifies its physical address to the KLIPA. The PCB is as follows:

```
            +=========================================================+
  .PBBDT    |        Buffer Descriptor Table Starting Address         |
            +---------------------------------------------------------+
  .PBQEL    |                   Queue Entry Length                    |
            +---------------------------------------------------------+
  .PBQ3I    |               Command Queue 3 Interlock                 |
            +---------------------------------------------------------+
  .PBQ3F    |                 Command Queue 3 FLINK                   |
            +---------------------------------------------------------+
  .PBQ3B    |                 Command Queue 3 BLINK                   |
            +---------------------------------------------------------+
  .PBQ2I    |               Command Queue 2 Interlock                 |
            +---------------------------------------------------------+
  .PBQ2F    |                 Command Queue 2 FLINK                   |
            +---------------------------------------------------------+
  .PBQ2B    |                 Command Queue 2 BLINK                   |
            +---------------------------------------------------------+
  .PBQ1I    |               Command Queue 1 Interlock                 |
            +---------------------------------------------------------+
  .PBQ1F    |                 Command Queue 1 FLINK                   |
            +---------------------------------------------------------+
  .PBQ1B    |                 Command Queue 1 BLINK                   |
            +---------------------------------------------------------+
  .PBQ0I    |               Command Queue 0 Interlock                 |
            +---------------------------------------------------------+
  .PBQ0F    |                 Command Queue 0 FLINK                   |
            +---------------------------------------------------------+
  .PBQ0B    |                 Command Queue 0 BLINK                   |
            +---------------------------------------------------------+
  .PBRQI    |               Response Queue Interlock                  |
            +---------------------------------------------------------+
  .PBRQF    |                 Response Queue FLINK                    |
            +---------------------------------------------------------+
  .PBRQB    |                 Response Queue BLINK                    |
            +---------------------------------------------------------+
  .PBMFI    |             Message Free Queue Interlock                |
            +---------------------------------------------------------+
  .PBMFB    |               Message Free Queue FLINK                  |
            +---------------------------------------------------------+
  .PBMFB    |               Message Free Queue BLINK                  |
            +---------------------------------------------------------+
  .PBDFI    |             Datagram Free Queue Interlock               |
            +---------------------------------------------------------+
.PBDFF      |              Datagram Free Queue FLINK                  |
            +---------------------------------------------------------+
.PBDFB      |              Datagram Free Queue BLINK                  |
            +---------------------------------------------------------+
.PBRSV      |                  Reserved to Port                       |
            +---------------------------------------------------------+
.PBER1      |                    Error Word 1                         |
            +---------------------------------------------------------+
.PBER2      |                    Error Word 2                         |
            +---------------------------------------------------------+
.PBPBA      |                  PCB Base Address                       |
            +---------------------------------------------------------+
.PBPIA      |                      PI Level                           |
            +---------------------------------------------------------+
.PBIVA      |              Interrupt Vector Assignment                |
            +---------------------------------------------------------+
.PBCCW      |                Channel Command Word                     |
            +---------------------------------------------------------+
.PBRSP      |                  Reserved to Port                       |
            +=========================================================+
```

As can be seen, each of the queues requires three words: an interlock word, a forward pointer (FLINK) and a back pointer (BLINK).

All addresses in the PCB, and all addresses given to the KLIPA are physical memory addresses. This is so because the KLIPA accesses memory directly over either the E-bus or the C-bus. If it uses the C-bus, it has no way of performing a virtual to physical address translation. Consequently, the monitor must perform this translation in advance of specifying the address, and it must insure that the association remains valid until the KLIPA is finished with the data.

Queue interlock words are conventional "spin locks". That is, to lock a queue, one simply performs an

```
AOSE ADD
 {lock previously locked, test for timeout}
{lock successfully locked}
```

and one unlocks a queue by

```
SETOM ADD
```

This is easy for the processor, but not so easy for the KLIPA. In order to assist the KLIPA, a new microcode IOP function has been added that performs the equivalent of:

```
AOS EBUS,ADD
```

That is, the function directs the microcode to increment the value at ADD and to send the resulting value over the E-bus to the requesting device. It is then up to the KLIPA to determine if it owns the interlock by examining the result.

PHYKLP's lock procedure includes a "time out" if the lock is unavailable for a long time. If the spin lock does not succeed within one second, PHYKLP will consider the KLIPA to be malfunctioning and reload its microcode.

The PCB is essential to all interactions with the KLIPA. Therefore, anytime PHYKLP wishes to send a message or process an interrupt, it must determine the PCB address so that it can access the appropriate queue or status information.

In general, the PCB address is passed as an argument to the inner routines of PHYKLP. This is to allow additional KLIPAs someday without changing the bulk of PHYKLP. As can be seen in the next section, the system block associated with other node points to the appropriate PCB.

2.2 System Blocks

A system block describes the status of another node on the CI. PHYKLP builds a system block whenever it first encounters a node. In general, this will be when the poller receives a valid reply to a "request ID message", but it may also occur when the other node attempts to begin the initialization protocol and PHYKLP has not yet noticed the node. The system block is as follows:

```
       !=======================================================!
.SBANB !             Address of next system block              !
       !-------------------------------------------------------!
.SBAPB !      Address of associated port control block         !
       !-------------------------------------------------------!
.SBACD !      Address of associated channel data block         !
       !-------------------------------------------------------!
.SBVCS !    Path validity info     !    Dest vir cir state     !
       !-------------------------------------------------------!
.SBDSP !                   Destination port                    !
       !-------------------------------------------------------!
.SBDRQ !             Datagram return queue header              !
       !-------------------------------------------------------!
.SBLMB !              Local message buffer header              !
       !-------------------------------------------------------!
.SBSBI !          Reserved          !      SBI of this SB      !
       !-------------------------------------------------------!
.SBFCB !           Pointer to first connection block           !
       !-------------------------------------------------------!
.SBLCB !           Pointer to last connection block            !
       !-------------------------------------------------------!
```

```
.SBTWQ !                 FLINK for SCA work queue                   !
       !------------------------------------------------------------!
.SQBWQ !                 BLINK for SCA work queue                   !
       !------------------------------------------------------------!
.SBQOR !         Pointer to queue of outstanding requests           !
       !------------------------------------------------------------!
.SBDSS \                                                            \
       \                   Destination system                       \
       \                                                            \
       !------------------------------------------------------------!
.SBMMS !  Max mess size (bytes)   !    Max DG size (Bytes)          !
       !------------------------------------------------------------!
.SBDST !               Destination software type                    !
       !------------------------------------------------------------!
.SBDSV !             Destination software version                   !
       !------------------------------------------------------------!
.SBDSE !           Destination software edit level                  !
       !------------------------------------------------------------!
.SBDHT !               Destination hardware type                      !
       !------------------------------------------------------------!
.SBDHV !             Destination hardware version                   !
       !------------------------------------------------------------!
.SBDPC \                                                            \
       \          Destination port characteristics                  \
       \                                                            \
       !------------------------------------------------------------!
.SBTIM !        TODCLK at last message from this remote             !
       !------------------------------------------------------------!
.SBFLG !                        Flags                               !
       !============================================================!
```

As can be seen, the system block contains some fields and information maintained by SCA alone. That is, this data structure, although created by PHYKLP, is shared by SCA and PHYKLP. In effect, this is much like the other PHYSIO data structures, UDB and CDB, that contain information for several of the layers.

The fields describing the destination system are acquired from the "ID request" message exchanged by the ports.

A system block contains all of the information needed for PHYKLP to correctly process a request from a higher level service (viz. SCA). In particular, the system block contains the PCB to be used and the state of the port-to-port virtual circuit between that system and the local port. Therefore, each request made by SCA will use the system block to validate that the operation is legal for the current virtual circuit state (see section 3.1.2) and for locating the PCB and its data structures.

Interrupt requests begin with the PCB of the interrupting KLIPA and then determine the appropriate system block from the source node address in the received message (see section 3.3). Once the system block is determined, then PHYKLP can validate tha the operation is valid for the current virtual circuit state.

3.0 PHYKLP DESCRIPTION

3.1 State Representation Of The Driver

3.1.1 Initialization -

The port-to-port VC is opened by means of a "three way handshake" protocol. Diagrammatically, the protocol looks like:

(Figure 1)

Notes to figure 1:

> The messages, START, STACK and ACK are always to be sent from the lowest priority command queue (see 6.2.2).
>
> INIT is a holding state. In the case of a previously known system, INIT will "latch" until SCA releases it (see KLPOPN call). In the case of a newly located node, INIT releases immediately.
>
> Although error detection is not mandatory, it is recommended in order to avoid unnecessary interactions with higher-level services.

This protocol insures that the two systems reach the "open" state at the same time and that they maintain this state together.

The initialization sequence is begun once PHYKLP recognizes another node.

3.1.2 Other States -

PHYKLP operates as a state machine. Anytime the local system wishes to communicate with any system, or the local system receives data from another system, the state machine is used to validate the operation and to perform state transitions and notifications.

In particular, the system block describes the state of the connection between the local system and the remote system.

Any operation has a unique definition for the state in the system block. Data handling, error handling and calls to other services are all defined by the current state.

The initialization sequence states and transitions are defined in figure 1. Table 1 shows the additional transitions and operations.

Table 1

STATE \ EVENT	msg rec	datagram rec	send err datagram	send err message	VC closed	VC open
ISTRT	KLIPA error	drop it	ignore	NA	NA	tell SCA
ASTRT	""	""	""	""	""	tell SCA
RUN	tell SCA	tell SCA	""	Close VC goto INIT	tell SCA goto INIT	NA

The entries containing "tell SCA" indicate that PHYKLP will call the appropriate routine in SCA; each of the events makes use of a unique entry point in SCA.

3.2 Message Handling

The CI supports various types of port-to-port packets. There is a class of packets directed to SCA defined as messagess and datagrams. In addition, there are other types of packets, used for CI housekeeping functions and for maintenance, that are exchanged by the ports. In fact, the three initialization messages, START, STACK and ACK are of this latter class.

All of the housekeeping and maintenance packets are datagrams. That is, they are handled on a "best effort delivery" basis. The protocol that uses these packets must either be tolerant of lost packets or must provide some other means of guaranteeing no data is lost. The timers used in the initialization protocol are one example of how a datagram-based protocol tolerates and recovers from lost data.

Packets intended for SCA are sent only when a port-to-port virtual circuit has been established. Until the port-to-port circuit is RUNning, the only data exchanges are port housekeeping datagrams.

SCA message delivery is guaranteed by the KLIPA. That is, any SCA message is either correctly delivered, or the port-to-port circuit is closed. This low-level service off-loads considerable responsibility from the software thereby freeing it to provide other, more sophisticated services.

SCA datagram delivery is not guaranteed (as one might imagine).

3.2.1 Buffers -

All buffers, datagram and message, are provided by SCA. PHYKLP has no local buffer pool and takes no responsibility for acquiring or maintaining buffers.

PHYKLP will enqueue and dequeue buffers, datagram or message buffers, on request. Only SCA should use these routines as only SCA should be concerned with CI buffer management.

3.3 Message Formats

The CI portion of a packet looks like:

```
          +-----------------------------------------------+
 .PKFLI   !                    FLINK                      ! 0
          +-----------------------------------------------+
 .PKBLI   !                    BLINK                      ! 1
          +-----------------------------------------------+
 .PKRSV   !              Reserved for software            ! 2
          +---------+---------+---------+---------+-------+
 .PKSTS   !00<--->07!08<--->15!16<--->23!24<--->31!32<->35!
          ! Status  !  Flags  ! Opcode  !  Port   !  MBZ  ! 3
          +---------+---------+---------+---------+-------+
 .PKLEN   !00<------------->15!16<--------------------->35!
          !      PPD byte     !    Length of text data    ! 4
          !-------------------+---------------------------+
          !                                               ! 5
          !                                               !
          !                                               !
          !                  Text                         !
          !                                               !
          !                                               !  Queue
          !                                               ! Length
```

```
            !                                               !    - 1
            +-----------------------------------------------+
```

The opcode field describes the type of CI message. Packets intended for SCA are distinguishable from packets used by the ports.

The information above is called the "PPD header" and is meant for use by the port and its software driver. SCA has its own header information that is part of the "text" carried by the PPD packet.

3.4 Polling

The poller, which is part of the scheduler's context, is responsible for testing the validity of the port-to-port VC for each of the known nodes. Part of this testing function is to implement the "timer" noted in figure 1.

In addition, the poller attempts to "locate" newly "on line" nodes by periodically sending "request id" messages to any node not yet identified. This is necessary because the 'request id" message is a datagram, as are all port-to-port overhead messages, and therefore there is no guarantee when and if one such message and its repsonse will be successfully delivered. Therefore, each node on the CI must continually poll the other nodes in the fervent hope that one of them will succeed. Although this may sound hopelessly frustrating, the chances of not succeeding are actually quite small, and therefore the polling activity is infrequent. Were it not for the likelihood of losing a request id or its response, the polling of unknown nodes could be replaced by the requirement that each newly activated port simply send each other CI drop a "request id". This would be sufficient to guarantee complete connectivity of all CI ports. So much for the ideal world.

The poller's chief testing function is to implement the timers for the ISTRT and the ASTRT states. In addition to this, the poller could test the connectivity of the RUN connections by periodically sending "request id" messages to these ports as well. However, the SCA idle chatter is a more inclusive test of port-to-port connectivity and therefore will be used instead of this more primitive testing facility.

4.0 PHYKLP OPERATION

4.1 Major Interfaces

4.1.1 PHYSIO Interface -

PHYKLP is a device driver. As such, it is called by PHYSIO to perform device polling and to process interrupts.

The KLIPA does not support "vectored interrupts". PHYSIO expects its drivers to behave like RH20 channels, and therefore to request "vectored interrupts". In order to make all of this balance, the interrupt skip chain for channel 5 calls directly into PHYKLP when the KLIPA requests service. PHYKLP then fills in the CDB locations for its channel (RH slot 7) and calls PHYSIO as if a normal RH20-style vectored interrupt had occurred. PHYSIO is then in control of the rest of the interrupt processing.

PHYKLP is defined as a new type of channel controller. As a result, PHYSIO will acquire some new conventions for managing controllers other than RH20-style controllers.

The poller is called from PHYSIO's once-a-second routine. The action of the poller has already been described.

4.1.2 SCA Interface -

SCA relies on PHYKLP to send messages and datagrams. Also, it requires that PHYKLP be able to establish "named buffers" for DMA transfers and for PHYKLP to manage the DMA transfer including notifying SCA of the completion status. Finally, SCA relies on PHYKLP to inform it of nodes coming on-line or going off-line.

SCA is responsible for providing message and datagram buffers sufficient for all of the SCA traffic on the CI. In particular, SCA cannot rely on the meager number of datagrams that PHYKLP owns for all of the datagram traffic.

4.2 PHYKLP Structure

4.2.1 Initialization -

PHYKLP is initialized by a call to PPDINX. On this call, PHYKLP constructs the PCB, calls SCA so it may initialize and create the buffers that PHYKLP needs.

Once SCA has been called, PHYKLP will enable the KLIPA.

4.2.2 CI Configuration -

The CI is "configured" by a combination of interrupt level and polling activity. The poller continually sends request ID messages to other CI drops in an effort to locate previously unknown or not operating nodes.

The result of the polling activity, the ID message, is received at interrupt level. Once a node is recognized, or reintialized, the intialization sequence is carried out at interrupt level.

SC.ONL is called whenever the interrupt level code manages to create a port-to-port virtual circuit.

The configuration activity is ongoing as the CI configuration may change at any time.

PHYKLP will establish a VC with another node that it "sees" for the first time. However, if any existing port-to-port VC is closed, SCA must call KLPOPN before PHYKLP will attempt to open, or allow the other node to open, the VC again. This is provided to allow SCA to clean up its own data base and to prevent races between incarnations of the port-to-port VC. The system block for the remote node indicates whether a VC may be established.

4.2.3 Other Interrupt Activity -

Fig. 2 is a flow chart describing the general flow of the KLIPA interrupt service.

The interrupt code processes the KLIPA's response queue as well as detects changes in the device's state. The chief source of response queue entries is received messages and datagrams. Any received message must be given to SCA as PHYKLP has no interest in CI messages.

Datagrams are divided into two classes: CI housekeeping datagrams and SCA datagrams. PHYKLP distinguishes these classes by the PPD opcode byte in the received datagram. CI housekeeping datagrams are either intialization datagrams or maintenance datagrams.

All KLIPA interrupts are processed in the routine KLPINT. The KLIPA is a standard device in that it has a CONI word describing the events that promoted the interrupt.

A KLIPA interrupt occurs for one of the following reasons:

1. free queue error

2. response queue available

3. E-bus or M-bus error

4. CRAM parity error

The "normal" case is "response queue available" meaning that some data has arrived that needs the attention of the monitor. The other cases are all abnormal conditions that require special handling.

In all cases, the interrupt entry routine determines the PCB address for the interrupting device, and loads it into one of the permanent ACs.

A response is one of the following:

1. A received message or datagram

2. A sent message or datagram that PHYKLP specified the reponse bit for

3. A sent message or datagram that encountered an error

4. DMA transfer done

Messsages or datagrams meant for SCA are given directly to SCA from the KLPINT code. Of course, it is important that PHYKLP first determine if the port-to-port circuit is RUNning. If it isn't, then either the KLIPA or the monitor has made a disasterous error. The port to port curcuit is validated by finding the system block for the other node and then checking the virtual circuit state in the system block.

A datagram meant for PHYKLP is one of the "port housekeeping messages". In general, these will be initialization messages, but they could be messages generated by PHYKLP itself to read information from the KLIPA. If the datagram is an initialization message, then the state description in Figure 1 applies.

In all cases, table 1 defines the legal operations for data transmission.

All port messages must be sent using the lowest priority message queue. This is to allow any messages still enqueued to be processed before any of the VC establishing messages. If this is not done, then it is possible for the port-to-port VC to be reestablished before the old message is processed and therefore the old message will be erroneously sent. For example, consider the case where SCA has requested that a message be sent to node A, but the request is still on the command queue. If node A sends a START, then the port-to-port VC will be closed. If the STACK sent in reply is sent at a higher priority than the queued message, and the ACK in reply is received promptly, the port-to-port VC could be reestablished before the SCA message is processed. If this happens, then the KLIPA will send the message and node A will declare a protocol error when it receives thereby closing the port-to-port VC. Although this situation should eventually reach a quiescent state, much time could pass before the nodes can once again communicate.

Sending the START, STACK and ACK datagrams at the lowest priority avoids this unfortunate condition.

4.2.4 Data Services -

The other major portion of PHYKLP is the interfaces for sending messages and datagrams used by SCA. These are simply a collection of interfaces that manages the PPD layer of the packets.

In general, there are buffers owned by SYSAPs that the SYSAP wants returned when the contents have been transmitted. This is accomplished by setting the "response bit" in the packet flag byte. When the data has been sent, the pakcet will be placed on the response queue and a repsonse queue available interrupt produced. KLPINT simply informs SCA that the transmission is complete, and SCA must inform the appropriate SYSAP. The following cases are known to set the response bit:

. SCA control messages

. CFS messages

4.2.5 Packet Conventions -

As was mentioned earlier, the driver must provide physical addresses to the KLIPA. Consequently, the link words of linked structures are physical addresses, not virtual addresses.

In order to make accessing these structures as simple as possible, many of the structures contain a word reserved for use by software. PHYKLP stores the virtual address of the datum in this word and uses the routines PMOVE and PMOVEM to reference the physical location.

5.0 ERRORS

Errors are detected at interrupt level. They fall into two broad categories: transmission errors and device errors. Each time PHYKLP detects an error, it will either execute a BUG. or it will explicitly make a SPEAR entry.

5.1 Transmission Errors

Transmission errors occur for following reasons:

1. The port-to-port curcuit is in the wrong state

2. The CI itself is defective

3. The destination port does not ACK the message

It turns out there are other conditions, namely bugs in the ports, that can appear to be one of these conditions. For example, a defective collisions avoidance algorithm can appear to be a defective CI.

A transmission error on a message is a fatal error. That is, the SCA virtual circuit carrying the message ceases to be usable. PHYKLP must inform SCA of this condition.

A transmission error on a datagram is not a fatal error. If the datagram is one of SCA's, then PHYKLP can simply log the error and not bother notifying SCA. This is acceptable because datagrams may be lost for many reasons, transmissions errors being only one of the possible failures.

A transmission error on an intialization message is of concern to PHYKLP. As noted in Figure 1, errors during initialization may be ignored and caught as "time out" conditions. However, to expedite establishing the port-to-port VC, PHYKLP will detect transmission errors during initialization and attempt to recover immediately.

In general, a transmission error happens either because the destination port is not operational (no ACK), or the CI is defective. The former case is an acceptable failure and does not require logging or any other special action. The latter case must be logged and, in addition, PHYKLP will declare the port-to-port VC down and attempt to initialize the circuit again. This implies that SCA must be notified of a "node offline" condition.

5.2 Device Errors

These errors correspond to bits in the device's CONI status word. Each is detected at interrupt level and handled immediately.

5.2.1 Free Queue Error -

A free queue error implies that SCA has not provided sufficient buffering for the KLIPA. Of course a flurry of activity, such as many nodes coming online at once, can provoke this error as well.

A free queue error is simply an "data overrun" condition. It may not indicate a problem with the software or with the CI, and therefore PHYKLP will simply log the event. If, in fact, the error resulted in the loss of important data, the sending system will execute the appropriate recovery (e.g. close the VC).

5.2.2 CRAM Parity Error -

This means that the KLIPA has halted. Most likely the error is a result of the UCODE executing a predefined location that contains a parity error; in other words, it's a BUGHLT!

PHYKLP will log the error, including the LAR and will request that the KLIPA microcode be reloaded.

5.2.3 Bus Errors -

These are errors that may or may not result in a protocol error. That is, the data that is lost might be unimportant (i.e. a datagram).

Since PHYKLP has no way of knowing if any informationw as lost, it must assume that a bus error is a fatal error and shut all of the port-to-port VCs.

6.0 OPEN ISSUES

PHYKLP is being written to be initialized during PHYINI as well as later on the monitor start-up procedure. Once PHYSIO is modified to allow interrupt handling during initialization, this feature of PHYKLP will have to be thoroughly tested.