TOPS-20 KLIPA Driver Functional Specification

Arnold Miller
Clair Grant

Version 1.6

December 14, 1983

1.0 ARCHITECTURAL POSITION

PHYKLP.MAC is TOPS-20's port driver for the IPA-20 (KLIPA); thus, it is the lowest level of software support for the CI-20. It is responsible for coordinating all operating system access to the KLIPA; it does this by:

1. maintenance of the port-to-port virtual circuits between the local system and remote systems on the CI

2. providing an interface to the CI services desired by higher level protocols

The port driver provides a generally passive service that responds to requests from the higher level protocol module SCAMPI.MAC and 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.

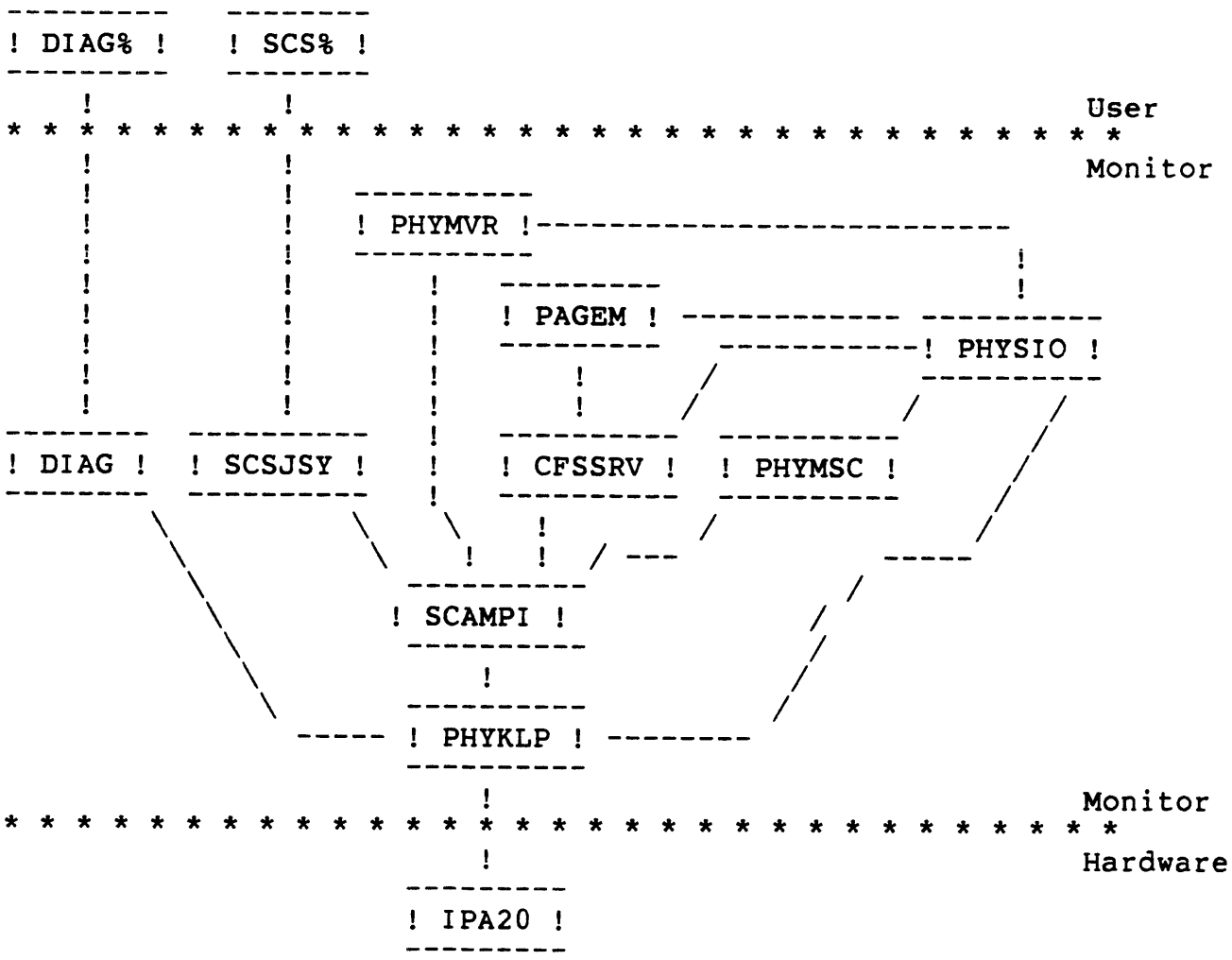


Figure 1 - CI-20 Software Components

2.0 RELATED STANDARDS AND SPECIFICATIONS

This document describes, at a functional level, those aspects of the KLIPA driver not specified in other documents. In order for you to understand this functional specification, you must be familiar with at least the terminology, if not the details, of the following:

1. LCG CI Port Architecture Specification, 18-Aug-83 (Dossa/Keenan)

   R60SPC:LCG-CI-PORT-ARCHITECTURE.SPC

2. IPA20-L Error Spec, 23-Nov-83 (Holewa)

   R60SPC:IPA20-ERROR-SPEC.TXT

3. Systems Communications Architecture, 20-July-82 (Strecker)

   R60SPC:SCA-CORP-SPEC.MEM

4. TOPS-20 SCA Functional Specification, 21-Nov-83 (Dunn)

   R60SPC:SCAFUN.MEM

5. TOPS-20 System Communication Service Tester, 10-May-83 (Grant)

   R60SPC:SCSTST.MEM

6. TOPS-20 Coding Standard, 23-Mar-83 (Murphy)

   DOC:CSTAND.MEM

The implementation details of the KLIPA port driver are described in:

TOPS-20 KLIPA Driver Design Specification, 16-Aug-83 (Grant/Miller)

R60SPC:KLPDES.MEM

3.0 TERMINOLOGY

The terms in the following list are defined in an attempt to clarify some potentially confusing topics discussed in this document.

1. port - an entity connected directly to the CI, capable of creating/destroying virtual circuits and transmitting packets; it may not have the ability to provide CI services to high-level protocols

2. node - an entity on the CI, possibly directly connected, possibly connected via a port; it provides higher-level protocols with an interface to the CI

3. port-to-port virtual circuit - the logical communications path between 2 ports

4. packet - a series of bytes (identified as a single entity) transmitted from one port to another

5. datagram - a packet which is not error controlled, that is, its delivery is not guaranteed.

6. message - a packet whose delivery is guaranteed

7. SCA connection - a logical communications path between 2 SCAs

4.0 KLIPA MICROCODE INTERFACE

The complete software/microcode interface is described in [1]. The following is a brief summary of the interface.

PHYKLP and the port communicate by using a queued protocol whose basic structure relies on 7 different doubly-linked queues. There are 4 command queues, a response queue, and 2 free queues (one for datagrams, one for messages). When PHYKLP wants to give the port a command, it places an entry on the tail of one of the 4 command queues. The port communicates with PHYKLP by putting entries on the tail of the response queue.

Both PHYKLP and the port use the free queues as a source of available buffers and as a repository of processed and discarded buffers.

This queue structure is defined and controlled by the Port Control Block (PCB), a data structure in the KL's memory. Both PHYKLP and the port read/write the PCB. The PCB is described in the GLOBAL DATA STRUCTURE section.

The CI-20 requires special KL10 microcode to allow the port to perform a read-pause-write increment memory reference. This is used by the port to interlock the queues in cooperation with AOSNs done by PHYKLP.

The port requires PHYKLP to perform the following initialization functions:

1. create initialize the PCB
2. Put a channel jump word in the EPT
3. Create a CCW in the PCB
4. load the KLIPA microcode
5. Start the KLIPA

The port reports error information by:

1. setting bits in the CSR
2. placing packets on the response queue
3. making entries in the PCB's error words

There may be at most one IPA-20 attached to any given KL-10 and the device must be located in RH slots 6 and 7. This restriction exists because there is no way to distinguish a KLIPA from a KLNI and therefore the monitor must have some a priori knowledge of where a KLIPA, if it exists, will be found. Aside from this restriction, however, we will make no conscious effort in the software to preclude multiple KLIPAs attached to a KL-10 in the future.

5.0 GLOBAL DATA STRUCTURES

5.1 Channel Data Block

PHYKLP creates and uses a channel data block (CDB), and uses the permanent AC conventions (P1-P4) used by the rest of the I/O system. There is also an entry in CHNTAB+7 for the KLIPA. However, the KLIPA has its own channel type; it is not an RH20. The KLIPA redefines some of the RH20 CDB words; they are:

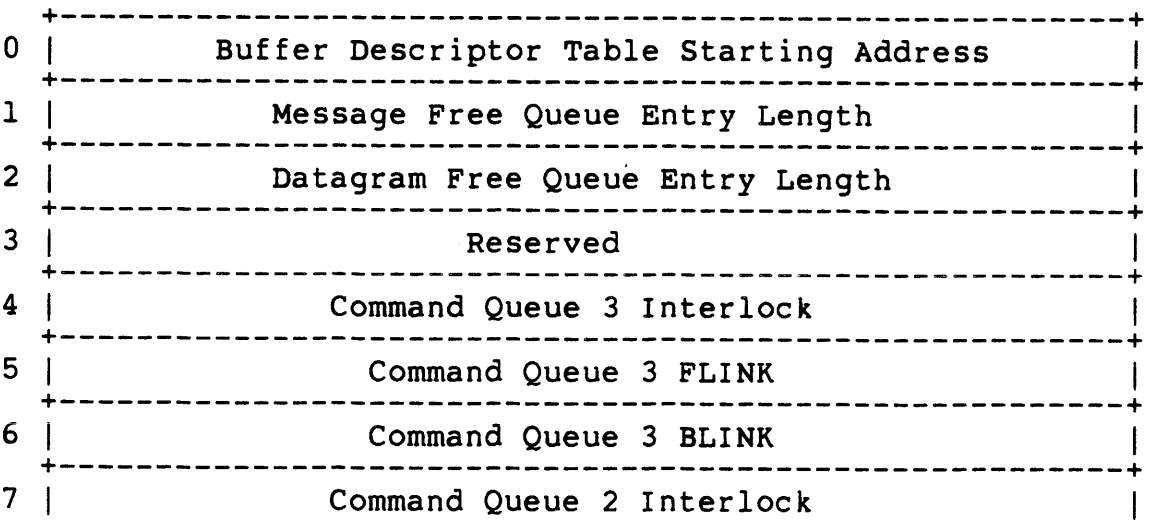
```
CDBFLG==CDBCAD                  ;FLAGS
        CB.DED==1B1             ;PORT IS DEAD
        CB.PST==1B2             ;PORT IS STOPPED
        CB.VOK==1B3             ;PORT VERSION IS OK (UCODE VERSION WORD)
CDBVER==CDBCAD+1                ;MICROCODE VERSION NUMBER
CDBLG0==CDBCC1                  ;LOGOUT WORD 0
CDBLG1==CDBCC2                  ;LOGOUT WORD 2
CDBLG2==CDBICR                  ;LOGOUT WORD 3
CDBQRQ==CDBRST                  ;NON-0 IF HAD TO REQUEUE A REQUEST
CDBCTR==CDBCL2                  ;MONOTONIC NUMBER,,FORK WHICH OWNS COUNTERS
CDBFQE==CDBCL2+1                ;MESSAGE,,DATAGRAM FREE QUEUE ERROR COUNT
CDBECW==CDBCL2+2                ;CCW FROM PCB AT ERROR
```

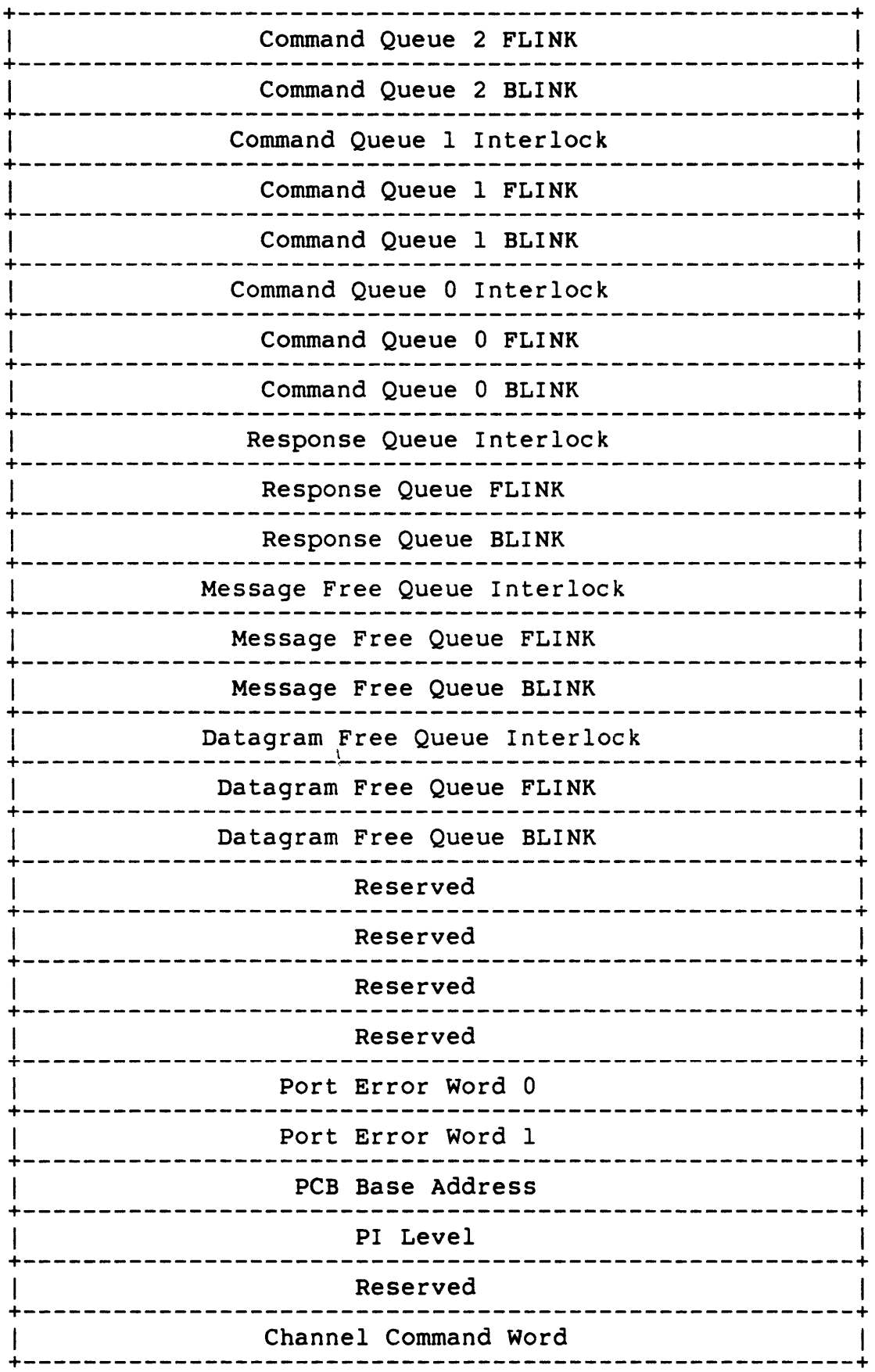
The KLIPA defines the device-depenedent words in its CDB as follows:

```
CDBPCB==CDBDDP                  ;ADDR OF PCB FOR MSCP CHANS
CDBSBS==CDBDDP+1                ;ADDR OF 1ST SYSTEM BLOCK FOR MSCP CHANS
CDBNOD==CDBDDP+2                ;OUR NODE NUMBER ON THIS PORT (MSCP CHANS)
CDBDGB==CDBDDP+3                ;LOC OF BUFFER FOR DATAGRAMS FOR PORT USE
```

5.2 Port Control Block

The PCB, officially defined in [1], is created by PHYKLP and is used by the port and PHYKLP as a communications region.

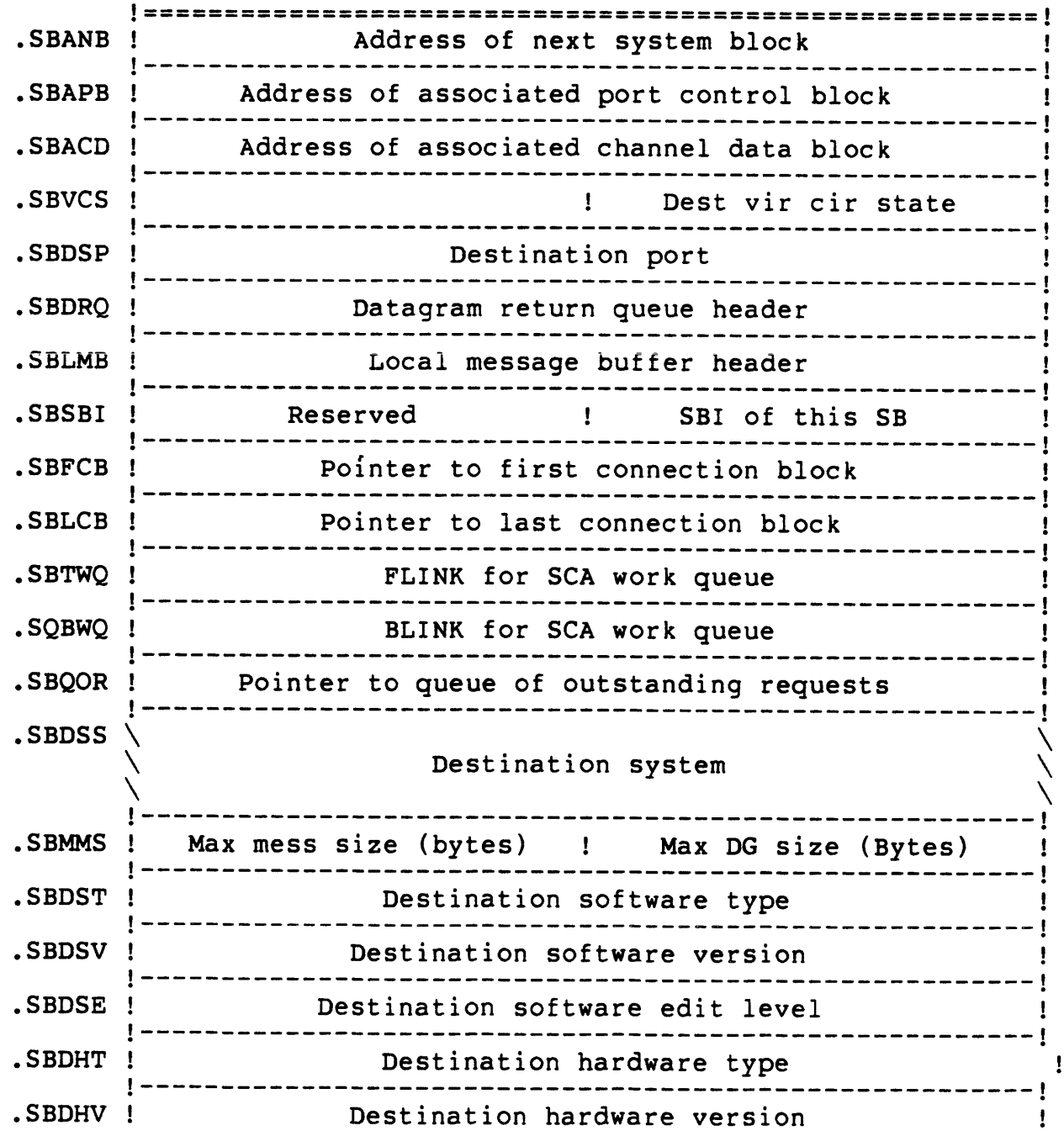


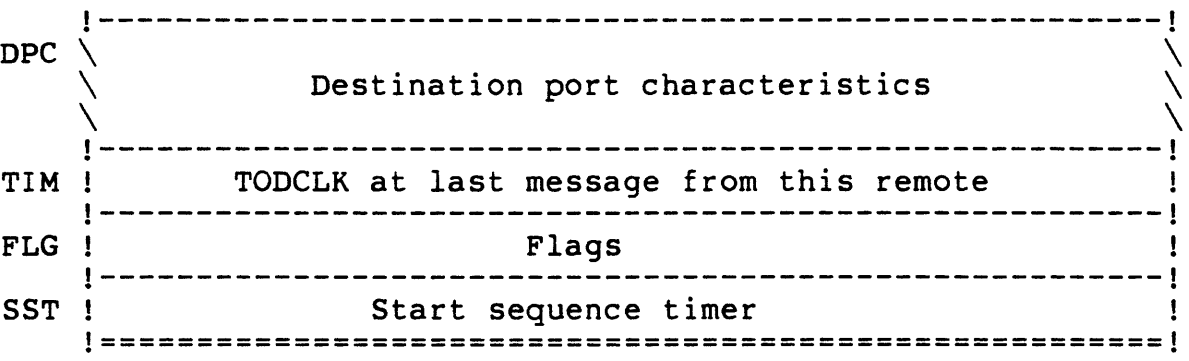

8 Command Queue 2 FLINK
9 Command Queue 2 BLINK
10 Command Queue 1 Interlock
11 Command Queue 1 FLINK
12 Command Queue 1 BLINK
13 Command Queue 0 Interlock
14 Command Queue 0 FLINK
15 Command Queue 0 BLINK
16 Response Queue Interlock
17 Response Queue FLINK
18 Response Queue BLINK
19 Message Free Queue Interlock
20 Message Free Queue FLINK
21 Message Free Queue BLINK
22 Datagram Free Queue Interlock
23 Datagram Free Queue FLINK
24 Datagram Free Queue BLINK
25 Reserved
26 Reserved
27 Reserved
28 Reserved
29 Port Error Word 0
30 Port Error Word 1
31 PCB Base Address
32 PI Level
33 Reserved
34 Channel Command Word

```
35 |                  Reserved to Port                  |
   +----------------------------------------------------+
```

5.3 System Block

PHYKLP creates a system block (SB) for each node it finds on the CI. The SB cells are defined as offsets in the device-dependent section of the controller data block (KDB). A system block is used by both PHYKLP and SCAMPI and looks like:




.SBDPC
Destination port characteristics
.SBTIM
TODCLK at last message from this remote
.SBFLG
Flags
.SBSST
Start sequence timer

6.0 PHYSIO INTERFACE

PHYKLP is a TOPS-20 device driver. As such, it is called by PHYSIO to perform channel polling and to process interrupts. PHYKLP does not provide many of the housekeeping and register operations which PHYSIO uses for RH20-type devices; such operations are simply not pertinent to KLIPA devices. Thus, there are few expected calls from PHYSIO to PHYKLP.

6.1 Initialization

PHYKLP's KLPINI routine is called from PHYSIO's PHYINI for channel initialization. KLPINI does the following:

1. resets the KLIPA
2. creates the channel data block (CDB)
3. fills in CHNTAB
4. sets up time out values for REQUEST-IDs and the START/STACK/ACK sequence

6.2 Once-a-second Device Polling

The poller, routine KLPCHK, is called by PHYSIO's PHYCHK routine by dispatching through KLPDSP.

The poller attempts to detect newly on-line nodes by periodically sending request-id packets to all nodes. This is necessary because the "request id" packet is a datagram, as are all port-to-port packets, and therefore there is no guarantee that it 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.

Once another node responds to a request-id, a three-way-handshake (START/STACK/ACK) initialization is used to open the port-to-port virtual circuit. The initialization packets are always sent from the lowest priority command queue (queue 3) and are timed by the poller. The protocol insures the two systems reach the open state by mutual agreeement.

6.3 Interrupt Service

The KLIPA does not support vectored interrupts; PHYSIO expects its device drivers to request vectored interrupts. In order to make all of this balance, the non-vectored interrupt skip chain will verify that the interrupt was generated by the KLIPA and then jump to PHYSIO as if a

vectored interrupt had occurred. A new channel is defined for the KLIPA so PHYSIO will not process the KLIPA as a funny RH20 device, but as a unique entity.

6.4 Channel Dispatch Vector

PHYKLP's channel dispatch vector is as follows:

```
KLPDSP::JRST KLPINI             ;0 - INITIALIZATION
        JRST DSPBUG             ;1 - STACK SECOND CHANNEL COMMAND
        JRST KLPSIO             ;2 - START I/O
        JRST DSPBUG             ;3 - POSITION REQUEST
        JRST DSPBUG             ;4 - RETURN BEST XFER
        JRST KLPINT             ;5 - INTERRUPT PROCESSING
        JRST KLPCCW             ;6 - MAKE CHANNEL XFER WORD
        JRST KLPHUN             ;7 - TRANSFER HUNG
        JRST KLPZAP             ;10 - RESET CHANNEL
        JRST KLPCHK             ;11 - PERIODIC CHECK
        JRST KLPEXT             ;12 - CHECK UNIT EXISTENCE
        JRST KLPCCA             ;13 - EXTRACT ADDRESS FROM CCW WORD
```

PHYKLP's controller/unit dispatch vector is as follows:

```
KLDSP:: JRST DSPBUG             ;0 - INITIALIZATION
        JRST DSPBUG             ;1 - START I/O
        JRST DSPBUG             ;2 - HANDLE INTERRUPT
        JRST DSPBUG             ;3 - ERROR RECOVERY
        JRST DSPBUG             ;4 - HUNG DEVICE
        JRST DSPBUG             ;5 - CONVERT BLK # TO CYLINDER/SURF-SEC
        JRST DSPBUG             ;6 - LATENCY COMPUTATION
        JRST DSPBUG             ;7 - START POSITIONING
        JRST DSPBUG             ;10 - ATTENTION INTERRUPT
        JRST DSPBUG             ;11 - SKIP IF POSITIONING REQUIREDD
        JRST DSPBUG             ;12 - STACK SECOND TRANSFER COMMAND
        JRST KLEXT              ;13- CHECK EXISTANCE OF UNIT
        RET                     ;14- CHECK FOR HALTED CONTROLLER
```

DSPBUG is a BUGHLT. We do not expect to get called from PHYSIO for this function if the channel is a KLIPA.

7.0 SCAMPI INTERFACE

The SCA protocol and its associated terminology are completely described in [3]. The following is a brief description of the services PHYKLP provides for SCAMPI to implement the protocol used by SYSAPs.

SCA defines 3 types of data transmissions:

1. messages
2. datagrams
3. named buffers

SCA data transmissions are sent only when a port-to-port virtual circuit has been established by PHYKLP and an SCA connection has been opened. SCA relies on PHYKLP to maintain the port-to-port virtual circuits. The states of a virtual circuit are 1) closed, 2) start-sent, 3) start-received, and 4) open.

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

SCA datagram delivery is not guaranteed so applications using SCA must be prepared for lost datagrams. When DECnet is implemented on the CI, it will use the datagram service.

SCA requires PHYKLP to establish named buffers and manage their transfers, including notifying SCA of the completion status.

SCA relies on PHYKLP to inform it of nodes coming on-line or going off-line.

Although SCA will test an existing port-to-port virtual circuit with idle chatter over an SCA connection on that virtual circuit, PHYKLP will still inform SCA when a virtual circuit must be closed, just in case the timing is such that PHYKLP detects the problem first.

7.1 Services Provided For SCAMPI

This section lists the routines in PHYKLP which are provided for SCA to manage use of the CI by SYSAPs.

The following is a list of error codes which may be returned by PHYKLP.

1. KLPX1 - no BHDs available

2. KLPX2 - no BSDs available
3. KLPX3 - no datagram buffers available
4. KLPX4 - no message buffers available
5. KLPX5 - KLIPA is not enabled
6. KLPX6 - KLIPA is in maintenance mode
7. KLPX7 - no KLIPA on system
8. KLPX8 - packet is bad
9. KLPX9 - no virtual circuit
10. KLPX10 - don't know our own CI node number

7.1.1 Port-to-Port Virtual Circuit Maintenance -

1. Open a port-to-port virtual circuit

   SCA must call KLPOPN for any system block that has previously had an open VC. Once a virtual circuit is closed, PHYKLP will not reopen it without a direct request from SCA. This is provided to allow SCA to clean up its own data base and to prevent races between incarnations of an open port-to-port VC.

   BLCAL. (OPENVC,<SBA>)

   ACCEPTS: SBA/ System Block Address
   RETURNS: +1 Failed. T1/ error code
   +2 Success

   Possible errors: KLPX3

2. Close a port-to-port virtual circuit

   BLCAL. (CLOSVC,<SBA>)

   ACCEPTS: SBA/ System Block Address
   RETURNS: +1 Failed. T1/ error code
   +2 Success

   Possible errors: KLPX3

7.1.2 Datagrams -

1. send a datagram

```
        BLCAL. (SNDDG,<SBA,DGA,LEN,FLG,PRI,PATH>)

ACCEPTS:        SBA/ System Block Address
                DGA/ Datagram Address
                LEN/ Length of Datagram
                        word count if high density
                        byte count industry compatible
                FLG/ Flags
                        F.RTB   ;1 - return buffer to SCA
                                ;0 - return buffer to free Q
                        F.SPM   ;1 - Send in high density mode
                                ;0 - Send in industry compatible mode
                PRI/ Priority (command queue number)
                PATH/ 0 - KLIPA selects
                      1 - Path A
                      2 - Path B


RETURNS:        +1 Failed. T1/ error code.
                +2 Success

Possible errors:  KLPX9
```

2. link datagram buffers onto the datagram free queue

```
        BLCAL. (LNKDFQ,<SBA,BFA>)

ACCEPTS:        SBA/ System Block Address
                BFA/ Datagram Buffer Address

RETURNS:        +1
```

3. unlink a datagram buffer from the datagram free queue

```
        BLCAL. (ULNKDG,<SBA>)

ACCEPTS:        SBA/ System Block Address

RETURNS:        +1 Failed. T1/ error code
                +2 Success -  T1/ Datagram address

Possible errors:  KLPX8
```

7.1.3 Messages -

1. send a message

```
        BLCAL. (SNDMSG,<SBA,MSGA,LEN,FLG,PRI,PATH>)

ACCEPTS:        SBA/ System Block Address
                MSGA/ Message Address
                LEN/ Length of Message
                        word count if high density
                        byte count industry compatible
                FLG/ Flags
                        F.RTB   ;1 - return buffer to SCA
                                ;0 - return buffer to free Q
                        F.SPM   ;1 - Send in high density mode
                                ;0 - Send in industry compatible mode
                PRI/ Priority (command queue number)
                PATH/ 0 - KLIPA selects
                      1 - Path A
                      2 - Path B

RETURNS:        +1 Failed. T1/ error code
                +2 Success

Possible errors:  KLPX9
```

2. link message buffers onto the message free queue

```
        BLCAL. (LNKMFQ,<SBA,BFA>)

ACCEPTS:        SBA/ System Block Address
                BFA/ Message Buffer Address

RETURNS:        +1
```

3. unlink a message buffer from the message free queue

```
        BLCAL. (ULNKMG,<SBA>)

ACCEPTS:        SBA/ System Block Address

RETURNS:        +1 Failed. T1/ error code
                +2 Success -  T1/ message address

Possible errors:  KLPX8
```

7.1.4 Named Buffers -

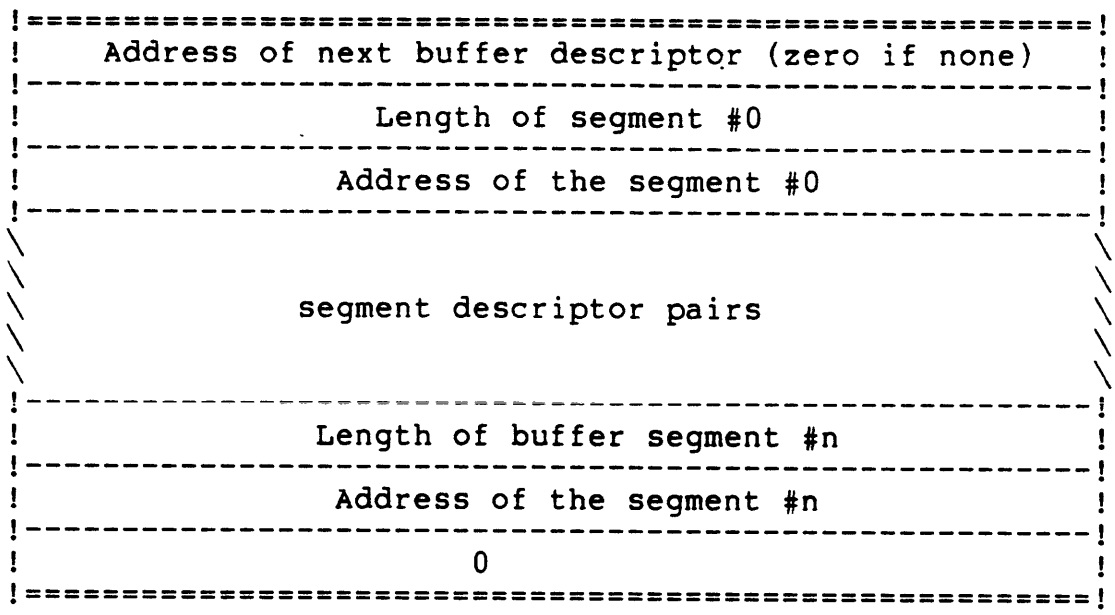
1. map buffers

```
BLCAL. (MAPBUF,<BDA>)

ACCEPTS:    BDA/ Buffer Descriptor Address
RETURNS:    +1 Failed. T1/ error code
            +2 Success  T1/ buffer name
```

Possible errors: KLPX1, KLPX2

The buffer descriptor has the following format:



2. unmap a buffer

```
BLCAL. (UMAP,<BNAM>)

ACCEPTS:    BNAM/ Buffer Name
RETURNS:    +1 Failed. T1/ error code
            +2 success
```

Possible errors:

3. send a buffer

```
BLCAL. (SNDDAT,<SNAM,RNAM,SBYT,RBYT,CID>)

ACCEPTS:    SNAM/ name of send buffer
            RNAM/ name of receive buffer
            SBYT/ Byte offset into send buffer
            RBYT/ Byte offset into receive buffer
            CID/ Unique code given back when operation completed
```

```
RETURNS:        +1 Failed. T1/ error code
                +2 Success

Possible errors:  KLPX9
```

4. request a buffer

```
        BLCAL. (REQDAT,<SNAM,RNAM,SBYT,RBYT,CID>)

ACCEPTS:        SNAM/ name of send buffer
                RNAM/ name of receive buffer
                SBYT/ Byte offset into send buffer
                RBYT/ Byte offset into receive buffer
                CID/ Unique code given back when operation completed

Returns:        +1 Failed. T1/ error code
                +2 Success

Possible errors:  KLPX9
```

7.1.5 Get The Local Port's CI Number -

```
        CALL LOCPRT

RETURNS:        +1 Failed. T1/ error code
                +2 Success. T1/ our CI port number

Possible errors:  KLPX7, KLPX10
```

7.1.6 KLIPA Maintenance Support - The maintenance functions do not require an SCA connection.

1. send a RESET or START maintenance packet

```
        BLCAL. (PPDSRS,<SBA,PKA,FUNC>)

ACCEPTS:        SBA/ System block address
                PKA/ Packet Address
                FUNC/ Function Bits ,, Start Address
                        F.SRS   ;1 - start
                                ;0 - reset
                        F.FRC   ;1 - force
                                ;0 - non-force

RETURNS:        +1 Failed - T1/ error code
                +2 Success
```

Possible errors: KLPX7

2. send maintenance data

```
BLCAL. (PPDSMD,<SBA,PKA,SNAM,RNAM,SBYT,RBYT,CBACK>)

ACCEPTS:    SBA/ System block address
            PKA/ Packet Address
            SNAM/ name of send buffer
            RNAM/ name of receive buffer
            SBYT/ Byte offset into send buffer
            RBYT/ Byte offset into receive buffer
            CBACK/ Callback address

RETURNS:    +1 Failed - T1/ error code
            +2 Success
```

Possible errors: KLPX7

3. request maintenance data

```
BLCAL. (PPDRMD,<SBA,PKA,SNAM,RNAM,SBYT,RBYT,CBACK>)

ACCEPTS:    SBA/ System block address
            PKA/ Packet Address
            SNAM/ name of send buffer
            RNAM/ name of receive buffer
            SBYT/ Byte offset into send buffer
            RBYT/ Byte offset into receive buffer
            CBACK/ Callback address

RETURNS:    +1 Failed - T1/ error code
            +2 Success
```

Possible errors: KLPX7

4. read port counters

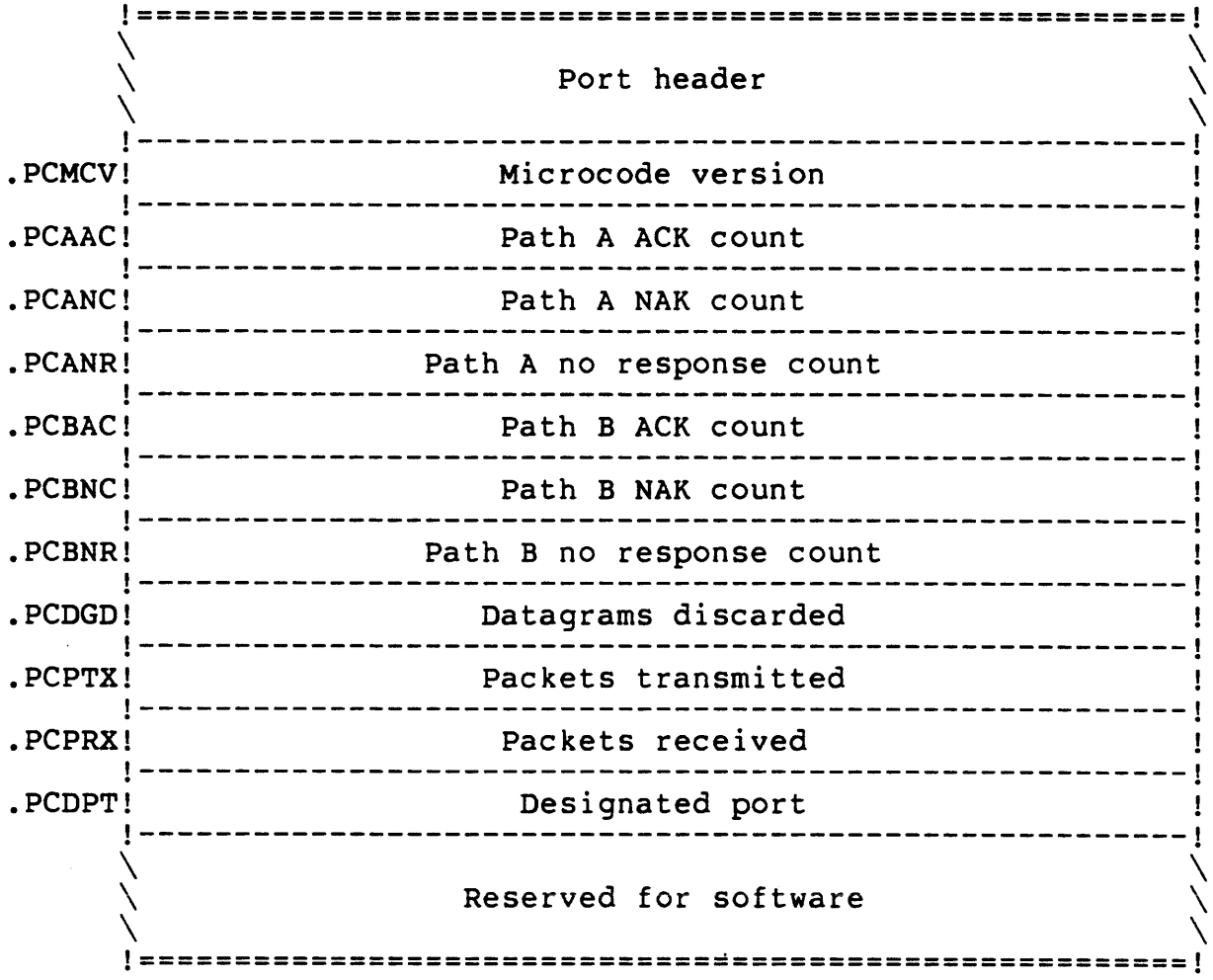
Request the port to return the value of the performance counters. This is an asynchronous operation; when the reply is received, PHYKLP will call SCAMPI.

```
BLCAL. (PPDRPT,<PKA,CID>)

ACCEPTS:    PKA/ Packet Address
            CID/ Connect ID

RETURNS:    +1 Failed. T1/ error code
            +2 Success
```

The following is the format of the counter data:



Possible errors: KLPX7

5. point and/or clear counters

```
BLCAL. (PPDSPT,<PKA,NODE,CTRS>)
CALL PPDSPT
```

ACCEPTS: NODE/ CI Node Number (377 for all)
CTRS/ Counter Bit Mask

B0	If on, count ACKs received on Path A.
B1	If on, clear the counter.
B2	If on, count NAKs received on Path A.
B3	If on, clear the counter.
B4	If on, count NO_RSPs received on Path A.
B5	If on, clear the counter.
B6	If on, count ACKs received on Path B.
B7	If on, clear the counter.
B8	If on, count NAKs received on Path B.
B9	If on, clear the counter.
B10	If on, count NO_RSPs received on Path B.
B11	If on, clear the counter.

B12 The count of discarded datagrams because of no DGFree Queue entries.
B13 If on, clear the counter.
B14 Count the packets transmitted to the designated port.
B15 If on, clear the counter.
B16 Count the packets received from the designated port.
B17 If on, clear the counter.

RETURNS: +1 Failed. T1/ error code
+2 success

Possible errors: KLPX7

7.1.7 Callbacks To SCAMPI - PHYKLP will call SCAMPI when the following events occur:

1. port-to-port virtual cirucit opened (SC.ONL)
2. port-to-port virtual cirucit closed (SC.OFL)
3. SCA packet or application (SYSAP) packet arrived (SC.INT)
4. named buffer transfer completed (SC.DMA)
5. port counter read completed (SC.RPC)
6. the port has detected an error (SC.ERR)
7. maintenance data transfer complete (SC.MDC)

7.2 Services Required Of SCAMPI

SCA is responsible for providing PHYKLP with all buffers to be used for CI message and datagram transmission.

Also, once a port-to-port virtual circuit is closed, SCA must specifically request opening it again. PHYKLP, will not, on its own, reopen a circuit.

8.0 DIAGNOSTIC INTERFACE

The following routines are provided for use by diagnostic functions:

1. start the KLIPA

```
        CALL STRKLP

RETURNS:        +1 Failed.  T1/ Error code
                +2 Success.

Possible errors:  KLPX7
```

2. stop the KLIPA

```
        CALL STPKLP

RETURNS:        +1
```

3. test the CI - send a loopback message to the star coupler

```
        CALL CIONLT

RETURNS:        +1 Failed.  T1/ error code
                +2 Success.

Possible error:  KLPNO, KLPMAN, KLPBUF, KLPENA
```

9.0 ERROR PROCESSING

TOPS-20 finds out about errors detected by the port by seeing certain CSR bits on, examining the error words in the PCB, and finding packets with errors on the response queue.

9.1 CSR Status

The control and status register (CSR) is used by the port and the software to communicate with one another. PHYKLP does a CONI KLP,AC to read the CSR; the complete set of CSR bits is in [1]. There are 5 CSR bits which indicate errors detected by the port:

```
CI.CPE          ;CRAM PARITY ERROR
CI.MBE          ;MBUS ERROR
CI.EPE          ;EBUS PARITY ERROR
CI.FQE          ;FREE QUEUE ERROR
CI.DPE          ;DATA PATH ERROR
```

The port generates an interrupt whenever it sets one of these bits. Why the port sets these bits and what is expected of TOPS-20 is described in [2].

9.2 PCB Error Words

The port places information in the PCB's error words and generates an interrupt whenever it has problems processing a command queue entry. The contents of the error words are defined in [1].

9.3 Response Queue Entries

The port generates an interrupt whenever it places a packet on the response queue and the response queue is empty. When PHYKLP processes a response queue packet it will check the status field to see if the port has set any error bits, indicating there was a problem in packet processing or transmission. These status field bits are defined in [1].

9.4 TOPS-20 Error Reporting

PHYKLP will make ERROR.SYS entries, BUGINF/CHK/HLT, and produce CTY messages as stated in [2].

9.5 Reloading The KLIPA Microcode

Whenever PHYKLP determines that the KLIPA microcode must be reloaded, it arranges for the program SYSTEM:IPALOD.EXE to be run by Job 0. This program is run by Job 0 at the earliest opportunity and a message is sent to the CTY.

IPALOD contains the appropriate KLIPA microcode version and the necessary code to load the microcode. If there is no file called SYSTEM:IPALOD.EXE, the KLIPA microcode cannot be reloaded.

10.0 KEY ALGORITHMS

10.1 KLIPA Initialization

Immediately after the swappable monitor has been loaded, the routine PPDINX is called from job 0; it does the following:

1. create and initialize the PCB
2. call SCA's initialization routine
3. stock the datagram and message free queues
4. enable the KLIPA
5. throw away any packets on the queues
6. give the KLIPA its PIA
7. determine the local port number
8. send REQUEST-IDs to the other 15 nodes
9. set SF%KLP in FACTSW indicating KLIPA initialization has completed

This is done in process context (rather than during PHYSIO start-up) so that any memory management or interrupt processing can happen in a conducive environment.

NOTE

If we decide to support PS: on CI disks, initialization will have to take place during the call to PHYINI and therefore we will have to redesign some of the implementation.

The processing of response queue packets during initialization is done in the asynchronous interrupt routine KLPINT and not synchronously within PPDINX. The one exception to this is the short spin-wait for the response to the READ-REGISTER commmand in which the port indicates our CI node number.

10.2 Once-a-Second Poller

The poller, routine KLPCHK, runs in scheduler context as part of PHYSIO's once-a-second routine. It is responsible for:

1. checking to see that the KLIPA is still running

2. verifying that the KLIPA still knows its PIA
3. locating newly on-line nodes
4. checking the paths of open virtual circuits
5. checking the REQUEST-ID timers
6. checking the start sequence timers

The poller is on when TOPS-20 is booted; SCA does not have to start it.

10.3 KLIPA Interrupt Service

All KLIPA interrupts are processed in the routine KLPINT which PHYSIO's PHYINT routine dispatches to through KLPDSP. A KLIPA interrupt occurs for one of the following reasons:

1. response queue packet available
2. free queue error
3. E-bus or M-bus error
4. CRAM parity error
5. data path error
6. command queue processing error

The normal case is response-queue-available, meaning some data has arrived that needs the attention of the port driver. Some arriving packets are for the port driver itself; thus, PHYKLP processes those and takes the appropriate action. The other packets are for SCA, so PHYKLP simply calls SCAMPI to say there is an incoming packet.

The error cases are descibed in the Error Processing section.

11.0 TESTING

PHYKLP will be tested by 3 major efforts: DVT (design validation testing), the SCSTST program, and the PAGES program.

DVT is conducted by hardware engineering and includes a comprehensive fault insertion effort aimed at validating the operation of the hardware, microcode and software in the face of failures. This should provide adequate assurance that PHYKLP's error recovery procedures are functioning correctly.

SCSTST is a test system developed to test the SCS%, DIAG%, and SCAMPI. However, as SCAMPI is the only user of PHYKLP, an adequate exerciser for SCAMPI is also an adequate exerciser for PHYKLP. SCSTST is capable of testing the major features of PHYKLP and using it in conjunction with fault insertion procedures should provide a comprehensive test of PHYKLP. To this end, there will be a standard set of SCSTST test scripts to serve as regression tests for PHYKLP and the associated hardware and software.

PAGES is a program which does continuous I/O to a file. This will be used as a major tool in verifying that disk I/O works.