```
 III     CCCC    CCCC    SSSS   PPPP     000    RRRR    TTTTT
  I     C       C       S      P   P   0   0   R   R     T
  I     C       C       S      P   P   0   0   R   R     T
  I     C       C        SSS   PPPP    0   0   RRRR      T
  I     C       C           S  P       0   0   R R       T
  I     C       C           S  P       0   0   R  R      T
 III     CCCC    CCCC  SSSS   P        000    R   R     T


M    M  EEEEE  M    M            1       1
MM  MM  E      MM  MM           11      11
M M M  E      M M M M            1       1
M    M  EEEE   M    M            1       1
M    M  E      M    M            1       1
M    M  E      M    M     ..     1       1
M    M  EEEEE  M    M     ..    111     111


*START* Job HSC-DC Req #101 for DEUFEL.TL    Date  9-Apr-82 10:16:56 Monitor: KL210
File PS:<DOC-SPECS>ICCSPORT.MEM.11, created: 16-Jul-79  9:41:33, printed:  9-Apr-82
Job parameters: Request created: 9-Apr-82 10:09:23   Page limit:246   Forms:NORMAL
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:ASC
```

This is a working paper on the KL10 ICCS port design. It is not intended to be a complete functional or design specification but rather a departure point for identifying the areas of concern and technical interest.

The KL10 ICCS port will provide a VAX-compatible interface to the ICCS bus. The port has three important interfaces:

1. Software to port control. This interface is used to request data transmission, post completion, and transmit status. This interface will be implemented with the standard PDP-10 I/O instructions.

2. The port to link interface. This interface is specified by the corporate interconnect specification and is beyond the scope of this specification.

3. The port to memory interface. This interface allows the port to reference and to modify the host memory. This interface has two divisions:

   a. The DMA interface for transferring data to and from the link. This interface will be via the C-BUS.

   b. The E-BUS interface for queue manipulations, and packet fetch and store. This will be implemented via the E-BUS IOP functions.

The principal goal is to implement the VAX port architecture in TOPS20 and in the KL10 port and provide a transparent migration to the 2080 interface.

Since the implementation will be compatible with the VAX port spec, and since that architecture is well-documented [1], this specification will be concerned with differences between the VAX port spec and the KL10 implementation as well with KL10-specific operations.

1.0 Initialization

The monitor will intialize the port operation by indicating the port's PI level and the address of the Port control block(PCB) (see section 8 of [1]). In order to facilitate the port's interface via the E-BUS, the KL10 port control block is as follows (all words are 36-bit words)

word offset	description
0	physical address of BDT
1	Free queue interlock word

2	Free queue forward pointer
3	Free queue back pointer
4	Command queue 0 interlock word
5	Command queue 0 forward pointer
6	Command queue 0 back pointer
7	Command queue 1 interlock word
8	Command queue 1 forward pointer
9	Command queue 1 back pointer
10	Command queue 2 interlock word
11	Command queue 2 forward pointer
12	Command queue 2 back pointer
13	Command queue 3 interlock word
14	Command queue 3 forward pointer
15	Command queue 3 back pointer
16	Response queue interlock word
17	Response queue forward pointer
18	Response queue back pointer

The buffer descriptor table indicated by offset 0 of the PCB is as follows:

word	description
0	count of number of BDT entries
1 to 2*n	n two-word entries (each entry, if valid, points to the buffer descriptor (BD) for this entry (see section 2.1).

The port control block will be identified to the port via a DATAO that specifies the physical address of the block.

In order to implement the interlock function, the port needs the following capabilities:

1. The ability to read a memory location and to simultaneously modify it. None of the existing IOP functions provide this an an atomic operation. It is recommended that the "increment" IOP function (function code 3) be modified to both increment the memory location and to drive the resulting value back onto the E-bus. In this way it becomes a true AOS function.

2. The ability to detect that the interlock is unavailable and to try again later. In order not to monopolize the processor, the retry algorithm should have a reasonable time delay. Once the monitor interface is designed, an estimate of the monitor's interlocked path can be made. If the IOP increment function is modified, then the port can detect if the interlock has been achieved by examining the returned data. A zero indicates successful interlock, anything else indicates a failure.

The manipulation of the queues is straight-forward and in accordance with the algorithms in [1].

1.1 Command queues

The port and the monitor maintain four distinct command queues. Each command queue has its own interlock word and list pointers. Furthermore each command queue has an implicit priority assignment with queue 0 being the highest priority and queue 3 being the lowest.

The priorities are used to control the execution of commands. Commands fall into three general categories:

a. Commands requested by the local host. These are commands placed on a command queue directly by the local operating systems.

b. Data transfers initiated by the local host. This differs from the previous category in that a single command causes a block data transfer with a DMA-like mechanism.

c. Data transfers initiated by a remote host. Again, this is accomplished with a command, but requires additional resources by way of the DMA data transfers mechanism.

The various priority levels may be used to sequence the execution of these types of commands in the most expeditious manner.

2.0 Sending messages and data

The procedure for sending messages and data is described in [1]. The BDT and packet formats in [1] are not appropriate for the needs of TOPS20, however, and need to be modified as follows:

2.1 Buffer descriptor formats

A buffer descriptor (BD) consists of a linked list of descriptors each of which describes a region of physical memory. BDs are described by entries in the buffer descriptor table. Each individual descriptor has the following format:

word	contents
0	physical address of next descriptor
1	mode(3), count(15), offset in buffer (18)
2	physical address of base of data buffer
3	port-available storage and final status

The head of the BD chain (i.e. the BDT itself) is located

in the BDT and is as follows:

word	contents
0	valid bit(1), buffer key (16), modifier(2),port
1	physical address of head of BD chain

Each BDT identifer given out by TOPS20 will be the offset from the start of the BDT table in the PDB. Therefore the address of the BDT header is:

BDTB+2*(n-1)

where

BDTB is the base address of the start of the BDTs and n is the descriptor specified in the data request. For n to be valid, it must be less than the number of BDT entries specified in the PCB.

Note that the BD has been designed so that all of the verification information is in one word. Since the port must verify that the requesting port has the right to use the selected buffer, this procedure may be performed by fetching only one word from PDP10 memory.

2.2 Messages

A message, as defined in [1], is a fixed-sized packet of data that is transmitted atomically over the bus. Messages have the peculiar property of being generated either by the host software (in this case, TOPS20), by the port itself (to generate a request for data), or by the remote port (to signify completion, or to request data). Therefore, there cannot be a KL10-specific format for messages that conflicts with messages generated and expected by other ports on the bus.

Given this restriction, it seems logical to define a message as being coincident with that defined [1]. This implies that messages will contain "byte" data but may identify BDs that address word data.

To simplify the handling of these message packets, the first three words of each packet will be interpreted as containing 36-bit data, and the message body as containing byte data. That is, the link pointers and internal flags and control information will use all 36-bits of each word. The message body will use the left-most 32 bits of each word (Since the contents of the first three words are not transmitted over the bus, this presents no conflict with the interface requeirements).

3.0 C-BUS interface

When the port needs to have DMA access to memory, it will use the C-BUS to reference its data. That is, the port will appear to be an RH20 and consequently needs to manipulate the appropriate EPT locations to control the internal channel it is "borrowing". This access is via the IOP functions of the E-BUS.

4.0 IO instructions

The following IO instructions are defined for the port:

CONO PRT,

bits	definition
33-35 (RW)	PSI level
32 (W)	enable PSI
31 (RW)	GO
30 (RW)	Stall
29 (R)	memory parity error detected
28 (R)	unrecoverable bus error
27 (R)	Free queue empty (message ovverrun)
(W)	Clear errors
26 (W)	Command queue loaded
25 (RW)	Response queue non-empty
24	Clear cache

GO bit: This bit is on whenever the port is active processing a request. If it is set on a CONO, it causes the port to poll the request queues for some work to do.

Stall: This bit is used to shutdown the port. If Stall is set, the port will complete its present operation and then clear GO and enter the idle state. Any new requests that arrive should be rejected. Setting GO clears Stall.

Response queue non-empty: This is asserted by the port when it inserts an entry on an empty response queue.

Command queue loaded: The software will assert this bit whenever it places an entry on an empty command queue. When this is asserted, the port should note that the command queue needs to be polled. The monitor may not assert this bit when it places an entry on a non-empty command queue (Since the port can place a message on the command queue itself, it should not rely on this bit being asserted to begin command queue processing. In this case, however, the port could assert the bit itself.).

Clear cache: Since the port needs to make frequent reference to memory, it is likely it will implement some form of memory cache to avoid the overhead of typical memory references. The monitor will assert this bit whenever it

wishes to invalidate a buffer descriptor and the port should respond by flushing from its local memory any knowledge of extant buffer descriptors. Since the monitor never unilaterally invalidates the queues without resetting the port, any cached queue information may be retained.

The error bits may need expanding with time.

NOTE: The "reset" state of the port should be that Stall is set and GO is off. This state is entered whenever an IO reset is issued or a DATAO PRT, is issued. This feature precludes the port's clobbering memory while BDTs are being established and allows the software to determine if a port exists without changing its state.

4.1 CSRs

The port CSR registers are read and written via DATAI and DATAO instructions. Each CSR has a distinct function and is selected by a DATAO to "select CSR register".

4.1.1 Load PCS base address and PCS CSR

CSR 00 (RW) PCS base address

This function is used to reset the port and declare the port descriptor block. The argument is the physical address of the descriptor block. If this function is issued while the port is active, the port should immediately shutdown and enter the "reset" state.

4.1.2 CSR 01 (RW) Port configuration register

This CSR contains:

port i.d. (R)
number of busses present (R)
busses enabled (RW)
bus selection mode (RW)

The bus selection mode indicates whether the busses are selected explicitly by the software (high performance mode) or are considered as one bus and may be used as the port desires (high availability mode). The active busses and the bus selection mode may be modified by the software. The "reset" state of the bus is to enable all connected bus and to employ the high availability mode for bus selection.

4.1.3 CSRs 02-10 (RW) port performance meters

TBS

4.1.4 CSR 11 (R) port error register

This CSR contains detailed information on any port, link or bus detected errors.

4.1.4 Other CSRs

All unspecified CSR registers are reserved to DEC.

4.2 Issues

An important issue is the manner in which diagnostics will access the port. We may want to include a rich collection of status and condition flags so that it is possible to exercise the port without having to use the speicifed queue structures.

4.3 Caching of port data

As mentioned earlier, the port may want to provide a cache for its data to avoid the overhead of constantly referencing main memory. Such a cache can contain BD, BDT and queue entry data that the port is likely to need.

It seems quite reasonable to cache BD and BDT data since these are the most performance sensitive as well as frequently needed data items. Buffer descriptors, once set up by the software, will not change as long as the buffer descriptor remains valid.

Caching queue entries is risky and possibly not justifiable considering the care that must be exercised. However, if it is possible to cache any portion of the queue data base, the performance gain is certainly welcomed.

Some careful thought should be given to this feature since its effectiveness will likley have a significant effect on the efficiency and performance of the port.

5.0 Data modes and modifiers

The data modes supported by the KL10 port are:

0 — byte mode. Data is packed as eight-bit bytes occupying the left-most 32-bits of each memory word

1 — word mode. Data is packed using all 36-bits of each memory word. When word data is transmitted or received over the bus, it is packaged as a series of 8-bit bytes. If the data sent is not an integral number of bytes (i.e., the word count is odd), then the last byte is zero-filled. If the data received is not an

integral number of words, any extra bits are discarded.

2 Tape mode. This mode requires that each word be transmitted or received as five bytes of data. The low-order four bits of the last byte (the first four received or transmitted) are zero-filled.

In each BDT is a modifier field. The defined values are:

1 Read-only

2 Read backwards

3 write-only

4 Write backwards

The modifier indicates the type of operation that may be requested on this buffer and the direction the data is to be accessed.

6.0 Programming considerations

The information up to now has been aimed at understanding how the KL10 ICCS port will operate. Several important points must be noted:

1. The addresses that the port uses are all physical memory addresses. This means that the queue pointers and headers must contain physical addresses. This has been done to make the port's memory references be independent of the M-BOX, pager refills, and various types of paging failures that may occur. The KL10 port could be built to deal with EXEC virtual addresses, but the 2080 appears not to have that flexibility.

2. The construction of BDTs and BDs is complicated. Processing and verification of these tables will be costly for both the port and the monitor.

These two points indicate that a unique apporach to creating the monitor's port driver must be considered. In particular, it is important that the monitor be able to enqueue and dequeue command and response messages from the appropriate queues quickly. Several techniques are possible:

1. Ensure that the PCB is in a page that is mapped virtual to physical. That is, if the PCB is in EXEC virtual page 100, then it must also be in physical page 100. It is also necessary to allocate BDs from

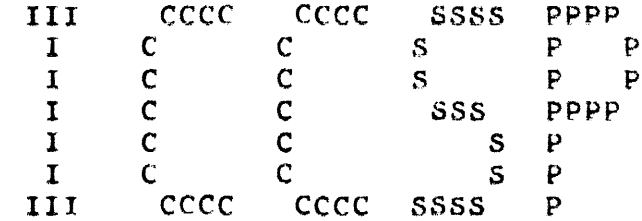
pages with the same property. This restriction will not apply to the 2080 as it will provide instructions that implement physical addressing.

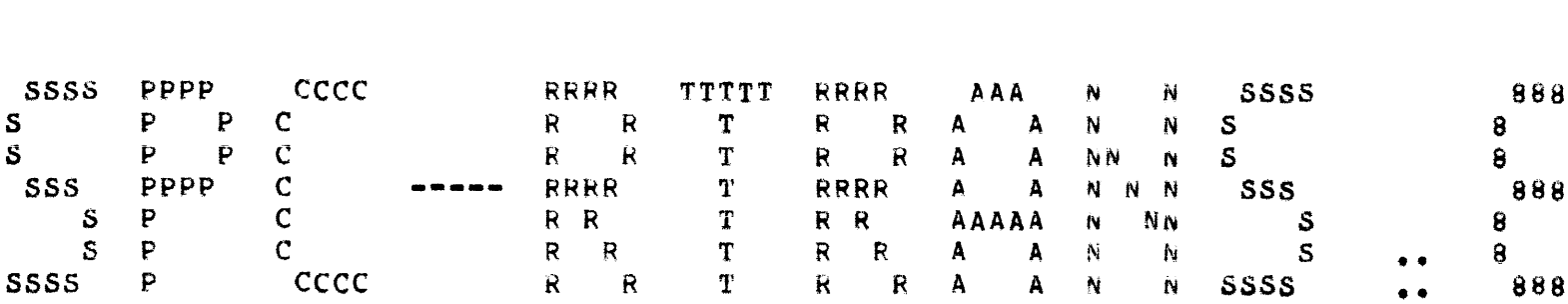
2. Since the monitor knows the virtual address of the PCB, it can always reference it quickly. However, since it cannot predict which packet it needs to dequeue next, it needs a quick means of translating the physical address to a virtual one. The most straight-forward technique is for the port driver to have a virtual address slot it may use for temporary mapping (the same kind of slot used in PAGEM e.g. CXBPG). When it needs to examine a packet, it would map the physical page to its page slot, perform any data modifications or extractions, and clear the map slot. Since all queue headers and interlock words are in fixed locations they can always be referenced with virtual addresses.

Despite the fact that the monitor always processes queues FIFO, it still needs to maintain both forward and backward queue pointers since the port references queues randomly. Therfore, enqueuing or dequeueing a packet requires modifying another packet as well as a queue header. To make this as efficient as possible, two temporary map slots could be allocated.

3. Building in some other innate knowledge about BDs that would enable rapid conversion of physical addresses into virtual ones. An example would be to allocate all BDs from a single exec virtual page thereby making the word offset in the physical page the unique identifer of the packet's location. Multiple pages could be used with another level of translation required.

The first technique seems adequate for the KL10. The 2080 provides sufficient flexibility in its addressing modes that the problem disappears.





START Job HSC-DC Req #101 for DEUFEL.TL Date 9-Apr-82 10:16:56 Monitor: KL210
File PS:<DOC-SPECS>ICCSPORT.SPC-RTRANS.8, created: 16-Jul-79 9:41:06, printed: 9-
Job parameters: Request created: 9-Apr-82 10:09:23 Page limit:246 Forms:NORMAL
File parameters: Copy: 1 of 1 Spacing:SINGLE File format:ASCII Print mode:ASC

This is a working paper on the KL10 ICCS port design. It is not intended to be a complete functional or design specification but rather a departure point for identifying the areas of concern and technical interest.

The KL10 ICCS port will provide a VAX-compatible interface to the ICCS bus. The port has three important interfaces:

1. Software to port control. This interface is used to request data transmission, post completion, and transmit status. This interface will be implemented with the standard PDP-10 I/O instructions.

2. The port to link interface. This interface is specified by the corporate interconnect specification and is beyond the scope of this specification.

3. The port to memory interface. This interface allows the port to reference and to modify the host memory. This interface has two divisions:

    a. The DMA interface for transferring data to and from the link. This interface will be via the C-BUS.

    b. The E-BUS interface for queue manipulations, and packet fetch and store. This will be implemented via the E-BUS IOP functions.

The principal goal is to implement the VAX port architecture in TOPS20 and in the KL10 port and provide a transparent migration to the 2080 interface.

Since the implementation will be compatible with the VAX port spec, and since that architecture is well-documented [1], this specification will be concerned with differences between the VAX port spec and the KL10 implementation as well with KL10-specific operations.

1.0 Initialization

The monitor will intialize the port operation by indicating the port's PI level and the address of the Port control block(PCB) (see section 8 of [1]). In order to facilitate the port's interface via the E-BUS, the KL10 port control block is as follows (all words are 36-bit words)

.literal

word offset	description
0	physical address of BDT
1	Free queue interlock word
2	Free queue forward pointer
3	Free queue back pointer
4	Command queue 0 interlock word
5	Command queue 0 forward pointer

6	Command queue 0 back pointer
7	Command queue 1 interlock word
8	Command queue 1 forward pointer
9	Command queue 1 back pointer
10	Command queue 2 interlock word
11	Command queue 2 forward pointer
12	Command queue 2 back pointer
13	Command queue 3 interlock word
14	Command queue 3 forward pointer
15	Command queue 3 back pointer
16	Response queue interlock word
17	Response queue forward pointer
18	Response queue back pointer

.end literal

The buffer descriptor table indicated by offset 0 of the PCB is as follows:

.literal

word	description
0	count of number of BDT entries
1 to 2*n	n two-word entries (each entry, if valid, points to the buffer descriptor (BD) for this entry (see section 2.1).

.end literal

The port control block will be identified to the port via a DATAO that specifies the physical address of the block.

In order to implement the interlock function, the port needs the following capabilities:

1. The ability to read a memory location and to simultaneously modify it. None of the existing IOP functions provide this an an atomic operation. It is recommended that the "increment" IOP function (function code 3) be modified to both increment the memory location and to drive the resulting value back onto the E-bus. In this way it becomes a true AOS function.

2. The ability to detect that the interlock is unavailable and to try again later. In order not to monopolize the processor, the retry algorithm should have a reasonable time delay. Once the monitor interface is designed, an estimate of the monitor's interlocked path can be made. If the IOP increment function is modified, then the port can detect if the interlock has been achieved by examining the returned data. A zero indicates successful interlock, anything else indicates a failure.

The manipulation of the queues is straight-forward and in accordance with the algorithms in [1].

1.1 Command queues

The port and the monitor maintain four distinct command queues. Each command queue has its own interlock word and list pointers. Furthermore each command queue has an implicit priority assignment with queue 0 being the highest priority and queue 3 being the lowest.

The priorities are used to control the execution of commands. Commands fall into three general categories:

a. Commands requested by the local host. These are commands placed on a command queue directly by the local operating systems.

b. Data transfers initiated by the local host. This differs from the previous category in that a single command causes a block data transfer with a DMA-like mechanism.

c. Data transfers initiated by a remote host. Again, this is accomplished with a command, but requires additional resources by way of the DMA data transfers mechanism.

The various priority levels may be used to sequence the execution of these types of commands in the most expeditious manner.

2.0 Sending messages and data

The procedure for sending messages and data is described in [1]. The BDT and packet formats in [1] are not appropriate for the needs of TOPS20, however, and need to be modified as follows:

2.1 Buffer descriptor formats

A buffer descriptor (BD) consists of a linked list of descriptors each of which describes a region of physical memory. BDs are described by entries in the buffer descriptor table. Each individual descriptor has the following format:

.literal

word	contents
0	physical address of next descriptor
1	mode(3), count(15), offset in buffer (18)
2	physical address of base of data buffer
3	port-available storage and final status

.end literal

The head of the BD chain (i.e. the BDT itself) is located in the BDT and is as foll

.literal

word	contents
0	valid bit(1), buffer key (16), modifier(2),port
1	physical address of head of BD chain

.end literal
Each BDT identifer given out by TOPS20 will be the
offset from the start of the BDT table in the PDB.
Therefore the address of the BDT header is:

BDTB+2*(n-1)

where

BDTB is the base address of the start of the BDTs and
n is the descriptor specified in the data request.
For n to be valid, it must be less than the number
of BDT entries specified in the PCB.

Note that the BD has been designed so that all of the verification
information is in one word. Since the port must verify that
the requesting port has the right to use the selected
buffer, this procedure may be performed by fetching only
one word from PDP10 memory.

2.2 Messages

A message, as defined in [1], is a fixed-sized packet
of data that is transmitted atomically over the bus.
Messages have the peculiar property of being generated either
by the host software (in this case, TOPS20), by the port
itself (to generate a request for data), or by the remote
port (to signify completion, or to request data).
Therefore, there cannot be a KL10-specific format for messages
that conflicts with messages generated and expected
by other ports on the bus.

Given this restriction, it seems logical to define a message
as being coincident with that defined [1]. This implies
that messages will contain "byte" data but may identify
BDs that address word data.

To simplify the handling of these message packets, the first
three words of each packet will be interpreted as containing
36-bit data, and the message body as containing
byte data. That is, the link pointers and internal
flags and control information will use all 36-bits of
each word. The message body will use the left-most
32 bits of each word (Since the contents of the
first three words are not transmitted over the bus,
this presents no conflict with the interface requeirements).

3.0 C-BUS interface

When the port needs to have DMA access to memory,
it will use the C-BUS to reference its data. That
is, the port will appear to be an RH20 and consequently
needs to manipulate the appropriate EPT locations
to control the internal channel it is "borrowing".
This access is via the IOP functions of the E-BUS.

4.0 IO instructions

The following IO instructions are defined for the port:

CONO PRT,

bits	definition
33-35 (RW)	PSI level
32 (W)	enable PSI
31 (RW)	GO
30 (RW)	Stall
29 (R)	memory parity error detected
28 (R)	unrecoverable bus error
27 (R)	Free queue empty (message ovverrun)
(W)	Clear errors
26 (W)	Command queue loaded
25 (RW)	Response queue non-empty
24	Clear cache

GO bit: This bit is on whenever the port is active processing a request. If it is set on a CONO, it causes the port to poll the request queues for some work to do.

Stall: This bit is used to shutdown the port. If Stall is set, the port will complete its present operation and then clear GO and enter the idle state. Any new requests that arrive should be rejected. Setting GO clears Stall.

Response queue non-empty: This is asserted by the port when it inserts an entry on an empty response queue.

Command queue loaded: The software will assert this bit whenever it places an entry on an empty command queue. When this is asserted, the port should note that the command queue needs to be polled. The monitor may not assert this bit when it places an entry on a non-empty command queue (Since the port can place a message on the command queue itself, it should not rely on this bit being asserted to begin command queue processing. In this case, however, the port could assert the bit itself.).

Clear cache: Since the port needs to make frequent reference to memory, it is likely it will implement some form of memory cache to avoid the overhead of typical memory references. The monitor will assert this bit whenever it wishes to invalidate a buffer descriptor and the port should respond by flushing from its local memory any knowledge of extant buffer descriptors. Since the monitor never unilaterally invalidates the queues without resetting the port, any cached queue information may be retained.

The error bits may need expanding with time.

NOTE: The "reset" state of the port should be that Stall is set and GO is off. This state is entered whenever an IO reset is issued or a DATAO PRT, is issued. This feature precludes

the port's clobbering memory while BDTs are being established and allows the software to determine if a port exists without changing its state.

4.1 CSRs

The port CSR registers are read and written via DATAI and DATAO instructions. Each CSR has a distinct function and is selected by a DATAO to "select CSR register".

4.1.1 Load PCS base address and PCS CSR

CSR 00 (RW) PCS base address

This function is used to reset the port and declare the port descriptor block. The argument is the physical address of the descriptor block. If this function is issued while the port is active, the port should immediately shutdown and enter the "reset" state.

4.1.2 CSR 01 (RW) Port configuration register

This CSR contains:

port i.d. (R)
.break
number of busses present (R)
.break
busses enabled (RW)
.break
bus selection mode (RW)

The bus selection mode indicates whether the busses are selected explicitly by the software (high performance mode) or are considered as one bus and may be used as the port desires (high availability mode). The active busses and the bus selection mode may be modified by the software. The "reset" state of the bus is to enable all connected bus and to employ the high availability mode for bus selection.

4.1.3 CSRs 02-10 (RW) port performance meters

TBS

4.1.4 CSR 11 (R) port error register

This CSR contains detailed information on any port, link or bus detected errors.

4.1.4 Other CSRs

All unspecified CSR registers are reserved to DEC.

4.2 Issues

An important issue is the manner in which diagnostics will access the port. We may want to include a rich collection of status and condition flags so that it is possible to exercise the port without having to use the speicifed

queue structures.

4.3 Caching of port data

As mentioned earlier, the port may want to provide a cache for its data to avoid the overhead of constantly referencing main memory. Such a cache can contain BD, BDT and queue entry data that the port is likely to need.

It seems quite reasonable to cache BD and BDT data since these are the most performance sensitive as well as frequently needed data items. Buffer descriptors, once set up by the software, will not change as long as the buffer descriptor remains valid.

Caching queue entries is risky and possibly not justifiable considering the care that must be exercised. However, if it is possible to cache any portion of the queue data base, the performance gain is certainly welcomed.

Some careful thought should be given to this feature since its effectiveness will likley have a significant effect on the efficiency and performance of the port.

5.0 Data modes and modifiers

The data modes supported by the KL10 port are:

0	byte mode. Data is packed as eight-bit bytes occupying the left-most 32-bits of each memory word
1	word mode. Data is packed using all 36-bits of each memory word. When word data is transmitted or received over the bus, it is packaged as a series of 8-bit bytes. If the data sent is not an integral number of bytes (i.e., the word count is odd), then the last byte is zero-filled. If the data received is not an integral number of words, any extra bits are discarded.
2	Tape mode. This mode requires that each word be transmitted or received as five bytes of data. The low-order four bits of the last byte (the first four received or transmitted) are zero-filled.

In each BDT is a modifier field. The defined values are:

1	Read-only
2	Read backwards
3	Write-only
4	Write backwards

The modifier indicates the type of operation that may be requested on this buffer and the direction the data is to be accessed.

6.0 Programming considerations

The information up to now has been aimed at understanding how the KL10 ICCS port will operate. Several important points must be noted:

1. The addresses that the port uses are all physical memory addresses. This means that the queue pointers and headers must contain physical addresses. This has been done to make the port's memory references be independent of the M-BOX, pager refills, and various types of paging failures that may occur. The KL10 port could be built to deal with EXEC virtual addresses, but the 2080 appears not to have that flexibility.

2. The construction of BDTs and BDs is complicated. Processing and verification of these tables will be costly for both the port and the monitor.

These two points indicate that a unique apporach to creating the monitor's port driver must be considered. In particular, it is important that the monitor be able to enqueue and dequeue command and response messages from the appropriate queues quickly. Several techniques are possible:

1. Ensure that the PCB is in a page that is mapped virtual to physical. That is, if the PCB is in EXEC virtual page 100, then it must also be in physical page 100. It is also necessary to allocate BDs from pages with the same property. This restriction will not apply to the 2080 as it will provide instructions that implement physical addressing.

2. Since the monitor knows the virtual address of the PCB, it can always reference it quickly. However, since it cannot predict which packet it needs to dequeue next, it needs a quick means of translating the physical address to a virtual one. The most straight-forward technique is for the port driver to have a virtual address slot it may use for temporary mapping (the same kind of slot used in PAGEM e.g. CXBPG). When it needs to examine a packet, it would map the physical page to its page slot, perform any data modifications or extractions, and clear the map slot. Since all queue headers and interlock words are in fixed locations they can always be referenced with virtual addresses.

Despite the fact that the monitor always processes queues FIFO, it still needs to maintain both forward and backward queue pointers since the

port references queues randomly. Therfore, enqueuing or dequeueing a packet requires modifying another packet as well as a queue header. To make this as efficient as possible, two temporary map slots could be allocated.

3. Building in some other innate knowledge about BDs that would enable rapid conversion of physical addresses into virtual ones. An example would be to allocate all BDs from a single exec virtual page thereby making the word offset in the physical page the unique identifer of the packet's location. Multiple pages could be used with another level of translation required.

The first technique seems adequate for the KL10. The 2080 provides sufficient flexibility in its addressing modes that the problem disappears.