bcc	title SOFTWARE FOR THE PHASE 2 COMMUNICATIONS SYSTEM	prefix/class-number.revision P2CSS/W-49
checked Butler W. Lampson	authors Paul Heckel	approval date 9/25/70 revision date
checked Judith Ford		classification Working Paper
approved Mel		distribution company private pages 49

ABSTRACT and CONTENTS

This document describes the software that is needed to make the Phase II communications system run.

It also provides a wealth of information on the table formats, CHIO variables and other such trivia.

CONTENTS

SOFTWARE FOR THE PHASE 2 COMMUNICATIONS SYSTEM

1. Introduction

This document describes the Phase 2 Communications System Software. Whereas most software assumes that the hardware was handed down by God, this software assumes that the firmware was similarly created and is similarly unchangeable. This software consists of six different subspecies:

1. Monitor initialization code (run at system startup)
2. The system configurator
3. MCALLs (monitor code to modify tables)
4. The Communications System Error Process
5. The Listener
6. RPU code in the DCC

Insofar as they affect the communication system, the functions of each of these will be discussed in detail.

It should be understood that this implementation of the Phase 2 software does not use all of the features of the communications system. The point is to have a working system; sophistication will come with more experience and understanding. For this reason, all of the lines attached to a particular DCC must be connected to the same group (although the Phase 3 system will allow several groups to share the same DCC).

II. Monitor Initialization Code

When a system restart is performed, the Monitor should set up some communications system tables and leave space for others.

In addition to setting up these tables, it should turn off all high speed devices (EFCLs).

Most of this section is devoted to specifying the locations, tables, scratchpad registers, etc. that should be initialized or contain interesting values. All such locations should be initialized to zero unless another value is explicitly stated.

Following each of the tables or variables that has to be set up is the name of the program that is expected to do it, either the Monitor during initialization (MI) the System Configurator (SC), or the Listener (L).

Because several tables must begin on locations having peculiar properties we adopt the notion of a constrained table (table entry, or buffer). If a table is constrained that means that if it is of size N, then it must begin on a boundary divisible by M where M is the smallest number greater or equal to N that is a power of 2. Thus a 33 word constrained table would have to begin on a location divisible by 64. Usually, however, the table or table entry size is divisible by a power of 2. Most tables or table entries are constrained. The notation (C) says the table is constrained, (C=N) says that it is constrained to be of

Size N, and thus must fall on a boundary divisible by M. (defined above).

Public Memory Locations

All memory locations and scratchpads must be set up by the Monitor as the CHIO has no internal initialization function.

The following Public Memory core locations should be initialized for the CHIO to function properly (MI):

WPL(15B): Wakeup pointer

SRMEM(5B): Restart Switches

RTLBASE(13B),RTHBASE(12B): Adjustment for real time clock

The following public memory locations may be modified by the CHIO: WPL(15B), RTLOW(11B), RTHIGH(10B) and the CPUIT(40B-57B).

The following locations must be initialized by the CPU (MI).

$(BUFØ+N)←Ø for N=Ø to 7 BUFØ is a scratchpad.

SYNWORD(66B): Should have 3 SYN characters in it.

XIB(6ØB),XOB(61B): Number of extra input (output) buffers used (should be zero).

JOEQ(67B): Queue of JOE buffers

JOEFL(70B): Free list of JOE buffers

BERTHOLD(62B): Should be zero

BRECHT(63B): Number of process to awaken if buffer count goes out of range. (NFB > MAXFB or NFB < MINFB)

NFB(75B): Should be set to the number of free buffers.

The following locations should be initialized to zero. They are incremented whenever something interesting happens (MI).

BADWK(71B): An attempt to awaken a process found the wakeup buffer full.

NULCPU(73B): Null CPU request count.

EFRECNTR(64B): Count of input rate errors on EFCL.

OUTCC(72B): Number of characters processed by WST.

INCC(74B): Number of characters processed by RSTB.

INLCNT(65B): Incremented if nothing to do.

Scratchpad Registers

The following scratchpad registers must be set up by the CPU (MI):

BUFØ (7B): Origin of free storage. First 8 words following BUFØ should be Ø. (C=8) Size=8* (number of free buffers) +8.

MTAA (1B): Origin of Main line table. Size=(no line pairs) *4 (C=4)

MTCA (2B): Origin of character pointer line table. Size=(no line pairs)*6, =(no of lines)*3

DVTBA (6B): Points to device table. (C=16).

MOLLY (53B): Points to the EFCL/medium speed table. (C=2ØØB).

EBTAB (13B): Points to 32 word echo-break-strategy table. (C=32).

FREEL (51B): Pointer to first buffer in free list (taken relative to BUFØ) (C=8).

MP (55B): Multiplier for buffer counts. (Ø-2B7) in 2B5 increments. Maximum buffer count of a line is defined as XBCNT (line) * MP/2B7. MP's normal value is 2B7.

MAXFB (17B) and MINFB (34B): If the free buffer count (NFB, a core location) is less than MINFB or greater than MAXFB a special process (BRECHT) is awakened.

XLINE (4B): Largest legal line number. If the line number in a CPU request is greater than

the value of XLINE the CHIO aborts.

AVB (52B), PRB (54B): Should be set to Ø.

BFLAG (16B): Should be set to 4B7.

CPUFLAG (74B): Setting CPUFLAG to -1 is equivalent to doing an IG (Ignore non-CPU requests). Setting it to Ø is equivalent to 'NIG'.

NORPRT (56B): Normal priority for memory request (try Ø).

HPRT (57B): Priority for high priority memory requests (try 6). Should be higher than the CPU priority.

NLSPD (12B): Should be Ø

GPRMS (35B): Should be 5ØØD. (Number of milliseconds in 1/2 second.

LCWS1 (20B): Should be Ø.

In-Core Tables

The following In-core tables must also be set up. The section called Other CHIO Tables beginning on page 14 describes what each of the fields does.

CPU Line Table: Before a line is used the CPU line table and the character pointer table entries should be initialized. The CPU line table entry should be zero except that the following fields should be set up:

BS, WS, ES, RCND, EBC, DTYP, QUIT, ESCAPE, XWT, LDVN, PROC.

A free line has the process number = -1. (L) (C=4)

Character Pointer Line Table: XBCNT and WKCNT should be initialized. Other quantities should be zero. (L). For efficiency's sake it is desirable that the origin of the character pointer line table be on an odd location.

Character Buffers: Should be linked (relative to BUFØ) through the field NEXT. The last buffer should contain a zero in NEXT (MI) (C=8). (NFB should be initialized to the number of free buffers).

The CPU Interface Table: RWCH should be set to zero.

Device Table: CØ through C255 should be set up. (SC) (C=N * 32). N=no device types.

MOLLY is a 4ØØB word constrained table whose first 200B words are indexed by the low order 7 bits of the Multiplexor Line number. The sign bit of the indexed word, if on, indicates that this is a medium speed line, and the rest of the word

is the CPU line number for the Medium Speed line. If the sign bit is off, then the rest of the word points to the origin of the EFCL table associated with the EFCL line. Presumably all inactive lines are connected as medium speed lines to some CPU process which aborts the system. The last 200B words of MOLLY contain the timeout counters used for the EFCLS. If MOLLY +n points to an EFCL, MOLLY + 200B +n points to the timeout counter for it.(SC), (C=4ØØB).

EBTAB is a 128 word table that is indexed by a 7 bit quantity made up of (in order) not EBC, not BS, not WS and not of the character type. If the sign bit of the entry is on, the character should be echoed. The low order two bits specify the wakeup type: (MI) (C=128).

Ø: Not a wakeup character

1: Wakeup character, but not a break character

2: Both break and wakeup character.

Communication Line Tables

The most important table used for a communications line is the EFCL table. This constrained 64 word table is the 'Context Block' for the communications line. It is divided into 4 parts: EFCL input, EFCL output, Input multiplexing, and Output Multiplexing. In addition, there are 4 other tables which are pointed to from the EFCL: LARRY which buffers the output characters; JOE which buffers the input characters; NCITAB which specifies the device timing; and LDT, the logical device table. The only set of these tables which is not unique to a DCC is JOE, which is obtained from a free list of JOES. The entries that need to be set to a non-zero value by the Monitor are marked with an asterisk (*). We now discuss each of these tables or subtables in detail.

The Output EFCL Subtable (SC) consists of:

*LARRY: Pointer to origin of LARRY, the output buffer.

EBO: Pointer to current output buffer Base (usually in LARRY).

NCHO: Character pointer to current character in output buffer. This is merged with EBO to get a pointer to the current character. If EBO is -1 the buffer is empty.

AO: The output block number. This number is used to identify the block to be retransmitted if an error occurs.

*NTCHO: (The total number of characters (data plus checksum) in the output block plus 1). Right cycled 1

*LNBU: Top two bits contain LOG_2 (LBN)-2 where LBN is the length of a sub-buffer in LARRY. Bottom bits contain LBN-1.

*CA: The Governor counter. If CA is less than or equal to Ø transmission will not occur.

RELAY: Not used in the CHIO, only in the DCC.

OTCO: Set by microcode to indicate next block goes to the triangle partner.

MISTRESS: Pointer to EFCL base of a triangle partner.

MSGCNTR: Counts messages transmitted.

GOVCTR: Counts number of times Governor is invoked.

TOTCTR: Counts the number of times the Timeout counter was invoked.

*NCSCHO: Number of checksum words -1 merged with the sign bit.

STATE: State of EFCL line.

TIMADDR: Address of timout timer in Molly.

The message and acknowledgement buffers (location) should be initialized by the CHIO, thus location 24D to 31D should be set up as follows:

24: 0

25: 0

26: 26B (SYN)

27: 0

28: 0

29: 100002B (message for block zero.)

30: 100002B (copy of above for checksumming)

31: (26B) (SYN)

The Input EFCL Table consists of (SC):

AI: Input block count.

OTCI: Other computer switch used for input.

NCHI: Pointer to next input character.

OBUFQ: Pointer to a list of JOES for the line during triangle mode.

*NTCHI: Total number of characters (data plus checksum) in a block. Right cycled 1.

*NCSCHI: Number of checksum words -2 (normally -1)

*LBU: Length of a buffer unit negated (see description of LARRY).

*VTIME: The valpariso round trip time used for timeout. This is a number of 10 microsecond intervals that it should take to get a reply from a message. This includes the delay time due to buffering in the buffers, modems and telephone lines. (See also description of LARRY).

*EBI: Should point to a free JOE.

The Output Multiplexing variables are (SC):

NCHSR: Number of characters to send or receive for the current device.

NCLII: Number of characters left in the interval.

GHSDEV: The number of the selected high speed device.

*INCLII: The initial value of the number of data characters in an interval (1/10 of a second). For a 4800 baud line with 13 data characters and 2 check characters per block, the number is 56.

*GSELDEV: The selected device. Initialized to origin of LDT (like GLDTORG).

*PASS: The current pass. Should be set to 4B7.

*GLDTORG: Pointer to the beginning of the LDT.

*NCIPTR: Pointer to NCITAB divided by 16

*NCIWP: Pointer to NCITAB.

MATST: Used by the output multiplexor

The input Multiplexor Variables (SC) are:

IMRST: Input multiplexor state

IMRDVN: Current Device number

*CMLICP: Pointer to LDT table.

LARRY is a constrained buffer that has N constrained sub-buffers each of length LBU. Both N and LBU must be powers of two. LBU must be at least one character larger than the number of data characters plus then number of checksum characters in a block. N must be picked so that N-1 sub-buffers can hold at least enough data and checksum characters to sen in VTIME.

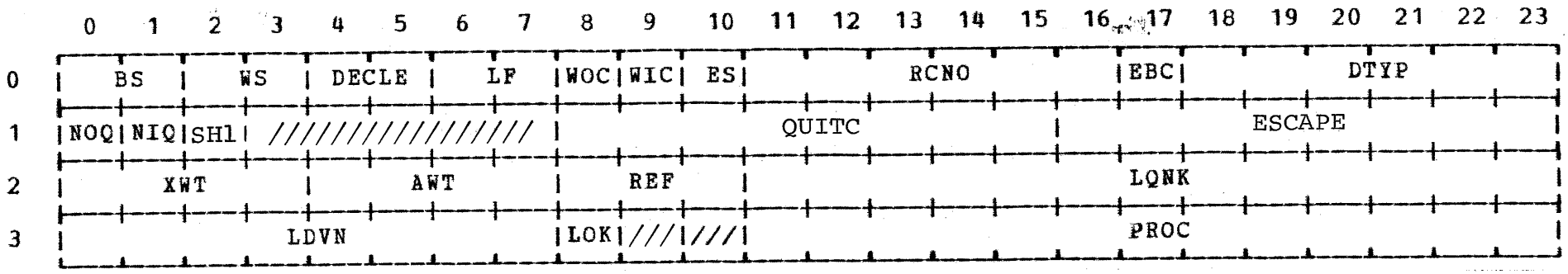
JOE buffers come in two parts, the header and the constrained buffer proper. The buffer header is in the two words previous to the buffer proper. The second of these two words is the linkage pointer. The end pointer is a zero. Two JOE buffers should be provided per EFCL.

The NCITAB (SC) is a constrained table (C=4*N) that has N+1, 4 word entries, where N is the number of device types. The last entry should be 4 words of zeros. Each of the other entries is indexed by the device type (times four). The first word in each table entry should contain 256 times the

number of characters the device can type out in 1/10 of a second (this number may be fractional).

The logical device table, LDT, is a constrained table that has a one word entry per logical device. Indexed by the local device number, it contains the CPU line number in the bottom bits, and the active bits used by GNL in the top few bits. The last entry is a dummy, and should be set to -1.

Public Memory for Input/Output Line - 4 words (MTAA,MTAE,/4)



AWT: Actual wait time left (2:4,7)
AWT is set to XWT when an input wakeup is generated.

BS: Break strategy (stop echo strategy) (0:0,1)
0) Never
1) Control characters only
2) Punctuation characters and control characters
3) Always

CCP:* [CCPBIT] if on when line is empty wakeup process in link field (0:6,6)

CLE:* Character locally echoed (for local lines) (0:5,5)

DE:* Deferred echo. If on, echo characters when input to CPU (0:4,4)

DECLE: Both DE and CLE (0:4,5)
Normally DE and CLE are 0. When a break char is input to CHIO CLE is set. When a break char is input to the CPU DE is set. When an attempt to read chars to the CPU is made that fails both are reset.

DTYP: Device type for local lines (dispatched on) (0:18,23)

EBC: Echo break character (0:17,17)

ES: Echo strategy. If off, don't echo characters (0:10,10)

ESCAPE: Escape character (1:16,23)

LDVN: Local device number (3:0,7)

LF: Link flag (0:6,7)
0) NRMLNK: ignore LQNK field
1) LNKLNK: LQNK field has linked to line number
2) CCPLNK: LQNK field has process to wakeup to CCP mode
3) ADVLNK: Advise mode LQNK field has line advising

LNK:* [LNKBIT] if on link output (0:7,7)

LOK: Link OK (3:8,8)

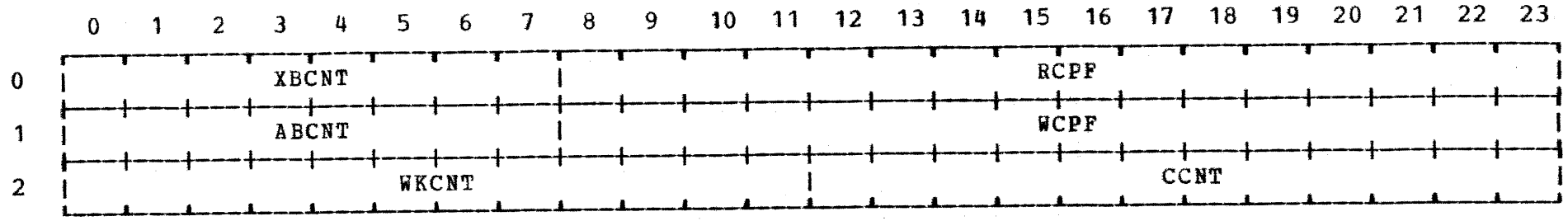
LQNK: Linked to line (2:11,23)

MDTYP:* Device type (0:21,23)
0) Null

```
                1)  Model 37 teletype
                2)  Model 35 teletype
                3)  IBM 2740 selectric
NIQ:        No input requests desired    (1:1,1)
NOQ:        No output requests desired    (1:0,0)
PROC:       Process    (3:11,23)
QUITC:      Quit character    (1:8,15)
RCNO:       Remote concentrator number    (0:11,16)
            RCNO=0 means local bit scanned ttys
            RCNO=1 means local 2400 baud lines
REF:        Resume echo flag    (2:8,10)
                0)  Normal case
                1 to 6)  Resume after N input blocks
                7)  Resume after 7 input blocks
SH1:        SHIFT1 was last character input from device (1:2,2)
WIC:        Wakeup if input wakeup condition is valid    (0:9,9)
WOC:        Wakeup if output wakeup condition is valid    (0:8,8)
WS:         Wakeup strategy - same values as BS.  NOTE:  WS>=BS    (0:2,3)
XWT:        Wait time for guaranteed poor response    (2:0,3)


**  Base address:  MTAA
**  Entry address:  MTAE
**  Base address equal to:  0 mod MTAM(4)
**  Number of entries:  one for each line pair
**  Use:  contains information common to a line pair
**  Entry indexed by:  line number /2
**  Initialization:  all entries must be initialized
```

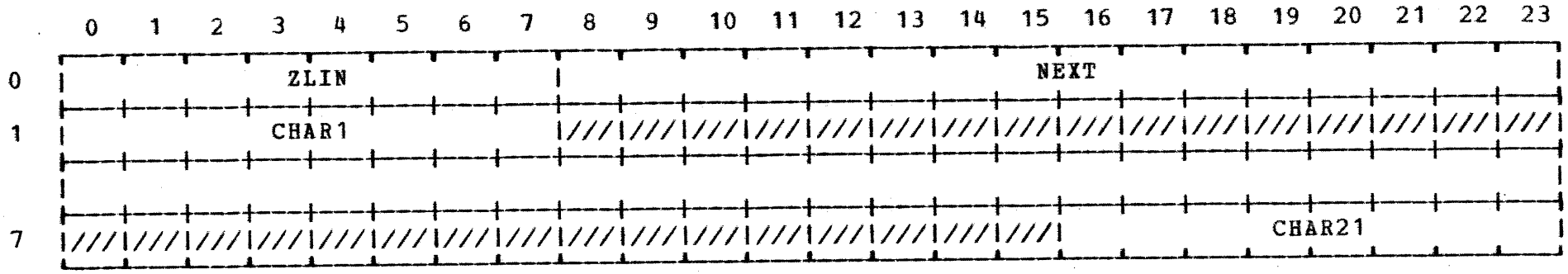
Main Pointer Table - 2 Blocks Input, Then Output (MTCA,MTCE/2=1)



ABCNT: Actual buffer count (1:0,7)
CCNT: Character count (2:12,23)
RCPF: Read character pointer (0:8,23)
WCPF: Write character pointer (1:8,23)
WKCNT: Wakeup character count (2:0,11)
XBCNT: Maximum buffer count (0:0,7)

** Base address: MTCA
** Base address equal to: 1 mod MTCM(2)
** Number of entries: 1 for each line (2 for each line pair)
** Use: contains information unique to a line
** Entry indexed by: line number
** Initialization: all entries, (RCP=WCP=CCNT=ABCNT=0)

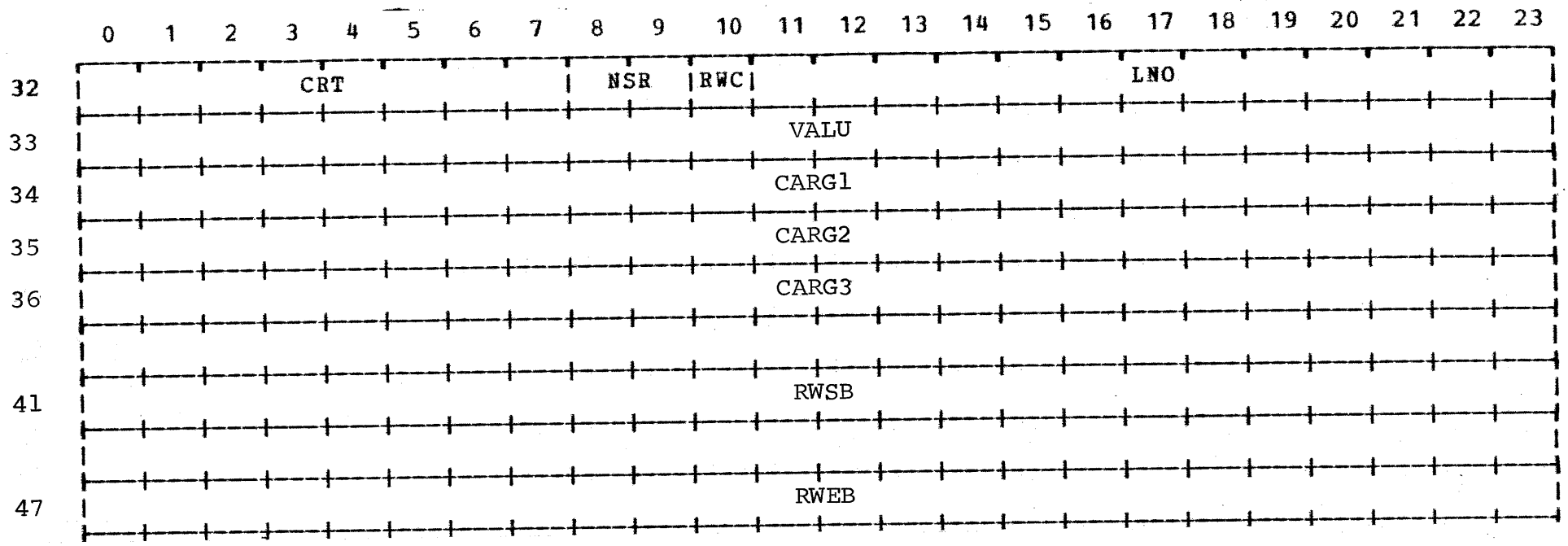
Character Buffer Format (BUF0,/32)



CHAR1: First char in buffer (1:0,7)
CHAR21: Twenty-first character (7:16,23)
NEXT: Next buffer pointer (0:8,23)
ZLIN: Line number mod 256 (used to detect errors) (0:0,7)

** Base address: BUF0
** Entry address: -
** Base address equal to: 0 mod CBOM (32)
** Number of entries: enough for demand
** Use: contains characters. Pointed to by line table
** Entry indexed by: pointer from previous buffer
** Initialization: freelist of buffers linked through NEXT

CPUIT - The CPU Interface Table (CPUIT,/16)



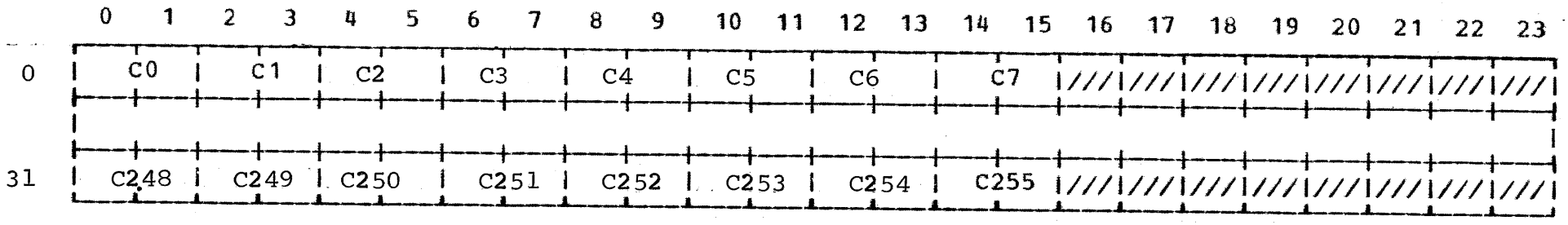
CARG1: First special argument location (43B) (34:0,23)
CARG2: Second special argument location (44B) (35:0,23)
CARG3: Third special argument location (45B) (36:0,23)
CRT: CPU request type (32:0,7)

0 Null request
1 WST (write string)
2 RCND (read character non destructively)
3 RSTB (read string to break character)
4 MBLK (move memory block)
5 SCF (Putfield)
6 LCF (Getfield)
7 STR (store CHIO registers)
8 LDR (load CHIO register)
9 RTUPDATE (update real time clock in core)

```
            10  DOAPOT
            11  DOAPIN
LNO:      Line number for request   (32:11,23)
NSR:      Not satisfied return   (32:8,9)
            0  Normal return
            1  First abnormal return
            2  Second abnormal return (rarely used)
RWC:      [RWCH] request waiting for CHIO   (32:10,10)
RWEB:     End of a 7 word string buffer   (47:0,23)
            The read/write string buffer
RWSB:     First of a 7 word string buffer   (41:0,23)
VALU:     Value of CHIO call (42B)   (33:0,23)


**  Base address:  CPUIT exists in absolute core from 40B to 100B.
**  Entry address:
**  Base address equal to:  40B
**  Number of entries:  1
**  Use:  communications region between CPU,CHIO
**  Entry indexed by:
**  Initialization:  initialization RWCH=0
```

Device Table One Per Device Type ((DVTBA,LB24T),DVTBE,/4)



C0: Character type for 0th character (0:0,1)
- 0) Alphanumeric character (A-Z 0-9)
- 1) Punctuation characters (,.:*.) (etc)
- 2) Echoable control characters (carriage return, line feed)
- 3) Non echoable control characters (control A etc.)

C1: Character type of first character (0:2,3)

C255: Last character type (31:14,15)

C7: Seventh character type (0:14,15)

** Base address: DVTBA
** Entry address: DVTBE
** Base address equal to: 0 mod DVTBM(4)
** Number of entries: 1 per device type (currently 3)
** Use: contains information unique to device
** Entry indexed by: device type number*36 - see device type for public
** line table

Control Characters Used by the CHIO and of External Interest

SHIFT1 *(1B): Shift1 is used to transmit a character between Ø and 37B. The character following shift1 is taken mod 40B. This character would either be sent by the CPU to the CHIO or by the CHIO to the CPU.

CBREAK (27B): Received by CPU if a line becomes disconnected (Break key on teletype depressed). This character is received once. Shift1, 4ØB are received when the break is released.

SUBI (17B): Indefinite number of characters lost in input.

SUB (3ØB): One character lost in input.

INIT (2ØB): This character is used in the procedure to initialize the DCC. (See that Startup Scenario.)

TAG*(1ØB): This character has the delightful property that if it is sent to a device it will not be output by the device, but rather will be echoed back to the CPU program. Thus it can be used to mark whether an input string occurred before or after certain output.

LRC (5): The LRC control character turns device service and slow task processing on and off and allows the RC core to be loaded and dumped over the EFCL. It is always followed by a 'control byte' whose bits are decoded as follows:

1. Load RC core if set, otherwise don't.
3. Run slow tasks if set, otherwise don't.
4. Service devices if set, otherwise don't.

If load is set, the control byte is followed by three characters which give a word count and core address for the block of core to be referenced.

First: The word count (WC) + 40B. Limit is 127 words.

Next: The top 7 bits of core address + 200B.

Last: The bottom 7 bits of core address + 200B.

The core address is the address of the word beyond the last word to be loaded. 2*WC successive characters are then stored. If characters <40B are to be transmitted, they should be prefixed with LRCSH1.

LRCSH1 (23B): See LRC.

CST*(4B): The change status control character takes the character following it as an argument. The 8 bits of the argument are as follows:

1, LFK, UNUSED, EBC, BS1, BS2, ES, UNUSED

The ES bit is only used by the CHIO.
LFK is used only by the remote concentrator.
The CHIO sets both the Wakeup and Break strategy to BS1, BS2.
When this sequence is received, the above bits in the status word are set to the values specified by the argument.
CST should be sent only when local echoing is off. In this case any action that the DCC takes will be held in abeyance, so that no synchronization problem with the CHIO exists.

SYN (26B): Synchronization character; used in the initialization procedure.

NULL*(Ø): This character is legal for WST to use. It will be deleted from the output string.

SNULL*(13B): This character will cause a 1/10 second output delay at the device to which it is sent (except the first SNULL in a string of SNULLs will allow from 1 character time

to 1/10 of a second depending on chance). SNULL is usually used to allow the carriage to return.

* Control characters flaged with an asterisk (*) may be sent by an unprotected CPU program. Any other control characters if sent must have the protect argument for WST set.

III. System Configurator

The system configurator is used to specify which DCCs are connected to the system, the number of logical and physical devices each DCC has, and the group that each DCC is connected to. It also does the initial Load of a DCC core image.

The system configurator is run when the system is started up and after some system crashes. It will set up the hardware configuration by reading a file that specifies the actual configuration. A new DCC can be incrementally added to the system by giving the configurator a file (possibly the teletype) that specifies the new DCC.

Initially, however, the system configurator will execute the appropriate monitor calls to set up the hardware configuration in an ad-hoc manner, rather than being table (file) driven.

The one complex thing that the system configurator must do is to activate a DCC. A special section specifies how this is done in detail.

Startup Scenario

When the loading of a DCC from the M-500 begins, nothing is known about the state of the data concentrator except that it is listening to the EFCL lines. (This might not be true, in which unlikely event an on-site operator must push a reset switch.)

Startup Scenario

The initialization of the DCC consists of two sub-functions: first a file must be generated that specifies a DCC core image, and second it must be loaded into the DCC. The file generation is not discussed here except to note that such a file already exists, although it requires effort on the part of a system programmer to specify a file for a DCC with a different configuration.

The program that loads a DCC core image is now specified in detail.

1. GEN'MS'LINE generates a Medium speed line that is connected to the DCC. (This will eventually be changed to an EFCL.)
2. 16 INITCHARS are sent on this line to the DCC.
3. The line will be turned off when there are no more characters and SYNCHARS will be sent.
4. The DCC should now send a message asking for (re)-transmission of block Ø, followed by a null character. If this does not occur within 1 second, it is tried a few more times (starting at step 2), and if it still fails a message is printed on operators console indicating the DCC does not respond. An on site operator should push a reset switch and the restart procedure should be tried again; if it still fails, either the DCC is malfunctioning or the communications line is inoperative.

5. We have an initial contact, send the DCC a message asking for block Ø (DCC to CHIO) to be transmitted.
6. Send an acknowledgement indicating block Ø follows. (CHIO to DCC).
7. The line will be turned off when there are no more characters and SYNCHARS will be sent.
8. The M-500 should now generate an acknowledgement followed by SYNCHARS. If this does not happen (sigh) go to step 2; and if we continue to fail, the communication line just failed.
9. We are now in a stable situation. Lacking any other impetus the two communications lines will continue sending SYNCHARS. This is also a state which the EFCL line can run if CA is set to a negative number and the state is set to Bliss. Therefore a MAKE'EFCL LINE MCALL is executed that converts the Medium speed line to an EFCL line (setting CA to a large negative value and STATE to BLISS). When this command is executed, the EFCL line is turned back on so that the SYNCHARS will be generated by the EFCL mechanism rather than by the multiplexer.
10. The EFCL line is now running even if in a degenerate sort of way. We now get ready to load the EFCL with its initial core image. This is done by using the LRC facility in the DCC to load up a whole core image.

The DCC will not transmit anything because CA is a large negative number.

11. Get logical device number 1 (the control line) and a free CPU line number (call it L), and connect them. This CPU line will not be used to send the loading information to the DCC.
12. Set GSELDEV to the logical line selected in step 11, and set NCHSR to +1000000.

We now explain what we are doing. The CHIOs output multiplexer is fooled so that it will send up to 1000000 characters to a logical device without its attempting to switch to another line. The DCC output multiplexer, on the other hand, sees that all of the messages it gets are of the form LRC data, so that its multiplexing metaalgorithm never gets invoked. We, however, do have a problem. If the output line empties before the system configurator can be brought into core to write more characters in it, then the output multiplexer would start to output multiplex. Therefore we go through a loop of some complexity that allows CA, the governor counter, to be incremented enough to send not quite all of the characters in the line to the DCC.

13. We wait for CA to be less than or equal to Ø. If it is less than Ø, set it to Ø.
14. For some value of N, write 11*N characters into the CPU line with a WST. (If no more characters, go to 18.)

15. Set CA to CA + N.
16. Block on output for CPU line number 1 (logical device 1).
17. Go to step 13.
18. When there are no more characters to write in the line, an LRC string that zeros the governor counter should be sent, and CA should be set to CA + 3. Because the load has set up certain locations correctly, the initialization task will be activated with the DOL that follows the LRC command. Then CHS characters will be sent because there is no data to multiplex. Similarly, the DCC will send data to the CHIO whenever it inputs a block because of the new value of CA. We are now multiplexing on both input and output. (But even now we are cheating because we are pointing at a degenerate LDT.)
19. The governor counts are now incorrect (they are too low), and the line is operating at less than full bandwidth. Set CA to a large negative number. Record the output block number (AO) and wait a second.
20. Record MSGCTR in EFCL.
21. Wait a second.
22. If the MSGCTR has changed, go to 2Ø.
23. CA should be set to the (number of buffers in LARRY) -1.

24. The multiplexing algorithm is now running correctly with the exception that the DCC may not yet have initialized itself. When the DCC has initialized itself it sends an ACKL on the control line (LDT1) to indicate that initialization is complete. When the ACKL has been received, the DCC is loaded and running properly. (Actually the ACKL was probably sent before steps 19 to 23 were executed).

Kludge Initialization

A much simpler initialization is for the DCC initial load to be loaded by paper tape. TASKI is then started. It does any initialization that is necessary, and begins communication with the CHIO by asking for a (re)transmission of block Ø. When the CHIO is also sending this message the two DCCs will begin to communicate with each other.

With this scheme, the only thing that the monitor needs to do is to set up a correct EFCL with the state set to Confusion. The Timeout timer will generate messages until an acknowledgement is received.

MCALLs Used by the System Configurator

MAKE'EFCL'LINE

1. EFCL Line Number
2. Number of logical devices.*

MAKE'EFCL'LINE converts a Medium speed line into an EFCL line. The CPU line associated with the EFCL line is freed. The EFCL line is turned on. LDT, NCITAB, and LARRY are constructed. All of the values are set to reasonable values. In particular

CA = Ø

GSELDEV = logical device #1;

and NCHSR = -1000000.

If the CPU output line has any characters in it, a failure return is made.

* Eventually this command will be rewritten to allow more generality.

GEN'MS'LINE

1. Multiplexer line number

This command activates a Medium speed line. The value of this MCALL is the CPU line number used to communicate with it.

SET'EFCL'FIELD Set the Value of an EFCL Field

1. DCC Number
2. Field Number

The setable fields that are currently defined are:

1. CA
2. NCHSR

DISC'MS'LINE Disconnect Medium Speed Line

1. Multiplexer line number

The specified Medium speed line is disconnected, and turned off.

A failure return results if the specified line is not connected to a Medium speed line. Pending input and output are both aborted.

MAKE'DEV'TYPE Make Device Type

1. Device Number
2. Number of characters in interval *256 (fractional part need not = Ø).
3. 32 word character type table.

This command, used by the system configurator, allows for the definition of character types. It should be executed before any DCCs are loaded.

MAKE'DEV'TYPE sets up the device type table and adds a new entry to the prototype NCITAB.

A failure return occurs if the device type has already been defined, or if there is no room for a new device type.

IV. The Communications Error Process

The Communications System Error Process is a process that searches for EFCL lines that are generating errors. It can find these lines by checking counters in the EFCL table. Whenever it finds that a line is down it enters into a file the line number and the time. If a line stays down for more than ten seconds, or if it is so bad that less than 1/3 of the bandwidth can be used, the Error Process puts the DCC into triangle mode. This process is also responsible for typing out messages on the operators console indicating these facts.

MCALLs used by the Communications System Error Process

READCSSTAT Read Communications System Statistics

Value:

1. Number of input rate errors counter (EFRECNTR)
2. Number of extra input buffers (XIB)
3. Number of extra output buffers (XOB)
4. Idle count (IDLCNT)
5. Number of times Wakeup delayed due to full wakeup count (BADWK)
6. Number of Null CPU requests (NULRQC)
7. Number of Free Buffers (NFB)
8. Number of characters written by WST (OUTCC)
9. Number of characters read by RSTB (INCC)
10. Real Time Clock Base, high order bits (RHTBASE)
11. Real Time Clock Base, low order bits (RTLBASE)
12. Multiplier for buffer allocation (MP)
13. Current Minimum Buffer Count (MINFB)
14. Current Maximum Buffer Count (MAXFB)

This command is used to get all of the pertinent statistics for the CHIO. Needless to say, none of these counts is reset; therefore, the calling program must save the old values of the statistics if the differential values of these statistics are felt to be of import.

READESTAT Read EFCL Table

1. DCC number

This MCALL reads the 64 word EFCL table.

V. The Listener

The Listener is a utility program that receives information on the status of devices in each DCC for its group on a control line for each DCC.

The Listener is responsible for logging users on and off. It has several MCALLs available for its use.

The Listener is expected to interface with the control task for its DCC. Thus the Listener must be familiar with the RPU control task.

Scenario for Logging on a User

This scenario gives a possible interaction between the Listener and the DCC to log on and off a user.

1. Initially all physical devices are active. A phone rings. The status changes. Suddenly a pirate ship looms on the horizon. The Physical Device Status Reporter sends a message to the Listener indicating the physical device number. This message is repeated every 6 seconds until the phone is answered.
2. The Listener checks to make sure it has enough resources (logical lines, etc.) to answer the phone, and observing it has, gets a free CPU line number and a free logical line number and connects them.
3. The Listener allocates buffers for the logical device in the DCC. (A large input buffer for unknown device type mode).
4. The Listener connects the physical device to the logical device. This initializes all of the entries in the LDT except the buffer pointers to the standard value.
5. The Listener then answers the phone (turns data terminal ready on).
6. The Listener waits for about 2 seconds by which time it should get a message indicating that the data set is ready.

7. The Listener sets the device type to unknown. This sets up bits in the bitscanning tables.
8. The Listener gets a TAG character which it ignores and then several character pairs indicating device types and the character interpreted. It recognizes the carriage return as being on a model 35.
9. The Listener uses the CST character to set the strategy bits in the LDT in the DCC (and the CPU line table in the CHIO).
10. The Listener sends a change device type to model 35 command to the DCC and throws away all garbage until a TAG character is echoed back.
11. The Listener makes the buffer size more commensurate with the volume of characters produced by a 35 (4 characters).
12. The Listener logs on the user.

    ...time passes...
13. The Listener logs off the user.
14. The Listener changes the device type to undefined which stops the bit scanning.
15. The Listener disconnects the CPU line from the Logical line.
16. The Listener changes the buffer sizes to zero, thereby releasing them.
17. The Listener deactivates and busies out the physical device.

MCALLs Used by the Listener

GET'FREE'LDN Get Free Logical Device Number

1. DCC number
2. Group number

An unused logical device number for the specified DCC is returned. A failure return is made if there are no free logical lines in the selected DCC for the selected group. The group number will be ignored in the initial implementation.

GET'CPU'LINE'NO Get Free CPU Line Number

1. Group number

A failure return if no CPU lines are available for the group. Sets up MTAA and MTCA in some standard way. These values can be changed by other MCALLs.

CONNECT'LDN'LINE Connect logical device number to CPU Line Number

1. Logical Device Number
2. CPU Line Number

DIS CON'LDN'LINE Discount logical device to CPU Line Number

1. CPU line number

The specified CPU line number and logical device numbers are freed.

VALUE: Freed Logical device number

VI. RPU Programs

During normal operation of a DCC, there are several RPU tasks running. The most important of these is the Control Task,which is the task that interprets the commands which arrive on the control line, logical device number 1. This task determines the subroutine or task that should be called and calls it. There are several other tasks that perform functions, namely:

The physical device status reporter (PDSR), which scans the physical devices looking for a status change to report to the Model-500.

BSDSERV, which is awakened whenever a character calling for RPU service is typed on a bit-scanned device. For a carriage return it checks line feed kludge to determine if a line feed should be put in the output buffer.

The buffer allocator (BALLOC), which is a low priority task that coalesces buffer space.

The error task (TASKE), which is called when an input buffer is filled. It finds the line buffer that was full and replaces the last character in the buffer with a SUBI indicating that input was lost.

Commands for the Control Task

The Control Task is a DCC task that accepts commands from Model 500 and dispatches on them. Each command is of a well-defined length so that the subprocessor for each command will read to the end of its argument string.

Therefore the command processor knows that the first character specifies the command. This character is allowed to be in the range of 100B to 177B.

The commands are:

Set Physical Device Status (SPDS)

1) Physical device number (bit-scanned devices only)

2: 2-4) Status mask

2:2-7) Status bits

The allowable number of physical devices is 256.

The three status bits are:

2.2,5) Busy out

2.3,6) Data terminal ready (answer the phone)

2.4,7) Active (watch for status changes

Connect Physical Device (CPD)

1) Logical device number

2) Physical device number (bit scanned devices only)

The specified physical device number is connected to the specified logical device number. Or more precisely, the physical device number is put in the logical device table, and the logical device number is put in the physical device table.

Disconnect Logical Device (DLD)

1) Logical line number

The specified logical line number is disconnected from its accompanying physical device, (bit-scanned devices only) and therefore deactivated.

Set Device Type (SDT)

1. Line
2. Type (377B is unknown device type, 376B is undefined device type)

The selected logical device number is set to the selected device type. For bit-scanned devices this requires setting bits in the bit scanning tables (the input-idle and output-active bits); thus the physical device number must be set up. Device type 255D is recognized as the unknown device type. When a bit-scanned device is set by this command (or CPD) it is activated, i.e., the bit scanning table is updated to indicate the new device, if it is a Medium speed device, its tasks are activated. When this command is done a tag is sent back on the logical line affected.

Set Task Descriptors (STD)

1. Logical line number

2,3. Taskdi

4,5. Taskdo

The specified logical line, which must not be a bit-scanned line, has the input and output words loaded with task descriptors. BSD and LSD are turned off in the logical device table.

Load Scratchpad Register (LSPR)

1. Register

2,3. Value

does the obvious thing

Dump Scratchpad Register (DSPR)

1. Register

This command causes the querying program to be sent the value of the selected scratchpad register.

Set Field (SETFLD)

1,2. pointer

3. Index for the field descriptor

The currently defined field discriptors are

0. LSP (lowspeed device) in logical device table

Dump Core Location (DCL)

1,2. Core location

Dump Core (DC)

1,2. First core location

3,4. Number of locations

Load Core Location (LCL)

1,2. Core location

3,4. Contents of core location

Load Core (LC)

1,2. First core location

3,4. Number of locations

5,..n. Contents of the locations

Protect (PROT)

Unprotect (UNPROT)

These commands will ensure that the commands they bracket will be uninterrupted in their execution.

Allocate Buffer for Line (ALLBL)

1. Line number
2. Buffer size
3. Ø for input buffer, 1 for output buffer.

The old buffer in the LDT is freed and replaced with a new buffer of the size specified. If the buffer size is zero, a zero is put in the descriptor if a zero is in the descriptor there is no buffer to free.

Allocate Buffer (ALLB)

1. Index to table to specify where buffer is.
2. Buffer size.

This allocates a buffer, freeing the old buffer.

The currently defined indexes are:

Ø. RBSBUF - RPU service buffer

List of Control Strings from the DCC

Physical Device Status Change (PDSC)

1. Physical device number

2:2-4. New status

2:5-7. Status bits that changed

The three characters following the Command Character give the device number, the 3 bit status, and a 3 bit mask that

indicates which bits have changed status. The bits are

bit 2,5. Ring on line

bit 3,6. Data set ready

bit 4,7. Clear to send

When the ringing signal stops (at the end of an individual ring) its status is not transmitted (as it is redundant). Every six seconds while a phone is ringing, a status change saying the phone is ringing is sent. The Listener can infer the phone has stopped ringing if more than, say, 10 seconds pass without this status being sent. The Listener can, of course, stop the transmission of these signals by deactivating the device.

Scratchpad Value (SPV)

1,2. Scratchpad number

2,3. Value

Core Location Value (one location) (CLV)

1,2. Core location

3,4. Value

Core Dump (CD)

1,2. First location

3. Number of locations

4..n. Value of core locations

Acknowledge Load (ACKL)

The ACKL character is sent by the initialization task when it has finished initialization and the DCC is operating.

Buffer Allocation in the DCC

There is a free storage area in the DCC from which buffers are allocated, each new buffer being allocated from the bottom of the 'free' area, the pointer to the bottom of the free area being moved above the first created buffer. The character count a buffer has is required to be an odd number, meaning that the buffer when full will hold an even number of characters (one less than the buffer size), with two characters wasted: the buffer size character and the unused character of a full buffer. (if a buffer was allowed to get full then the beginning pointer would equal the end pointer, which is indistinguishable from an empty buffer).

The routine that frees a buffer does it by decrementing the buffer size count by one, thus making it an even number. There is a background task that coalesces the free buffer space by searching the buffer storage for free buffers (from the bottom up). It then moves these free buffers slowly to the top, coalescing them as it goes. Whenever a free buffer is moved to the top, ones that are in use must be moved down, so the pointers to these buffers must be found and re-located. This is done by the grossly inefficient method of searching all possible pointers to find the one of interest.

APPENDIX I
EFCL TABLE Field Values

Field	Offset (octal)	Field	Offset (octal)
STATE	0	NCSCHO	24
MSGCTR	1	MISTRESS	25
EBO	2	GOVCTR	26
NCHO	3	NCHSR	41
AO	4	GSELDEV	42
LNBU	5: bits 2,23	NCLII	43
LLBU	5: bits Ø,1	PASS	44
LARRY	6	GHSDEV	45
NTCHO	7	GLDTORG	46
OTCO	10	MATST	47
RELAY	11	not used*	40
OTCI	12	NCIPTR	51
AI	13	NCIWP	52
NCHI	14	IMRST	53
EBI	15	IMRDVN	54
NTCHI	16	CMLICP	55
NCSCHI	17	unused*	56-77
CA	20	TOTCR	27
OBUFQ	21	MESSAGE & ACK. Buffers	30-37
VTIME	22	TIMADDR	40
LBU	23		

*Unused means that the microcode does not explicitly know of these locations. The initialization might put data in them and then point to them. NCITAB for example might be put in 60B-77B.