SOFTWARE COMPONENT SPECIFICATION

SYSTEM:	LEVEL 6 MOD400 OPERATING SYSTEM
SUBSYSTEM:	LOCAL AREA NETWORK
COMPONENT:	LACS DRIVER 802 LOGICAL LINK CONTROL LAYER SERVER
PLANNED RELEASE:	MOD400 4.0
SPECIFICATION REVISION NUMBER:	B
DATE:	JULY 25,1985
AUTHOR:	PETER STOPERA

This specification describes the current definition of the subject software component, and may be revised in order to incorporate design improvements.

TABLE OF CONTENTS

REFERENCES

[1] CLM User Extensions, Richard Taufman, May 14,1979.

[2] Engineering Product Specification (H/W), Local Area Controller Subsystem (LACS), Rev F, A. C. Hirtle, Oct 4, 1984.

[3] Engineering Product Specification, LAN Software, R. Dhondy, Aug. 16, 1985.

[4] LAN S/W Component Specification, System Management, D. O'Shaughnessy
Aug. 16, 1985.

[5] LAN S/W Component Specification, LACS Driver Interface Services, P. Stopera, Aug. 16, 1985.

[6] LAN S/W Component Specification, LACS Driver Megabus Services, P. Stopera, Aug. 16, 1985.

[7] Lan S/W Component Specification, Configuration Requirments L. Vivaldi, Aug. 16, 1985.

[8] LAN S/W Component Specification, LACS Link Layer Protocol, H. King Aug. 16, 1985.

[9] L6 LAN Data Structures, P. Stopera - Aug. 16, 1985

1 INTRODUCTION AND OVERVIEW

1.1 BACKGROUND

The 802 logical link control layer server (802 llc ls) is a component of the lacs driver in the lan subsystem. The 802 llc ls interfaces to the llc type 1 layer instance in the lacs. The 802 llc ls can be termed a l6 entension of the lacs layer instance. The 802 llc ls currently supports type 1 services only.

1.2 BASIC PURPOSE

The 802 llc ls is an extension of the 802 llc layer entity in the lacs. The user of this service will be operating in a connectionless enviroment. The type 1 llc user must associate, then activate the local and remote saps as described in the ldis component specification. The user may then issue read, write, event or deactivate with queue abort requests. The read and write requests must include the remote sap the user wishes to commuicate with..

The 802 llc ls has several purposes, they are: processing read data, transmit data, and event requests and processing flow control at the sap level.

1.3 BASIC STRUCTURE

Figure 1 shows the relation of the 802 llc ls to the other components of the lacs dirver. Figure 2 shows the subcomponents of the 802 llc ls. The following is a brief description of the functions of each subcomponent of the 802 llc ls.

determination routine - This routine determines which routine to branch to, the desision is made from the major function code in the iorb. The routine will branch to the xmit data or read data or event request processing routines depending on the iorb function code.

transmit data request processing routine - This routine will perform pre and post processing of the transmit data iorb. The routine will build a xmit data icb from information in the iorb, issue the icb to the lacs. When the lacs completes the icb, the routine will update the iorb status, then post the request back to the user.

read data request processing routine - This routine will perform pre and post processing of the read data iorb. The routine will build a read data icb from information in the iorb, issue the icb to the lacs. When the lacs completes the icb, the routine will update the iorb status, then post the request back to the user.

LAN L6 SOFTWARE ARCHITECTURE

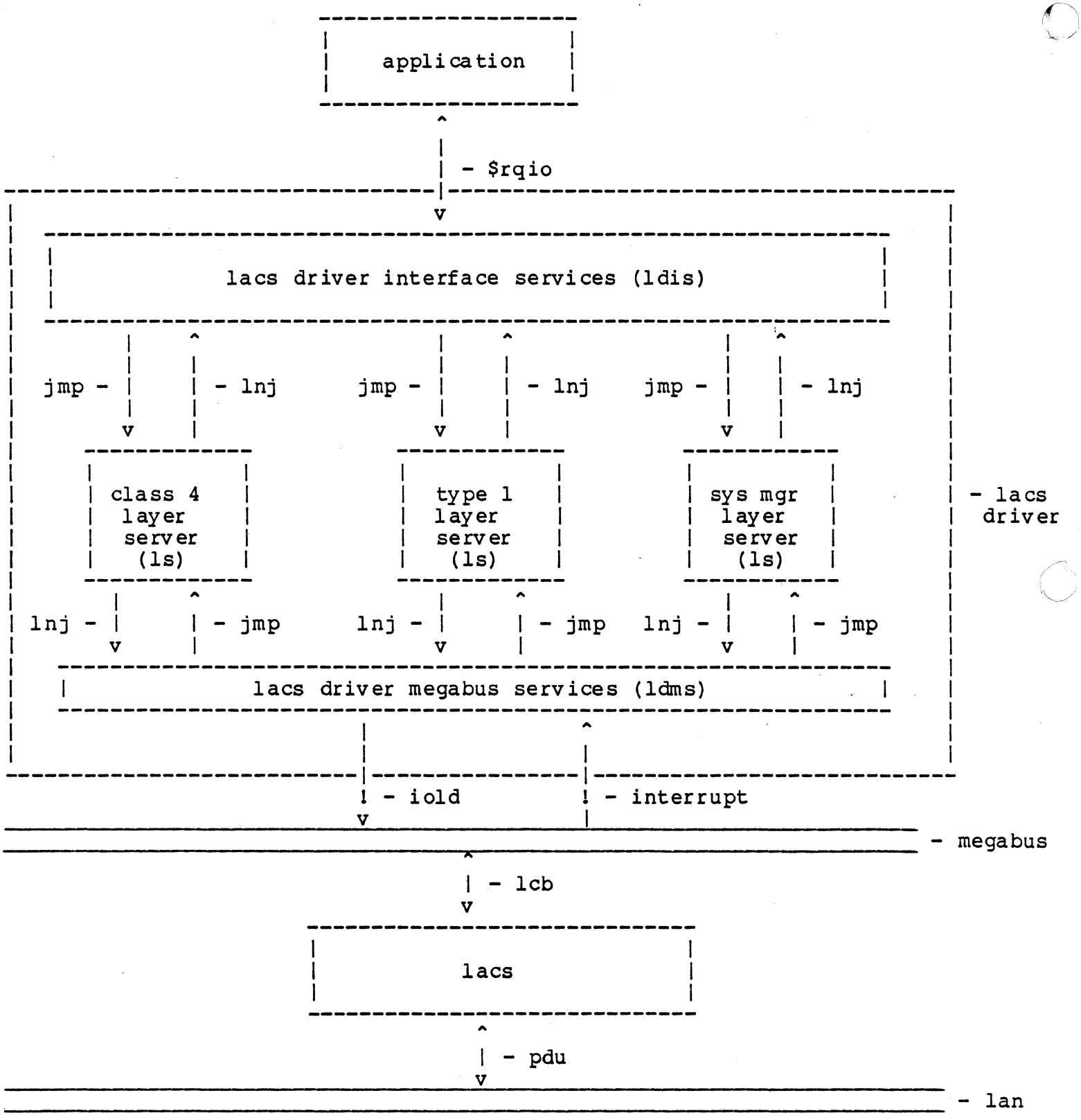

figure 1

LACS DRIVER TYPE 1 LAYER SERVER SUBCOMPONENTS

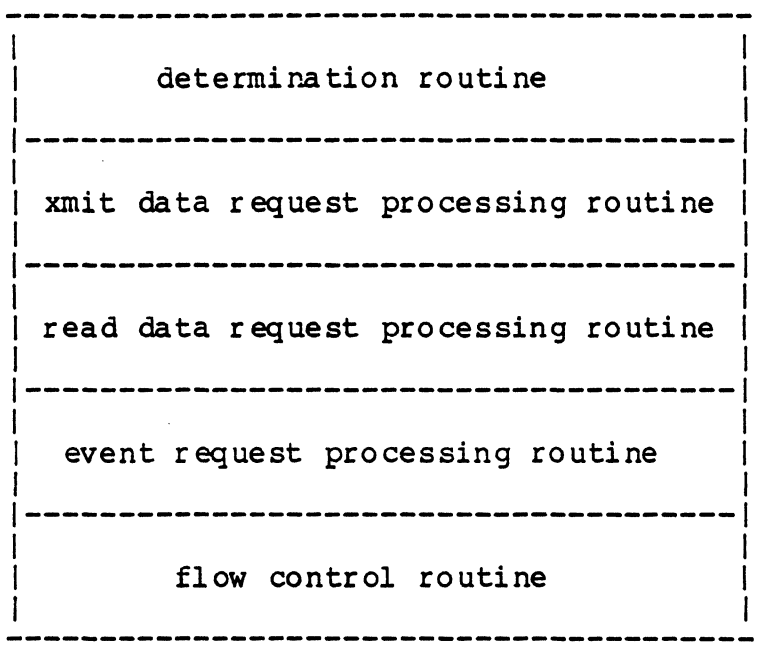

figure 2

event request processing routine - This routine will perform pre and post processing of event iorbs. The routine will build an event lcb from information in the iorb and the event mask in the rt, then issue the lcb. When the lacs completes the lcb, the routine will update the iorb status, then post the request back to the user.

flow control routines - These routines will test flow control counts for the sap. The lacs maintains the counts and these routines update the counts via a credit mechanism which is defined in the eps.

issued to the lacs. After the determination the routine will return to the calling routine wilh a status specifying whether the lcb can be sent or not.

1.4 BASIC OPERATION

1.4.1 DETERMINATION ROUTINE

The determination routine is invoked via a branch from the ldis determination routine. All requests issued to the 802 llc is enter at this routine. The routine will fetch the major function code from the iorb, from the function code the routine will branch to either the xmit data routine, read data routine or event routine.

1.4.2 TRANSMIT DATA REQUEST PROCESSING ROUTINE

The xmit data request processing routine will be invoked via a branch from the determination routine for pre request processing or a call from the ldms when a lcb is completed for post request processing. Therefore the routine has two subcomponents: pre request processing and post request processing. The routines are defined below:

xmit data pre request processing routine - This routine will validate the request. Call the build lcb routine in the ldis. The routine will fill in the llc type 1 specific fields of the lcb. The routine will call the executive address absoultizing routine and place the address(es) into the lcb. The routine will call the flow control routine. Upon return from the flow control routine if there is no error the routine will call the ldms to issue the lcb to the lacs. When the ldms return the routine will call the terminate task rotuine in the ldis.

Errors reported by the xmit data pre request processing routine are:

1. Invalid iorb parameter.
2. Flow control routine returns an error.
3. Ldms rotuine returns an error.

4. Build lcb routine returns an error.

xmit data post request processing routine - This routine will be invoked at interrupt level from a call from the ldms. The routine will be passed a pointer to a completed lcb, from the lcb the routine will find the iorb associated with the lcb, update the iorb status from information in the completed lcb. The routine will call the flow control routine. The routine will then return to the ldms so the ldms can finish it's interrupt processing.

Errors are reported by the xmit data post request processing routine are:

1. Flow control routine returns with an error.

1.4.3 READ DATA REQUEST PROCESSING

The read data request processing routine will be invoked via a branch from the determination routine for pre request processing or a call from the ldms when a lcb is completed for post request processing. Therefore the routine has two subcomponents: pre request processing and post request processing. The routines are defined below:

read data pre request processing routine - This routine will validate the request. Call the build lcb routine in the ldis. The routine will fill in the llc type 1 specific fields of the lcb. The routine will call the executive address absoultizing routine and place the address(es) into the lcb. The routine will call the flow control routine. Upon return from the flow control routine if there is no error the routine will call the ldms to issue the lcb to the lacs. When the ldms return the routine will call the terminate task rotuine in the ldis.

Errors which may occur in the read data pre request processing routine are:

1. Invalid iorb parameter.

2. Flow control routine returns an error.

3. Ldms rotuine returns an error.

4. Build lcb routine returns an error.

read data post request processing routine - This routine will be invoked at interrupt level from a call from the ldms. The routine will be passed a pointer to a completed lcb, from the lcb the routine will find the iorb associated with the lcb, update the iorb status from information in the completed lcb. The routine will call the flow control routine. The routine will then return to the ldms so the ldms can finish it's interrupt processing.

Errors are reported by the read data post request processing routine are:

1.4.4 EVENT REQUEST PROCESSING

The event request processing routine will be invoked via a branch from the determination routine for pre request processing or a call from the idms when a icb is completed for post request processing. A chagne event mask request can also be issued. Therefore the routine has three subcomponents: pre request processing and post request processing and change event mask. The routines are defined below:

event request pre processing routine - This routine will upon receiving the request determine from the device specific bits in the iorb if the request is a change event mask request and if so will jump to the change event mask routine. Otherwise, the routine will set the event request processing bit in the first tt. If there is an event request outstanding the routine will dequeue and post the current event with an error status, then enqueue the reuqest on the first tt. The routine will then test if there any event icbs queued on the event icb queue. If there are icbs queued the routine will check the mask of each completed event icb for a matching mask in the iorb. If a matching mask is found, the event request is updated, the event which occured is placed in the iorb and the request is dequeued and posted, the routine will then turn off the event request processing bit off and terminate the task. If a matching mask is not found in the queue of event icbs, the routine will turn off the event request processing bit off and terminate the task. If no icb are on the event queue, the routine will turn of the event request processing bit and terminate the task.

Errors which may occur in the event request pre processing routine are:

1. Invalid iorb parameter.

event request post processing routine - This routine will be invoked when a event icb is completed by the iacs. The routine will test the event request processing bit, if the bit is set the routine will queue the icb on the tail of the event icb queue off the first tt, then return to the idms. If the bit is not set, the routine will test if there is an event iorb outstanding. If there is no event iorb outstanding the routine will queue the icb on the tail of the event icb queue off the tt, then return to the idms. If an event iorb is outstanding, the routine will test if the event iorb matches the mask in the event icb, if the masks do not match the icb is queued on the tail of the event icb queue off the tt, and returns to the idms. If a match is found the routine will dequeue the event iorb, update the iorb status,

place the event mask from the lcb into the lorb, then posts the request, the routine will then return to the ldms.

Errors which may occur in the event request post processing routine are:

Currently there no errors.

change event mask routine - This routine will be invoked by the event request pre processing routine. When this routine is activated there will be a pointer to an change event mask lorb. The routine will change the event mask field in the tt, then dequeue and post the reuqest, then terminate the task.

Errors which may occur during processing of the change event mask rotuine are:

1. Invalid lorb parmameter.

1.4.5 FLOW CONTROL PROCESSING

There are two flow control routine in the 802 type 1 llc ls, they are pre order flow control processing and post order flow control processing.

pre order flow control routine - This routine will be called by the read data or transmit data pre request processing routines. The routine will test if the user has exceeded the read or write credit value, and if so the routine will post the request back to the user with an error, then return to the caller with an error status. Otherwise, the routine will return to the calling routine with a successful status.

post order flow control routine - This routine will be called by the read data or transmit data post request interrupt processing outines. The routine will add the credit amount to the current credit amount, then return to the calling routine.

2 EXTERNAL SPECIFICATIONS

2.1 OWNED DATA STRUCTURES

The the data structures owned and used by the lan subsystem are defined in the lan 16 data structures document.

2.2 EXTERNAL INTERFACES

2.2.1 MOD400 EXECUTIVE SOFTWARE ROUTINES

2.2.1.1 ZHCOMM - Null address

function: Will load the null address when referenced i.e. ldb

$b5,<zhcomm will load $b5 with the null address.

2.2.1.2 ZXD_TR - Internal terminate

entry: lnj $b5,zxd_tr

input: $r2 = completion status for request
$b4 = new default start address or null
$b5 = return

output: $r1 = 0 - no error on internal terminate
$b1 = address of IRB for next request or is null if hardware interrupt
$b4 = address of RB for next request

modifies: any register may be modified

function: Dequeue and post currently dispatched request. Get next request in queue of task requests. Delete task if queue empty and delete bit on. Suspend task until next request or hardware interrupt at issuing task's level if queue empty and delete bit off.

2.2.2 STANDARD IORB FORMAT

rb_irx

input - bit 0 - rb_adr points to a bd when set
bit 1-3 - na
bit 4-f - irn when rb_ct2 bits 0-7 = x'fd'
output - na

rb_rrb

input: na
output: na

rb_ct1

input: bit 0-7 - mbz
bit 8-e - na
bit f - must be set
output: bit 0-7 - status
60 - event successful
64 - invalid iorb parmaeter
6b - event aborted, reference rb_ fss field for reason
6c - inconsistent request, reference rb_ fss field for reason
bit 8-f - same as input

rb_ct2

```
      input:    bit 0-7 - lrn
                bit 8-a - na
                bit b -   must be set
                bit c-f - function code
                  bit c-f = e - event iorb
                  bit c-f = 1 - write iorb
                  bit c-f = 2 - read iorb
      output:   same as input
```

rb_adr

```
      input:    pointer to buffer, or if rb_lrx bit 0 is set this
                field contains a pointer to a buffer discriptor,
                or this field may be null if no data is being
                passed (i.e. event iorb)
      output:   same as input
```

rb_rng

```
      input:    range of the buffer, or total range of all buffers
                in the buffer discriptor block if bit 0 in rb_lrx
                is set, or mbz if no data is being passed
      output:   same as input
```

rb_dvs

```
      input:    bit 0-7 - class of service
                bit 8-f - mbz
      output:   same as input
```

rb_rsr

```
      input:    mbz
      output:   residual range of the buffer for read operations
                only, or same as input for all other operations
```

rb_st1

```
      input:    mbz
      output:   bit 0-7 - same as input
                bit 8 -   invalid function code when set
                bit 9 -   ram memory exausted when set
                bit a -   ram location non-existent when set
                bit b -   ram parity error when set
                bit c -   level 6 memory yellow when set
                bit d -   level 6 memory non-existent when set
                bit e -   level 6 bus parity error when set
                bit f -   level 6 memeory red when set
```

rb_ext

```
      input:    bit 0-7 - xx
                bit 8-f - xx
      output:   same as input
```

rb_fsf

```
      input:    function specific function code
                  iorb function code = e
                          0010 - sap event
                          0020 - connection event
                          0040 - sm event
                  iorb function code = 1
                          0010 - ci write
                          0020 - co write
                          0040 - expideted co write
                  iorb function code = 2
                          0010 - ci read
                          0020 - co read
                          0040 - expideted co read
      output:   same as input
```

rb_fss

```
      input:    mbz
      output:   function specific status
                  0001 - sap already active
                  0002 - lack of resources
                  0004 - controller down
                  0008 - sm is error
                  0010 - lrn already in use
                  0020 - sap already active
                  0040 - sap already disconnected
                  0080 - receive buffer too small
```

rb_abs

```
      input:    mbz
      output:   accual buffer size when rb_fss = 0080, otherwise
                same as input
```

rb_lra

```
      input:    logical remote address for writes, mbz for read
                and event requests
      output:   logical remote address for read, otherwise same as
                input
```

2.2.2.1 EVENT IORB EXTENSIONS

rb_evm

```
input:    event mask
            bit f - data arrival event
            bit e - additional write credits available
            bit d - sap deactivated
output:   same as input
```

rb_evi

```
input:    mbz
output:   event indication mask
            bit f - data arrival event
            bit e - additional write credits available
            bit d - sap deactivated
```

rb_elf

```
input:    mbz
output:   amount of additional write credit when rb_evi bit
          e set, otherwise same as input
```

rb_ecb

```
input:    mbz
output:   trash
```

2.2.2.2 WRITE IORB EXTENSION

rb_awc

```
input:    mbz
output:   amount of additional write credit
```

rb_wcb

```
input:    mbz
output:   trash
```

2.2.2.3 READ IORB EXTENSION

rb_arc

```
input:    mbz
output:   amount of additional read credit
```

rb_rcb

input: mbz
output: trash

2.2.3 STANDARD LCB FORMAT

cb_pri

input: mbz
output: na

cb_ncb

input: mbz
output: na

cb_rct

input: address of caller's rct
output: same as input

cb_lit

input: address of the lit in which this lcb will be queued
output: same as input

cb_frw

input: bit 0-3 - 9
bit 4-7 - mbz (iorb major function code ?)
bit 8-f - xx
output: same as input

cb_itp

input: address of the post processing routine, or trb, on null
output: same as input

cb_ind

input: indicators
bit 7 - cb_itp points to a trb when set
bit 6 - sm_lcb when set
all other bits mbz
output: same as input

cb_icw

```
      input:    bit 0-5 - mbz
                bit 6-9 - cpu number to interrupt
                bit a-f - level to interrupt the cpu
      output:   same as input
```

cb_fsf

```
      input:    function specific function code
                  function codes for read icbs are:
                          0012 - cl read
                          0022 - co read
                          0042 - co expideted read
                  function codes for write icbs are:
                          0011 - cl write
                          0021 - co write
                          0042 - co expideted write
                  function codes for event icbs are:
                          001e - sap event
                          002e - connection event
                          004e - sm event
      output:   same as input
```

cb_cts

```
      input:    mbz
      output:   bit 0-7 - rfu and mbz
                bit 8 -   invalid function code when set
                bit 9 -   ram memory exausted when set
                bit a -   ram location non-existent when set
                bit b -   ram parity error when set
                bit c -   level 6 memory yellow when set
                bit d -   level 6 memory non-existent when set
                bit e -   level 6 bus parity error when set
                bit f -   level 6 memeory red when set
```

cb_fss

```
      input:    mbz
      output:   function specific status
                  0001 -  sap not active
                  0002 -  lack of resources
                  0004 -  controller unavailable
                  0008 -  sm layer instance error
                  0020 -  sap already active
                  0040 -  sap already deactivated
                  0080 -  recieve buffer too small
                  0100 -  illegal logical address
                  0200 -  invalid icb
                  0400 -  write credit violations
                  0800 -  read credit violations
```

cb_cbs

input: mbz
output: bit 0 - lcb is complete when set
bit 1 - lcb not processed when set
bit 2-f - rfu and mbz

cb_abs

input: mbz
output: actual buffer size if cb_fss = 0080, otherwise same as input

cb_lsa

input: logical local address for cl operations
output: same as input

cb_lra

input: logical remote address for cl write operation, mbz for event and read operations
output: logical remote address for cl read operations, otherwise same as input

cb_trg

input: total byte range
output: same as input

cb_bct

input: number of buffers
output: same as input

cb_ad1

input: buffer #1 address
output: same as input

cb_rg1

input: buffer #1 range
output: same as input

cb_rs1

input: mbz
output: buffer #1 residual range

cb_ad2

input: buffer #2 address
output: same as input

cb_rg2

input: buffer #2 range
output: same as input

cb_rs2

input: mbz
output: buffer #2 residual range

cb_ad3

input: buffer #3 address
output: same as input

cb_rg3

input: buffer #3 range
output: same as input

cb_rs3

input: mbz
output: buffer #3 residual range

cb_ad4

input: buffer #4 address
output: same as input

cb_rg4

input: buffer #4 range
output: same as input

cb_rs4

input: mbz
output: buffer #4 residual range

cb_ad5

input: buffer #5 address
output: same as input

cb_rg5

input: buffer #5 range
output: same as input

cb_rs5

input: mbz
output: buffer #5 residual range

cb_ad6

input: buffer #6 address
output: same as input

cb_rg6

input: buffer #6 range
output: same as input

cb_rs6

input: mbz
output: buffer #6 residual range

cb_ad7

input: buffer #7 address
output: same as input

cb_rg7

input: buffer #7 range
output: same as input

cb_rs7

input: mbz
output: buffer #7 residual range

cb_ad8

input: buffer #8 address
output: same as input

cb_rg8

input: buffer #8 range
output: same as input

cb_rs8

input: mbz
output: buffer #8 residual range

2.2.3.1 EVENT LCB EXTENSIONS

cb_evi

```
input:    mbz
output:   event indication mask
          bit f -   data arrival event
          bit e -   additional write credits availbale
          bit d -   sap deactivated
```

cb_elf

```
input:    mbz
output:   event information field
          aditional write credit (only if cb_evi = 0002)
          lenght of read buffer (only if cb_evi = 0001)
          reason for sap deactivatin (only if cb_evi = 0004)
```

2.2.3.2 READ LCB EXTENSION

cb_arc

```
input:    mbz
output:   read credit count
```

2.2.3.3 WRITE LCB EXTENSION

cb_awc

```
input:    mbz
output:   additional write credit
```

2.2.4 MOD400 EXTERNAL DATA STRUCTURES IMPLEMENTED

The following system owned data structures are referenced by the llc 802 llc is:

Task Control Block (TRB)
System Control Block (SCB)
Logical Resource Table (LRT)
Group Control Block (GCB)
Resource Control Table (RCT)
Intermediate Request Block (IRB)

2.2.5 LACS DRIVER INTERFACE SERVICES ROUTINE USED

The 802 llc is uses the following ldis routines, the definition of the input and output parameters can be found in the ldis component specification:

isblcb - build lcb
isviob - validate iorb
istmtk - terminate task
isevnt - evnet routines
isasvb - assign segiment visibility
isabsl - absoultize buffers

2.2.6 LACS DRIVER MEGABUS SERVICES ROUTINES USED

The 802 llc ls uses the following ldms routines, the definition of the input and output parameters can be found in the ldms compnent specification.

msllor - issue lold or issue lo routine

2.3 INITIALIZATION REQUIREMENTS

The clm process will load the llc 802 llc ls into system memory, and configure at least one task level for the llc 802 llc ls to run under. When the llc 802 llc ls is loaded into memory, it will perform a initialization subroutine. This routine will:

1. Reclaim patch space.

No initialization processing is done when the llc 802 llc ls is activated initially by a request.

2.4 TERMINATION REQUIREMENTS

The llc 802 llc ls will be active as long as mod400 is active, therefore there are no termination requirements.

2.5 ENVIRONMENT

The following items are required by the llc 802 llc ls for it to perform it's task:

1. Mod400 operating system.
2. Any l6 computer model except 6/10 and 6/20.
3. A lacs attached to the l6 megabus.
4. Lan clm.
5. System manager layer server.
6. A user of the lan subsystem to driver the llc 802 llc ls.

2.6 TIMING AND SIZE REQUIREMENTS

Currently memory usage and timing requirements are not an issue. However, the code should be be as efficient as possible.

2.7 ASSEMBLY AND LINKING

The software will be written in Series 6 Assembly Language using a subset of the instruction set that is present on all Series 6 systems. The llc 802 llc ls will be linked by the gcos6 mod400 linker to produce one of the lacs driver's bound units.

2.8 TESTING CONSIDERATIONS

Since the product is new, all functions will be tested by the developer, and software test.

2.9 DOCUMENTATION CONSIDERATIONS

The llc 802 llc ls source listing will include a program design language used by the developer to aid in the maintenance by future developers and also to aid in the development by the developer.

2.10 ERROR MESSAGES

2.10.1 IORB ERROR MESSAGES

1. 0164 - Invalid iorb parameters.
2. 016b - Request was not processed.
3. 016c - Inconsistent request.

3 INTERNAL SPECIFICATION

3.1 OVERVIEW

The 802 llc ls is activated by either the ldls determination routine branching to the type 1 determination routine or a call from the ldms to one of the 802 llc ls interrupt routines, the interrupt routine must return to the ldms.

3.2 SUBCOMPONENT DESCRIPTION

3.2.1 DETERMINATION ROUTINE

The determination routine ls invoked by the ldls determination routine. The ldls determination routine performs a branch to the 802 llc ls determination routine. The ldls determination routine will supply the following input parameters:

```
$b1 = a(lrb)
$b2 = a(rct)
$b4 = a(iorb)
```

The 802 llc ls determination routine supplies the following output parameters:

```
$b1 = a(lrb)
$b4 = a(iorb)
$b2 = a(rct)
```

The routine performs the following function:

1. Retrieve the major function code from the iorb.

2. Using a jump table index into the table by the major function code to jump to the appropriate routine. The jump table is described below:

```
jmptbl    equ  $
          jmp  error      function code = 0
          jmp  xmitdata   function code = 1
               jmp   readdata      function
                                   code = 2
               jmp   error         function
                                   code = 3
                     :
                     :
               jmp   event         function
                                   code = e
               jmp   error         function
                                   code = f
```

Where xmitdata is the xmit data routine, readdata is the read data routine, and event is the event routine, and error is a invalid function code handling routine.

3.2.2 XMIT DATA ROUTINES

3.2.2.1 XMIT DATA PRE PROCESSING ROUTINE

The xmit data pre processing rotuine is invoked by the 802 llc is determination routine. The xmit data preprocessing routine requires the following input parameters:

```
$b1 = a(irb)
$b2 = a(rct)
$b4 = a(iorb)
```

The xmit data pre processing rotuine supplies the following output parameters:

none - task is terminated

The routine performs the following function:

1. Validate the iorb, this involves testing the extension field to see if the iorb is of proper lenght, the range field must be non zero, and test of other fields that will be determined when the routine is inplemented. If an iorb is invalid, the routine will call the dequeue and post routine with an invalid iorb error status.

2. If the iorb is valid the routine will set a pointer to the icb (in the extended iorb), then call the build icb routine in the idis.

3. Upon return from the build lcb routine, the rotine will fill in the following fields of the lcb:

   cb_frw - set the left byte to 9, set the right byte to the range of the lcb from the cb_sz word.

   cb_itp - set the address of the xmit data post processing routine.

   cb_ind - currently no bits are set, field is cleared.

   cb_csf - set the channel specific function code, where the code is retrieved from is tbd.

4. The routine will call the flow control routine, if an error occurs the routine will call the terminate task routine.

5. If no error resulted from the call to the flow control routine, the xmit data pre processing routine will set up the registers for the call to the ldms, then call the ldms.

6. If the ldms returns to the xmit data pre processing routine with an error the routine will tbd, then call the terminate task routine.

7. If the ldms returns with no error, the xmit data routine will call the terminate task routine.

3.2.2.2 XMIT DATA POST PROCESSING ROUTINE

The xmit data post processing routine is invoked by the ldms when an lcb completes. The ldms retrieves the cb_itp field from the lcb and performs a lnj to the address. The address in the lcb will be this routine. The xmit data pre processing routine sets up the address. The xmit data post processing routine requires the following input parameters:

$b1 = a(lcb)
$b5 = return address

The routine will supply the following output parameters:

none - returns to the address in the inputed $b5

The routine performs the following function:

1. Retrieve the pointer to the iorb associated with the lcb.

2. Update the rb_st1 in the iorb status word from the cb csf and cb_cts fields in the lcb.

3. Call the flow control routine.

4. Call the dequeue and post routine in the ldis.

5. Return to the ldms.

3.2.3 READ DATA ROUTINES

3.2.3.1 READ DATA PRE PROCESSING ROUTINE

The read data pre processing rotuine is invoked by the 802 llc is determination routine when a read iorb is issued by the user. The read data preprocessing routine requires the following input parameters:

$b1 = a(irb)
$b2 = a(rct)
$b4 = a(iorb)

The read data pre processing rotuine supplies the following output parameters:

none - task is terminated

The routine performs the following function:

1. Validate the iorb, this involves testing the extension field to see if the iorb is of proper lenght, the range field must be non zero, and test of other fields that will be determined when the routine is inplemented. If an iorb is invalid, the routine will call the dequeue and post routine with an invalid iorb error status.

2. If the iorb is valid the routine will set a pointer to the lcb (in the extended iorb), then call the build lcb routine in the ldis.

3. Upon return from the build lcb routine, the rotine will fill in the following fields of the lcb:

   cb_frw - set the left byte to 9, set the right byte to the range of the lcb from the cb_sz word.

   cb_itp - set the address of the read data post processing routine.

   cb_ind - currently no bits are set, field is cleared.

   cb_csf - set the channel specific function code, where the code is retrieved from is tbd.

4. The routine will call the flow control routine, if an error occurs the routine will call the terminate task routine.
5. If no error resulted from the call to the flow control routine, the read data pre processing routine will set up the registers for the call to the ldms, then call the ldms.

6. If the ldms returns to the read data pre processing routine with an error the routine will tbd, then call the terminate task routine.

7. If the ldms returns with no error, the read data routine will call the terminate task routine.

3.2.3.2 READ DATA POST PROCESSING ROUTINE

The read data post processing routine is invoked by the ldms when an lcb completes. The ldms retrieves the cb_itp field from the lcb and performs a lnj to the address. The address in the lcb will be this routine. The read data pre processing routine sets up the address. The read data post processing routine requires the following input parameters:

$b1 = a(lcb)
$b5 = return address

The routine will supply the following output parameters:

none - returns to the address in the inputed $b5

The routine performs the following function:

1. Retrieve the pointer to the iorb associated with the lcb.

2. Update the rb_st1 in the iorb status word from the cb csf and cb_cts fields in the lcb.

3. Call the flow control routine.

4. Call the dequeue and post routine in the ldis.

5. Return to the ldms.

3.2.4 EVENT ROUTINES

3.2.4.1 EVENT REQUEST PRE PROCESSING ROUTINE

The event request pre processing routine is invoked by the type 1 determination routine via a branch. The routine will require the following input parameters:

$b1 = a(irb)
$b2 = a(rct)
$b4 = a(iorb)

The read data pre processing rotuine supplies the following output parameters:

none - task is terminated

The routine performs the following function:

1. If bit 0 in the rb_dvs word is not set, then call the post request routine with an invalid iorb parameter status, call the terminate task routine.

2. If bit 0 is set in the rb_dvs word, set the event active bit in the tt_idl word in the tt.

3. If there is an event iorb pointer in the tt_erb field, then call the post request routine to post the request pointed to by the tt_erb field with an error status.

4. Place the pointer to the iorb into the tt_erb field.

5. If any event lcbs are are in the event lcb queue then test search the queue looking for a matching event mask, mask in iorb and lcb must match. If a match is found call the post request routine with a successful status, clear the tt_erb field, set the event active bit off and call the terminate task routine.

6. If no event lcb are in the queue or a matching mask is not found to the iorb in the event lcb queue set the event active bit off, call the terminate task routine.

3.2.4.2 EVENT REQUEST POST PROCESSING ROUTINE

The event request post processing routine is invoked by lnj from the ldms when it has a completed event lcb. The routine requires the following input parameters:

$b1 = a(lcb)
$b5 = return address

The routine supplies the following output parameters:

none - a return to the ldms is performed

the routine performs the following function:

1. Retrieve the pointer to the tt.

2. If the event active bit is set, queue the lcb on the tail of the event lcb queue, return to the ldms.

3. If the event active bit is not set, test if there is an event iorb, if there is no event iorb queue the lcb on the tail of the event lcb queue, return to the ldms.

4. If there is an event iorb, queue the lcb on the tail of the event lcb queue, search from the start of the event lcb queue, if a matching mask is found, call the post routine with a successful status, return to the ldms.

5. If there is no matching mask found return to the ldms.

3.2.5 FLOW CONTROL ROUTINES

3.2.5.1 PRE ORDER FLOW CONTROL ROUTINE

The pre order flow control routine is invoked via a lnj from the recieve or transmit data pre request processing routines. The routine requires the following input parameters:

```
$b4 = a(iorb)
$b2 = a(rct)
$b5 = a(return)
```

The routine supplies the following output parameters:

```
$b4 = a(iorb)
$b2 = a(rct)
$r1 = status
   0000 - ok to send lcb
   0001 - user exceeded flow control values
```

The routine performs the following function:

1. Retrieves the current read or transmit credit count from the lth tt, this is determined from the iorb function code.

2. If the count is zero, call the dequeue and post routine with an unsuccesful error status, return to the caller.

3. If the count is non zero, return to the caller.

3.2.5.1 POST ORDER FLOW CONTROL ROUTINE

The post order flow control routine is invoked via a lnj from the recieve or transmit data post request processing routines. The routine requires the following input parameters:

```
$b1 = a(lcb)
$b2 = a(rct)
$b5 = a(return)
```

The routine supplies the following output parameters:

```
$b1 = a(lcb)
$b2 = a(rct)
$r1 = status
   0000 - ok to send lcb
   0001 - user exceeded flow control values
```

The routine performs the following function:

1. Retrieves the current read or transmit credit count from the lth tt, this is determined from the lcb function code.

2. Add to the current count the credit value in the lcb.

3. Return to the caller.

4 PDL

TBD

5 ISSUES