Lan Test Software Component Interface Specification

SYSTEM:	Level 6
SUBSYSTEM:	DSA/OCL
COMPONENT:	LAN IN-LINE/ON-LINE Diagnostic - LLT1
PLANNED RELEASE:	MOD 400 R4.1
SPECIFICATION REVISION NUMBER:	1
DATE:	16 August, 1985
AUTHOR:	Alan P. Hicks

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] System Management Software Component Specification, Rev 1
D. O'Shaughnessy, 29 July 85

[2] Lacs Driver Interface Services, Revision B
P. Stopera, 19 July 85

[3] Layer Management Proposal for 802.2.2.3, 5 July 85

[4] CCITT RECOMMENDATION X.409, MESSAGE HANDLING SYSTEM:
Presentation Syntax and Notation, September 84

[5] QLT LAN CONTROLLER, S. Patel, 10 April 85

[6] The Am7990 Family IEEE 802.3/Ethernet Node
Advanced Micro Devices, November 84

[7] 09-0016-00 ESPL Software Technical Reference Manual, Vol 1
Kernel and Support Software (Bridge Communications, Inc.)

[8] MAC Firmware Component Specification, Rev 1
P. Collins, 26 July 1985

ABBREVIATIONS AND DEFINITIONS

COLLISION - A protocol whereby each node attempts to place a frame on the lan. If two nodes simultaneously access the lan, the messages are said to collide. Each node then waits a random amount of time and tries again.

CONSTRUCTOR - A term used by X.409 to describe a data element in the PDU whose contents is a constructor, series of constructors, a primitive, or a series of primitives.

CSMA/CD - A term used in decribing any lan which uses collision detection to control access for any node to that lan.

DSA - Distributed Systems Architecture is an attempt to create an open system architecture for distributed processing modeled on an earlier versimodeled on of the ISO standard.

IN-LINE - A test which uses the entire resource it is testing, whether it be a controller, adapter or physical cable. No normal use of the resource under test is permitted.

IORB - Input/Output Request Block is a method of passing i/o parameters to the MOD 400 communication executive.

LAN - Local Area Network

LLC - Logical Link Control is a network layer immediately above the MAC layer which controls logical connections.

LMI - Layer Management Interface is the point to which SM passes control information at each layer whether it is MAC, LLC or another layer.

LRN - Logical Resource Numbers are numerical values assigned to various physical and logical resources of a system either automatically or as selected values in the system's configuration load manager (CLM) file.

MAC - Media Access Control is the layer immediately above the PHYSICAL layer. This layer determines access to the physical adapter. It is very highly protocol specific.

NAIAD - Node Administrator application Interface for Administration allows a application to use Node Adminstrator services without being linked to the Node Adminstrator.

OCL	- The Operator Control Language used by system operators to control MOD 400 from the operator's console.
ON-LINE	- A test which shares the resource it is testing with normal users of that resource.
PDU	- Protocol Data Unit is described by X.409 and is used to standardize data transmission between network layers.
PRIMITIVE	- A data element defined by X.409 which has no other sub-elements.
SAP	- Service Access Point is the address that a higher network layer uses to address a lower layer.
SM	- System Management is a network layer whose task is the administrative control of other network layers.
SMI	- System Management Interface is the point to which other network layers pass information to SM.
TDR	- Time Domain Reflectometry is a technique used in CSMA/CD type lans to determine shorts or opens in the cable.
TEST FRAME	- A special type of message frame which the remote system must sense and wrap back to the sender.
TRANCEIVER	- A connector which connects a node to the CSMA/CD type of cable such as ethernet.
XID FRAME	- A special type of frame which the remote system must sense and return with it's identification.
X.409	- An CCITT standard which decribes recommendations for a system of inter-layer data transmission formats.

1. INTRODUCTION

1.1 BACKGROUND

Lan network maintainability requires a facility to verify the operability of the various components of the network data paths with minimal effect on other lan network users. The Local Line Tester (LLT1) program is being modified to serve this function for the lan network.

1.2 BASIC PURPOSE

The program LLT1 provides the lan network operator with the ability to test various lan subcomponents with minimal disturbance to other users of the network. This test program executes in a variety of operating system environments. When it runs in a DSA environment, it will communicate with the network operator via NAIAD in network control language. When it runs in any other environment it will communicate through a user-friendly menu which appears on the system (OCL) operator's console.

1.3 OVERVIEW

LLT1 will test the lan subsytems in both an in-line and on-line manner. For the controller, adapter, and tranceiver/modem tests, LLT1 will use the entire resource which is under test. If the operator wishes to test the controller, that entire controller cannot be used by any other user. It is the operator's responsiblity to use whatever resource lock out commands this operating system environment supports to lock out other users prior to invoking LLT1. For media and remote node tests, LLT1 can co-exist on the resource being tested with other users.

1.4 BASIC OPERATION

LLT1 will go through three phases for each lan subsystem that it tests. An initiation phase, an test phase, and an termination phase. In general, LLT1 will follow the following procedure for whatever lan subsystem is under test.

1. The lead task will parse the start test command, and search the physical line directory to see if the physical line exists that is to be tested.

2. The lead task will then issue an associate local user mcl (2a00) to get a lrn for SM. With this lrn, it will do an activate local SAP request to SM by sending an iorb to LDIS using the $RQIO call.

3. The lead task will then set up an event iorb to handle unsolicited errors from SM. Finally, it will place the SM lrn, start test command parameters, and name of the link under test in a parameter list prior to spawning the test task.

4. The test task will fetch its parameter list from a pointer in its task request block. The SM lrn will be used in all iorbs to SM, while the physical line name will be used as part of the SM PDU.

5. The test task will issue a SM iorb pointing to an action request PDU to perform a test or change the state of the lan subsystem under test. After each action request, the test task will wait for an action response PDU to arrive in a buffer pointed to by a receive SM iorb.

6. The lead task will monitor the operator interface for further operator requests. If another start test is received, the lead task will spawn another test task and pass the same SM lrn to it along with its other test parameters.

7. In event of an error, the lead task will get the error status out of the SM event iorb, issue another to SM, send error text to the operator, and if the error is fatal, shut down all test tasks.

8. The test task will start disconnect procedures when a fatal error occurs, a terminate command is sent by the operator, or a preset limit is reached that was part of the original set of parameters.

9. The lead task will monitor all active test tasks. If all of the test tasks have terminated, the lead task will issue a deactivate local SAP iorb to SM and then terminate itself.

2. CONTROLLER TESTING

2.1 FUNCTION

The lan network operator will require a method of checking the LACS without taking the complete system off-line to run conventional test programs. LLT1 will give the operator a means to invoke the LACS initialization quality logic self tests without re-booting the system. LLT1 must have total control of the LACS to do this since the initialization process destroys all user processes running on the controller. Once the controller and associated adapter qlt's have finished, LLT1 will fetch the qlt results from the controller's memory and analyze them for the operator. Prior to test termination, the LACS will be restarted for normal use.

2.2 BASIC STRUCTURE AND OVERVIEW

When testing the controller, the test software consists only of a lead task group that parses operator commands and a test task for each controller being tested. Both of these reside in the system's main memory system pool. When the operator starts a controller test, the lead task will connect to SM as in section 1.4. The test task will send an action request PDU to SM to reset the controller under test. SM will general initialize the controller when it receives this command. An action response PDU will arrive when the initialize is done. LLT1 will then send an action PDU to read the portion of LACS memory containing qlt results. SM upon receiving this request, fetches the requested LACS memory and places it on the system load media as a bound unit. After the action response arrives, the test task will load this bound unit into memory and report the qlt results. After this, another action request PDU is sent to restart the normal user LACS software. Finally, the lead task deactivates its local SAP to SM and terminates.

```
       ---------                                                 -------
       ! NAIAD !<-------->--------  ----------<----------->! OCL !
       ---------                 !  !                      -------
                            -------------
                            ! LAN LEAD  !
                            ! TASK LLT1 !
                            -------------
                              !     -    !
-------  -----------  -------------   !  -------------  -----------  -------
! PDU !--! SM IORB !--! TEST TASK1!   !  ! TEST TASK2!--! SM IORB !--! PDU !
-------  -----------  -------------   !  -------------  -----------  -------
   !                                  !                                 !
-----------------------------------------------------------------------------
! LACS Driver Interface Services Software (LDIS)                            !
-----------------------------------------------------------------------------
                                        !
                    -------------------------------------------
                    ! SYSTEM MANAGEMENT LAYER SERVER          !
                    -------------------------------------------
                                        !
---------------------------------------------------------------------------
! LACS Driver Megabus Services Software                                   !
---------------------------------------------------------------------------
                                                     -
LEVEL 6                                              !
////////////////////////////// MEGABUS ////////////////////////////////////
LACS                                                 !
                                                     v
                                    -----------------------------
                                    ! QLT RESULTS LEFT IN MEMORY !
                                    ! LOCATION X'300410' TO      !
                                    ! X'3004BA'.  THESE WILL BE  !
                                    ! COPIED TO L6 MEMORY AS A   !
                                    ! BOUND UNIT BY SM           !
                                    -----------------------------
```

Figure 1
Operational Relationships between the Lan Test and the various Network Administration Layers during controller tests.

2.3 INITIATION

The first phase of of initiation is standard and follows the procedure outlined in section 1.4. Next the test task issues a SM iorb that points to a SM PDU. The PDU is an action request PDU in the format specified by the System Management Software Component Specification [1]. The purpose of sending this PDU is to make SM general initialize the controller under test. The format of the PDU is a follows;

```
[CS,Construct,Id=1] [Length=34]       Request PDU
. [CS,Construct,Id=3] [Length=82]       Action Request
. . [CS,Construct,Id=0] [Length=57]       Resource Id
. . . [CS,Construct,Id=1] [Length=55]       Layer Info
. . . . [CS,Primitive,Id=0] [Length=01]       Layer
. . . . {00}                                  system management layer
. . . . [CS,Primitive,Id=1] [Length=01]       Sublayer
. . . . {2}                                   any
. . . . [CS,Primitive,Id=2] [Length=01]       Layer Instance
. . . . {10}                                  controller 1, layer 0
. . . . [CS,Construct,Id=3] [Length=44]       Layer Internal Selector
. . . . . [PU,Construct,Id=0] [Length=42]       Sequence of Selection P
. . . . . . [CS,Primitive,Id=0] [Length=01]       Class
. . . . . . {13}                                  controller
. . . . . . [CS,Primitive,Id=1] [Length=08]       Name
. . . . . . {43}{54}{52}{4C}{30}{31}{00}{00}      {CTRL01}
. . . . . . [CS,Construct,Id=2] [Length=06]       Sequence of Object St
. . . . . . . [CS,Primitive,Id=0] [Length=01]       State
. . . . . . . {03}                                  locked
. . . . . . . [CS,Primitive,Id=1] [Length=01]       Substate
. . . . . . . {00}                                  anysubstate
. . . . . . [CS,Primitive,Id=3] [Length=04]       Type
. . . . . . {4C}{4E}{43}{54}                      {LNCT} LACS controller
. . . . . . [CS,Primitive,Id=4] [Length=01]       Venue
. . . . . . {01}                                  local
. . . . . . [CS,Primitive,Id=5] [Length=10]       Mappings
. . . . . . {00000000000000000000}                default
. . [CS,Primitive,Id=1] [Length=02]       Exchange Id
. . {xxxx}                                undefined
. . [CS,Primitive,Id=2] [Length=02]       Access Control
. . {0000}                                default
. . [CS,Construct,Id=0] [Length=13]       Private Honeywell Action
. . . [CS,Primitive,Id=0] [Length=01]       Code
. . . {05}                                  update state
. . . [CS,Construct,Id=1] [Length=08]       Honeywell Action Info
. . . . [CS,Construct,Id=1] [Length=06]       Update State Info
. . . . . [CS,Primitive,Id=0] [Length=01]       Requested State
. . . . . {03}                                  test
. . . . . [CS,Primitive,Id=1] [Length=01]       Requested Substate
. . . . . {01}                                  reset
```

Once the controller is reset, SM will complete a outstanding receive iorb with the action response PDU. This iorb must be issued by the test task prior to sending the action request.

2.4 TESTS PERFORMED

LLT1 will only analyze the results of the LACS initialization tests for this release. To accomplish this, the test task will issue a dump action request PDU. The format of this PDU is as follows;

```
[CS,Construct,Id=1] [Length=107]      Request PDU
. [CS,Construct,Id=3] [Length=105]     Action Request
. . [CS,Construct,Id=0] [Length=57]     Resource Id
. . . [CS,Construct,Id=1] [Length=55]     Layer Info
. . . . [CS,Primitive,Id=0] [Length=01]     Layer
. . . . {00}                                 system management layer
. . . . [CS,Primitive,Id=1] [Length=01]     Sublayer
. . . . {02}                                 any
. . . . [CS,Primitive,Id=2] [Length=01]     Layer Instance
. . . . {10}                                 controller 1, layer 0
. . . . [CS,Construct,Id=3] [Length=44]     Layer Internal Selector
. . . . . [PU,Construct,Id=0] [Length=42]     Sequence of Selection P.
. . . . . . . [CS,Primitive,Id=0] [Length=01]     Class
. . . . . . . {13}                                 controller
. . . . . . . [CS,Primitive,Id=1] [Length=08]     Name
. . . . . . . {43}{54}{52}{4C}{30}{31}{00}{00}     {CTRL01}
. . . . . . . [CS,Construct,Id=2] [Length=06]     Sequence of Object St
. . . . . . . . [CS,Primitive,Id=0] [Length=01]     State
. . . . . . . . {08}                                 test
. . . . . . . . [CS,Primitive,Id=1] [Length=01]     Substate
. . . . . . . . {02}                                 halted
. . . . . . . [CS,Primitive,Id=3] [Length=04]     Type
. . . . . . . {4C}{4E}{43}{54}                     {LNCT} LACS controlle
. . . . . . . [CS,Primitive,Id=4] [Length=01]     Venue
. . . . . . . {01}                                 local
. . . . . . . [CS,Primitive,Id=5] [Length=10]     Mappings
. . . . . . . {00000000000000000000}               default
. . [CS,Primitive,Id=1] [Length=02]     Exchange Id
. . {xxxx}                               undefined
. . [CS,Primitive,Id=2] [Length=02]     Access Control
. . {0000}                               default
. . [CS,Construct,Id=0] [Length=35]     Private Honeywell Action
. . . [CS,Primitive,Id=0] [Length=01]     Code
. . . {53}                                 dump
. . . [CS,Construct,Id=1] [Length=30]     Honeywell Action Info
. . . . [CS,Construct,Id=3] [Length=28]     Dump Info
. . . . . [CS,Primitive,Id=0] [Length=00]     Dump Text String
. . . . . [CS,Primitive,Id=1] [Length=12]     Bound Unit Path Name
. . . . . {>>SID>LANOLT}                       dump file name
. . . . . [CS,Primitive,Id=2] [Length=04]     Low Address
. . . . . {00300410}                           qlt results low addr
. . . . . [CS,Primitive,Id=3] [Length=04]     High Address
. . . . . {003004BA}                           qlt results high addr
```

Upon completion of the dump from the LACS, SM will complete an outstanding receive iorb with an action dump response PDU. The test task will then invoke the MOD 400 file system executive to load the dump file bound unit into the system memory. Once there, the test task can analyze the qlt results and report failing controller and adapter systems that may be too bad to even invoke the adapter in-line tests.

2.5 TERMINATION

The test task will now have to restart the LACS normal user software. To accomplish this, the test task now issues a load action request PDU whose format follows:

```
[CS,Construct,Id=1] [Length=88]      Request PDU
. [CS,Construct,Id=3] [Length=86]      Action Request
. . [CS,Construct,Id=0] [Length=57]      Resource Id
. . . [CS,Construct,Id=1] [Length=55]      Layer Info
. . . . [CS,Primitive,Id=0] [Length=01]      Layer
. . . . {00}                                 system management layer
. . . . [CS,Primitive,Id=1] [Length=01]      Sublayer
. . . . {02}                                 any
. . . . [CS,Primitive,Id=2] [Length=01]      Layer Instance
. . . . {10}                                 controller 1, layer 0
. . . . [CS,Construct,Id=3] [Length=44]      Layer Internal Selector
. . . . . [PU,Construct,Id=0] [Length=42]      Sequence of Selection Pa
. . . . . . [CS,Primitive,Id=0] [Length=01]      Class
. . . . . . {13}                                 controller
. . . . . . [CS,Primitive,Id=1] [Length=08]      Name
. . . . . . {43}{54}{52}{4C}{30}{31}{00}{00}     {CTRL01}
. . . . . . [CS,Construct,Id=2] [Length=06]      Sequence of Object Sta
. . . . . . . [CS,Primitive,Id=0] [Length=01]      State
. . . . . . . {08}                                 test
. . . . . . . [CS,Primitive,Id=1] [Length=01]      Substate
. . . . . . . {02}                                 halted
. . . . . . [CS,Primitive,Id=3] [Length=04]      Type
. . . . . . {4C}{4E}{43}{54}                     {LNCT} LACS controller
. . . . . . [CS,Primitive,Id=4] [Length=01]      Venue
. . . . . . {01}                                 local
. . . . . . [CS,Primitive,Id=5] [Length=10]      Mappings
. . . . . . {00000000000000000000}               default
. . [CS,Primitive,Id=1] [Length=02]      Exchange Id
. . {xxxx}                               undefined
. . [CS,Primitive,Id=2] [Length=02]      Access Control
. . {0000}                               default
. . [CS,Construct,Id=0] [Length=18]      Private Honeywell Action
. . . [CS,Primitive,Id=0] [Length=01]      Code
. . . {52}                                 load
. . . [CS,Construct,Id=1] [Length=13]      Honeywell Action Info
. . . . [CS,Construct,Id=2] [Length=11]      Load Info
. . . . . [CS,Primitive,Id=0] [Length=00]      Bound Unit Path Name
. . . . . [CS,Primitive,Id=1] [Length=02]      Restart Indicator
. . . . . {RS}                                 restart when loaded
. . . . . [CS,Primitive,Id=2] [Length=04]      Start Address
. . . . . {00000000}                           default
```

When the restart of normal user software has been accomplished, SM will update an outstanding SM receive iorb with an action load response PDU. After assuring that the restart did take place, the test task will terminate itself. If this was the last active test task, the lead task will shut down as outlined in section 1.4.

3. ADAPTER/MODEM/TRANCEIVER TESTING

3.1 FUNCTION

The lan network operator will require a method of checking each adapter and its associated modem or transceiver without taking the complete system off-line to run conventional test programs. LLT1 will give the operator a means to swap out the normal MAC user software to test each adapter's special hardware with a specially written test MAC. The operator will be able to individually invoke adapter hardware check, wrap data internal to the adapter chip set, wrap data external to the adapter at a special test plug, or wrap data at the modem or transceiver. In addition, the operator can run specialized tests such as collision, crc error detection, crc generation,...etc. When the operator finishes testing, LLT1 will swap the normal MAC user software back into use.

3.2 BASIC STRUCTURE AND OVERVIEW

When testing adapters, the test software consists only of a lead task group that parses operator commands and a test task for each adapter being tested. All of these processes reside in the system's main memory system pool "$$" for access to MOD 400 executive and communications data structures. When the operator starts an adapter test, the lead task will connect to SM as detailed in section 1.4. The test task will send an action request PDU with the create test field set to the normal MAC layer management interface. When MAC receives this message, it will do a procreate call to the Bridge Communications Kernel on the LACS controller to create the test MAC process. Next MAC will issue a prorun call to activate the test MAC process. The new test MAC will immediately register a mailbox with the normal MAC layer management process so it can receive messages from the normal MAC layer management interface. In figure 2, these interfaces are double lines instead of the single lines used to denote the normal user mailbox connections. The normal MAC will then inhibit interrupts from the adapter transmit and receive physical layer. It will then send a message to the test MAC process to indicate that it should set its interrupt vectors for the transmit and receive physical layer processes. Once the test MAC receives this message, it will store pointers into the common interrupt table for its interrupt processing code. Next the test MAC will initialize the chip set and associated data structures. After all this it will then allow interrupts to occur. The test MAC will now send a message to

```
          ---------                 -------------             -------
          ! NAIAD !<--------------->! LAN LEAD   !<---------->! OCL !
          ---------                 ! TASK LLT1  !            -------
                                    -------------
                                      !    ^    !
-------  ----------   -------------       !     ------------   ----------   -------
! PDU !--! SM IORB !--! TEST TASK1!       !     ! TEST TASK2!--! SM IORB !--! PDU !
-------  ----------   -------------       !     ------------   ----------   -------
   !                                      !                                  !
----------------------------------------------------------------------------------
! LACS Driver Interface Services Software (LDIS)                                 !
----------------------------------------------------------------------------------
                                              !
                          ---------------------------------------------
                          ! SYSTEM MANAGEMENT LAYER SERVER            !
                          ---------------------------------------------
                                              !
--------------------------------------------------------------------------------
! LACS Driver Megabus Services Software                                        !
--------------------------------------------------------------------------------
                                                        ^
LEVEL 6                                                 !
////////////////////////////// MEGABUS //////////////////////////////////////////
LACS                                                    !
                                                        v
--------------------------------------------------------------------------------
! LACS Interface Software                                                      !
--------------------------------------------------------------------------------
                                                           !
                           --------   -------         -------
                           ! LLC  !<-->! LMI !<------->! L M !
                           --------   -------         ! A A !
  ------------          -------------  -------        ! C N !
  !  MAC TEST !<===>!     MAC     !<-->! LMI !<------->! S A !
  ------------          -------------  -------        !   G !
  ! RCV ! XMT !           !      !                    ! S E !
  ! TST ! TST !        -------  -------               ! Y M !
  ------------         ! RCV !--! XMT !               ! S E !
     "     "           !PROC !  !PROC !               ! T N !
     "     "           -------  -------               ! E T !
     "     "              !        !                  ! M   !
----------------------------------------             -------
! Interrupt Handler for this adapter !
--------------------------------------------------------------------------------
! Generic Interrupt Handler for all adapters
--------------------------------------------------------------------------------
```

Figure 2

Operational Relationships between the Lan Test and the various Network Administration Layers when testing adapters.

the normal MAC that the test creation is complete. In turn the regular user MAC layer management interface will update it with appropriate header information and send it to SM. SM in turn will format a action response PDU out of the message and return it to the test task. The test task now sends special action request PDU's to the MAC layer via SM. Each PDU has a test information field which specifies which data wrap mode the adapter is in, the test type, and data to be sent. System management takes these PDU's and formats Bridge Communication Kernel style messages out of them. Each message is then sent to the MAC LMI. The normal MAC LM then checks to see what type of test MAC message it is. If it is a run test type, it will pass it on to the test MAC process unchanged. The test MAC takes the message and sets the adapter to the right mode. If data is be sent, it queues it onto the transmit ring and waits for a receive interrupt to occur. When this happens, it de-queues the receive buffer from the receive ring and places status information at the head of the buffer. It then takes this message and sends it to the normal MAC LM. MAC LM updates the message with common SM action response information and sends it to SM. System Management then creates a PDU from the message and places it in an outstanding SM receive iorb which the test task has set up prior to the action request. The test task then compares the status and received data with what was expected. This sequence repeats until the test task is told to shut down by the lead task, or the test task satisfies a parameter limit, or a fatal error occurs. When it is time to terminate the test, the test task sends a special terminate test action request PDU to MAC LM via SM. Upon receipt of the message, the test MAC will re-initialize the physical layer chip set, inhibit interrupts, and send a message that it is terminating. The MAC LM when it receives the message from the test MAC will set all internal data tables and place its interrupt handler address back into the common interrupt handler address table. When this task is complete, the MAC LM will issue a message to SM that the test has terminated. System management will then place the action response PDU in the outstanding receive iorb buffer that the test task is monitoring. If the test MAC termination was successful, the test task will terminate itself. The lead task will shut down according to section 1.4 if this test task was the last active one.

3.3 SYSTEM MANAGEMENT PDU FORMATS

All commands are sent to System Management by a Request PDU. LLT1 will use all Action Request and Action Response PDU's when testing lan subsystems. The action PDU's will have special test fields appended to them that are not in PDU format. Since SM does not decode the PDU past the length of test information field, but simply appends this information to the layer management interface message, it is not necessary to burden the MAC layer with PDU decode algorithims.

3.3.1 SM ACTION PDU TEST FIELDS

The test information fields will be appended to the SM Action PDU as data that the System Management process knows nothing about. This information immediately follows the test parameter primitive mentioned in the SM component specification [1].

test parameter
length of test info
test state
test instance
test status
test data

Test parameter - This field specifies what type of test command this is. When this field is set to create test(0), or terminate test(1), no other parameters follow this field. This field is in PDU primitive format. The id is 1.

Length of test info - This field is the length word of the test parameter PDU construct. The rest of the fields shown only apply to run test(2).

Test state - This field specifies what type of state the lan subsystem under test will be placed in. There are five possible states; controller to adapter(0), internal adapter data wrap(1), special test connector data wrap(2), near modem/transceiver data wrap(3), or far modem data wrap(4).

Test instance - This field specifies the test which is to be run. There are an indeterminate number of these, some of which are individual MAC specific. The values are; chip set(0), data loop(1), crc generation(2), recv crc check(3), collision detection check(4), and TDR cable break test(5).

Test status - This field is initially zero on action requests, and is only filled for action responses. Some types of status possible are; test succeeded, test failed, no data received, or time out occured.

Test data - An indeterminate number of data bytes follow of various data pateterns and sizes. Some are all zeros, some are all ones, some are alternating ones and zeros, some are ascii; and some rotate a 1 through a field of zeros or a 0 through a field of ones.

3.4 SYSTEM MANAGEMENT LAYER INTERFACE MESSAGE

The Lacs System Management process takes incoming PDU's which are part of the lan control block passed across the megabus interface and formats them as Bridge Communications Kernel message structures prior to delivering them to a specific layer management interface. Each message is generally in three parts. Part one is a common message header which the kernel uses as routing information. Part two is a common SM format for each layer. The last section is a layer specific field which SM leaves alone. This section is processed only by the specific layer it is targetted for. For the test MAC case, there is an additional field which only the test MAC can process.

3.4.1 TEST MESSAGE FORMATS AT MAC LMI

The test MAC message structure appears below as a message composed of three discrete sections. The sections are shown as "C" language structure definitions for later explanations.

```
#define LMRQST struct lmrqst
struct lmrqst {
        MSG    mh;                          /* normal message header */
        struct sm_section sm_msg;           /* SM routing section */
        struct tst_section tst_msg;         /* test specific section */
};
#define MSG struct msg
struct  msg {
        MSG    *m_fwd;                      /* next message on circular lis
        MSG    *m_bwd;                      /* last message on circular lis
        PID     m_sender;                   /* process id of sending proces
        BD     *m_bufdes;                   /* buffer descriptor */
        short   m_prio;                     /* message priority */
        short   m_type;                     /* user message type */
```

```
};
struct sm_section {
        short  class;                 /* layer type [MAC(4)] */
        char  *name[8];               /* class name */
        short  state;                 /* state of object */
        short  substate;              /* substate of object */
        char  *type[4];               /* type of object [MAC(8023)] */
        short *mappings[5];           /* default to zero for now */
        short  exchange_id;           /* value filled by SM */
        short  access_control;        /* security purpose (0000) */
        char   source;                /* source id [MAC(1)] */
        char   status_id;             /* undefined */
        short  status_size;           /* filled for response */
        short  op_code;               /* type of primitive service */
        short  action_code;           /* action wanted [test(x)] */
        short  test_paramete~;        /* test action wanted */
};
struct tst_section {
        short  test_size;             /* length of this block */
        short  test_state             /* mode of operation */
        short  test_instance;         /* type of test being run */
        short  test_status;           /* filled at response time */
        char  *test_data[test_size-3] /* data to be sent, or recved
};
```

3.5 INITIATION

Once the operator enters a start test command, the LLT1 lead task will parse it and find what physical object is to be tested. When it has a channel number for this object; the lead task will find the address of the channel table directory from the MOD 400 system control block. Looking in the channel table directory and finding it a null field for this channel, it will then back up and find the address of the controller directory from the SCB and from that the address of the physical line directory. LLT1's lead task then searches the physical line directory tables for a match with the name of the physical line object. Now the lead task can find out whether the line exists and the type of adapter present so a adapter specific series of subtests can be run. Next the lead task connects to SM as described in section 1.4. Once this is complete, a lan test task is spawned and passed the test start up information the operator gave along with the SM lrn and physical line type and name. The lead task then sets up an SM event iorb in case of an error, and then settles down to waiting for an event, an operator command, or a test task shut down. The test task now creates a SM action request PDU specifying that the test MAC be created. A pointer to the PDU is placed in the SM iorb and then it is sent to SM with a

$RQIO monitor call. The test task previous to this has issued a read SM iorb with $RQIO call. When the locked MAC layer manager gets this message in its mailbox; it checks the test parameter field to see if it is a create code. If the test MAC process already exists; or it is not a create code; a bad status is set and the message is returned to SM. Otherwise, the MAC layer manager spawns the test MAC process with a kernel "procreate" call, and then activates it with a "prorun" call. The test MAC process goes through its initiation routine to place its mailbox number in the ethernet control block and then waits for a message. The MAC LM inhibits interrupts and then does a "sendmsg" call to pass the entire test MAC message to the test MAC process. Once test MAC gets it, a new interrupt handler pointer will be placed in the MAC global structure and interrupts are re-enabled. The test MAC then gives the message back to MAC LM with success status set . MAC LM updates the SM section of the message with status and sends it back to SM as an action response PDU. Once SM has set the recv iorb status complete bit, the test task checks the test status field of the PDU and if it is zero, the test initialization is complete. If any fatal error event occurs during this process; the lead task will issue another event iorb and then tell the test task to terminate. If this is the only test task, the lead task itself will terminate once the test task is finished shutting down.

3.6 TESTS PERFORMED

The test MAC will receive a message containing the test state, test instance, test status and test data for each subtest to be performed. When the message arrives, the chip set will be initialized according to the test state field and the test instance field. Next the data section will be queued on the transmit ring of the LANCE chip set, and chip set turned on to send it. Once the chip set receives the data, it will be placed into a receive data buffer pointed to by a buffer descriptor in the receive ring of the LANCE chip set. The mac TEST process will look at the status information in the mode words of the receive ring and place appropriate status into the test status field of the message. The data in the receive buffer will then be attached to the message where the transmit data was and the total test info size field will be updated. After this, the message type will be changed from action request to action response and then sent back to the MAC LM. The chip set tests will consist of reading and writing to the various registers such as CSR0. The internal adapter data tests will consist of a sequence of tests starting with simple data loop. The other internal loop tests will be crc generation, crc failure detection; and collision detection.

Other tests for different adapters will be created as needed. The special test connector, transceiver/modem tests will be the same as above with longer data patterns since internal loop is limited to 32 data bytes. A possibility exists of implementing the TDR test to check for open or shorted lan cable connections.

3.7 TERMINATION

The test task will shut down when it reaches a parameter limit specified by the operator at creation time, or when told to do so by the lead task because of a fatal error, or because of an entry by the operator to stop testing this line. If termination comes prior to starting the test MAC process, the test task will simply issue an error and terminate itself. If termination comes later, the following events occur. First the test task will format a test disconnect PDU. It will then place an receive iorb to SM for the action response. The transmit iorb to SM will be sent with a pointer to the disconnect PDU. The test task then waits for the completed receive iorb. The user MAC LM receives this PDU from SM and simply passes it on to the test MAC process. When test MAC gets the message, it will stop all activity on the LANCE chip set and then reset it to normal mode. The test MAC will also reset all associated data structures such as the ethernet control block(ECB). Before returning the message to MAC LM, the test MAC will inhibit 68000 interrupts. After sending the action response message with status, the test MAC will terminate itself with a "Mexit" call to the kernel. When normal MAC LM gets this message, it will re-install its interrupt routine pointer "int_ethernet" into the global MAC data structures. Then it will re-enable interrupts, update the SM routing section of the message with status and send it to SM. SM will update the receive iorb and alert the test task with status complete. The test task checks the status to see if the test MAC did shut down and then terminates itself if status was good. When the lead task sees that the last active test task has shut down, it will disconnect from SM as described in section 1.4.

3.8 CHANGES TO CURRENT MAC LMI

The current MAC firmware component specification does not specify any System Management interface. An attempt will be made here to piece together a shell of a structure to show what modifications are necessary for the test MAC process.

```
char     *mtst[] = {"ETH_TST0", "ETH_TST1", "ETH_TST2", "ETH_TST3"}

ECB      *ecb
MSG       *msgptr
LMRQST   *malmptr
PCB      *procreate()
PCB      *tstpcb
extern   ethtst(),ethmain()

case MA_LM_REQ:                                  /* MAC layer management */
   malmptr=(LMRQST *) msgptr;                    /* cast message template *
   (verify SM routing info section)
   switch (malmptr->sm_msg.op_code)  {           /* check for LM types */
   case LM_GET:
        (tbd)
   case LM_SET:
        (tbd)
   case LM_ACTION:
      switch (malmptr->sm_msg.action_op)  { /* check for action type
      case UPDATE_STATE:
           (tbd)
      case CREATE:
           (tbd)
      case LIST:
           (tbd)
      case TEST:
         switch (malmptr->sm_msg.test_parameter)  { /* see if any ac
         case TEST_START:
            /* Check to see if the test MAC process for this
               line is already running.  If it is, this test
               create message is in error. */

            if (ecb->lm_tid != NULL) {
               malmptr->mh.m_prio = URGENT;
               malmptr->mh.m_type = MA_LM_RESP; /* action response */
               malmptr->mh.m_bufdes = NULL;
               malmptr->sm_msg.status_id = RUNNING;
               sendmsg (malmptr,ecb->lm_did); /* send to SM */
               break;
            }
            /* Create the test MAC process.  Then activate it
               using the process control block id in the prorun
               call.  If prorun returns with a non-zero call,
               send an error to SM that the test MAC could not
               be created. */

            tstpcb = procreate (ethtst,port,mtst[port],4,SUPER,NULL)
            if (prorun(tstpcb)) {           /* send error, tst not run
               malmptr->mh.m_prio = URGENT;
               malmptr->mh.m_type = MA_LM_RESP; /* action response *
               malmptr->mh.m_bufdes = NULL;
```

```
                    malmptr->sm_msg.status_id = NO_START;
                    sendmsg (malmptr,ecb->lm_did);     /* send to SM */
                    break;
                 }
                 /* By the time MAC LM gets here, test MAC
                     initialization is complete and mbx is in ecb. */

                 sendmsg (malmptr,ecb->lm_tid); /* send to test MAC */
                 break;
              case TEST_STOP | TEST_RUN:
                 /* For the request side of the interface, MAC will
                    verify that a test process exists, and then send
                    the message to the test MAC process. */

                 if (ecb->lm_tid == NULL)   {
                    malmptr->mh.m_prio = URGENT;
                    malmptr->mh.m_type = MA_LM_RESP;
                    malmptr->mh.m_bufdes = NULL;
                    malmptr->sm_msg.status_id = NOTEST;
                    sendmsg (malmptr,ecb->lm_did); /* error to SM */
                    break;
                  }
                  sendmsg (malmptr,ecb->lm_tid); /* relay msg to test MAC
                  break;
              }
           }
        }

     /* The following section is a special MA_LM_RESP section that will
        only apply to messages comming from the test MAC process. */

     case MA_LM_RESP:
        malmptr = (LMRQST *) msgptr; /* cast message */
        switch (malmptr->sm_msg.test_parameter)    {
        case TEST_START | TEST_RUN:
           /* For this case, the message must simply be relayed back
              to SM.                                          */

           sendmsg (malmptr,ecb->lm_did);    /* send to SM mbx */
           break;
        case TEST_STOP:
           /* For this case, the interrupts vector must be
              set back into the global MAC data structure and
              68000 interrupts must be re-enabled. */

           imask = disable();
           mac.int_proto[port] = int_ethernet;
           enable(imask);                          /* re-enable intrs */
           malmptr->sm_msg.status_id = SUCCESS;
           sendmsg (malmptr,ecb->lm_did);          /* relay msg to SM */
           break;
```

```
    }
```

3.9 TEST MAC MAIN LOOP

The test MAC process shown here consists of two parts. The first is an initialization section that is only run once at prorun time. The second is a main body that processes incoming and outgoing messages.

```
/* Initialization code */

ethtst (port)
short port;
{
   ECB         *ecb
   MBID        mbid

   ecb = (ECB *)mac.proto_data[port];   /* get addr of ecb */
   mbid = mboxcreate (0);    /* create infinite depth mailbox */
   ecb->lm_tid = mbid;   /* test data mailbox */
}
/* main body of code goes here */

for EVER {
   breceive (&msgptr, &mboxid);   /* wait for a message */
   malmptr = (LMRQST *)msgptr;    /* cast lm request structure on msg
   switch (malmptr->sm_msg.test_parameter)   {
   case TEST_START:
      /* must install own interrupt handler vector, and re-enable
         interrupts. Then return message with good test status. */

      imask = disable();
      mac.int_proto[port] = int_eth_tst;
      enable(imask);
      malmptr->sm_msg.m_type = MA_LM_RESP;
      malmptr->mh.m_prio = NORMAL;
      malmptr->mh.m_bufdes = NULL;
      malmptr->tst_msg.test_status = SUCCESS;
      sendmsg (malmptr,ecb->lm_cid);
      break;
   case TEST_STOP:
      /* For this mode, must stop io on LANCE chip set,
       re-initialize the chip set, and return message when done. */

      *ecb->am79_rap = CSR0;   /* set address register */
      *ecb->am79_rdp = STOP;   /* set csr0 to stop mode */
```

```
            ecb_reset(ecb);            /* reset ecb state */
            load_block(ecb);           /* reset tx, rx rings */
            chip_reset(ecb);            /* reset chip set to normal */
            malmptr->mh.m_type = MA_LM_RESP;
            malmptr->mh.mprio = NORMAL;
            malmptr->mh.m_bufdes = NULL;
            malmptr->tst_msg.test_status = SUCCESS;
            sendmsg (malmptr,ecb->lm_cid);
            mexit();
            break;
        case TEST_RUN:
            /* For this mode, we process all incomming frames and initializ
               the chip set in the correct mode according to the test_state
               parameter.  Next the test_instance parameter is decoded to
               find the test type.  Each test type case takes the data and
               places it in the chip set tx ring for transmission.  When fi
               the recv data is placed back on the message and it is sent t
               the normal MAC LM. */

            switch (malmptr->test_state)  {
            case CHIP_TEST:
                 (tbd)
            case INTERNAL:
                 (tbd)
            case CABLE:
                 (tbd)
            case EXTERNAL:
                 (tbd)
            }
            switch (malmptr->test_instance)   {
            case DATA_LOOP:
                 (tbd)
            case CRC_GEN:
                 (tbd)
            case CRC_CHK:
                 (tbd)
            case COLLISION:
                 (tbd)
            case TDR:
                 (tbd)
            }
        }
    }
```

4. MEDIA/REMOTE SYSTEMS TESTING

4.1 FUNCTION

The lan network operator will require a method of checking the integrity of the physical line link to each separate remote system. The operator will also require a method of verifying the configuration of the lan independent of what any individual system's lan configuration file may claim is there. The IEEE specifications have provided two means of doing this which is standard to all implementations of the standard. LLT1 will utilize these frame constructs to echo data at any remote node the operator specifies, or poll the entire lan for all possible addresses that any node may be accessed by.

4.2 BASIC STRUCTURE AND OVERVIEW

The lead task LLT1 will receive an operator command to poll the lan for all possible nodes, or loop frames at any individual remote node. LLT1 will use either the XID frame format, or the TEST frame format respectively to accomplish the task. The lead task connects to SM as described in section 1.4. After the test task is spawned, it constructs a PDU destined for LLC which specifies the frame type and gives it to SM via an SM iorb. SM will then pass the information to the LLC layer on the Lacs board which executes the given frame type. In one case, each system that gets the XID returns a message containing its own ID. In the other case, an exact copy of the test frame should arrive back at the originating node after being sent to a far node. When LLC receives these frames, it then sends the message to SM which in turn completes the receive SM iorb that the test task has set up. The test task can then use the id as part of a lan resources table for the operator, or it can check the frame's integrity for the TEST frame case.

```
            ---------          -------------          -------
            ! NAIAD !<--------->! LAN LEAD   !<---------->! OCL !
            ---------      -<-->! TASK LLT1 !<-->-        -------
                           !    -------------    !
-------  -----------  -------------  .  -------------  -----------  -------
! PDU !--! SM IORB !--! TEST TASK1!  !  ! TEST TASK2!--! SM IORB !--! PDU !
-------  -----------  -------------  !  -------------  -----------  -------
   !                                 !                                 !
--------------------------------------------------------------------------
! LACS Driver Interface Services Software (LDIS)                          !
--------------------------------------------------------------------------
                                        !
                     -------------------------------------------
                     ! SYSTEM MANAGEMENT LAYER SERVER           !
                     -------------------------------------------
                                        !
------------------------------------------------------------------------
! LACS Driver Megabus Services Software                                !
------------------------------------------------------------------------
                                                   .
LEVEL 6                                            !
/////////////////////////// MEGABUS ////////////////////////////////////
LACS                                               !
                                                   v
------------------------------------------------------------------------
! LACS Interface Software                                              !
------------------------------------------------------------------------
                                                      !
                                                   -------
                     --------    -------           ! L M !
                     ! LLC  !<-->! SMI !<---------->! A A !
                     --------    -------           ! C N !
                        !                          ! S A !
                 ------------    -------           !   G !
                 !    MAC   !<-->! SMI !<---------->! S E !
                 ------------    -------           ! Y M !
                    !      !                       ! S E !
              -------   --------                   ! T N !
              ! RCV !<->! XMT  !                   ! E T !
              ! PROC !  ! PROC !                   ! M   !
              -------   --------                   -------
                 !          !
--------------------------------------------
! Interrupt Handler for this adapter       !
-----------------------------------------------------------------------
! Generic Interrupt Handler for all adapters
-----------------------------------------------------------------------
```

Figure 3
Operational Relationships between the Lan Test and the various Network Administration Layers when testing remote systems.

4.3 LAN RESOURCES MAPPING USING XID FRAMES

The PDU which must be sent for XID's is formatted as follows;

```
[CS,Construct,Id=1] [Length=xx]     Request PDU
. [CS,Construct,Id=3] [Length=xx]     Action Request
. . [CS,Construct,Id=0] [Length=57]     Resource Id
. . . [CS,Construct,Id=1] [Length=55]     Layer Info
. . . . [CS,Primitive,Id=0] [Length=01]     Layer
. . . . {00}                                 system management layer
. . . . [CS,Primitive,Id=1] [Length=01]     Sublayer
. . . . {01}                                 LLC
. . . . [CS,Primitive,Id=2] [Length=01]     Layer Instance
. . . . {31}                                 adapter 3, layer 1
. . . . [CS,Construct,Id=3] [Length=44]     Layer Internal Selector
. . . . . [PU,Construct,Id=0] [Length=42]     Sequence of Selection Pa
. . . . . . [CS,Primitive,Id=0] [Length=01]     Class
. . . . . . {05}                                 logical line
. . . . . . [CS,Primitive,Id=1] [Length=08]     Name
. . . . . . {4E}{4F}{44}{45}{30}{31}{00}{00}     {NODE01}
. . . . . . [CS,Construct,Id=2] [Length=06]     Sequence of Object Sta
. . . . . . . [CS,Primitive,Id=0] [Length=01]     State
. . . . . . . {08}                                 test
. . . . . . . [CS,Primitive,Id=1] [Length=01]     Substate
. . . . . . . {05}                                 operational
. . . . . . [CS,Primitive,Id=3] [Length=04]     Type
. . . . . . {38}{30}{32}{32}                     {8022} LLC
. . . . . . [CS,Primitive,Id=4] [Length=01]     Venue
. . . . . . {02}                                 image
. . . . . . [CS,Primitive,Id=5] [Length=10]     Mappings
. . . . . . {00000000000000000000}               default
. . [CS,Primitive,Id=1] [Length=02]     Exchange Id
. . {xxxx}                               undefined
. . [CS,Primitive,Id=2] [Length=02]     Access Control
. . {0000}                               default
. . [CS,Construct,Id=0] [Length=xx]     Private Honeywell Action
. . . [CS,Primitive,Id=0] [Length=01]     Code
. . . {xx}                                 test
. . . [CS,Construct,Id=1] [Length=xx]     Honeywell Action Info
. . . . [CS,Construct,Id=5] [Length=xx]     Test Info
. . . . . [CS,Construct,Id=7] [Length=xx]     LLC Test Info
. . . . . . [CS,Construct,Id=2] [Length=xx]     Send XID
. . . . . . . [CS,Construct,Id=0] [Length=xx]     LLC Destination Add
. . . . . . . . [CS,Primitive,Id=0] [Length=xx]     Mac Address
. . . . . . . . {xxxxxxxxxxxx}                       6 octets
. . . . . . . . [CS,Primitive,Id=1] [Length=xx]     Lsap Id
. . . . . . . . {integer, 0=>254 even values}
. . . . . . . [CS,Primitive,Id=1] [Length=xx]     MAC Service Class
. . . . . . . {undefined}
. . . . . . . [CS,Primitive,Id=2] [Length=xx]     LSDU
. . . . . . . {data octetstring}
```

4.4 REMOTE SYSTEMS TESTING USING TEST FRAMES

The format for the PDU to do remote data loop using test frames is;

```
[CS,Construct,Id=1] [Length=xx]     Request PDU
. [CS,Construct,Id=3] [Length=xx]     Action Request
. . [CS,Construct,Id=0] [Length=57]     Resource Id
. . . [CS,Construct,Id=1] [Length=55]     Layer Info
. . . . [CS,Primitive,Id=0] [Length=01]     Layer
. . . . {00}                                 system management layer
. . . . [CS,Primitive,Id=1] [Length=01]     Sublayer
. . . . {01}                                 LLC
. . . . [CS,Primitive,Id=2] [Length=01]     Layer Instance
. . . . {31}                                 adapter 3, layer 1
. . . . [CS,Construct,Id=3] [Length=44]     Layer Internal Selector
. . . . . [PU,Construct,Id=0] [Length=42]     Sequence of Selection Pa
. . . . . . [CS,Primitive,Id=0] [Length=01]     Class
. . . . . . {05}                                 logical line
. . . . . . [CS,Primitive,Id=1] [Length=08]     Name
. . . . . . {4E}{4F}{44}{45}{30}{31}{00}{00}     {NODE01}
. . . . . . [CS,Construct,Id=2] [Length=06]     Sequence of Object Sta
. . . . . . . [CS,Primitive,Id=0] [Length=01]     State
. . . . . . . {08}                                 test
. . . . . . . [CS,Primitive,Id=1] [Length=01]     Substate
. . . . . . . {05}                                 operational
. . . . . . [CS,Primitive,Id=3] [Length=04]     Type
. . . . . . {38}{30}{32}{32}                     {8022} LLC
. . . . . . [CS,Primitive,Id=4] [Length=01]     Venue
. . . . . . {02}                                 image
. . . . . . [CS,Primitive,Id=5] [Length=10]     Mappings
. . . . . . {00000000000000000000}               default
. . [CS,Primitive,Id=1] [Length=02]     Exchange Id
. . {xxxx}                               undefined
. . [CS,Primitive,Id=2] [Length=02]     Access Control
. . {0000}                               default
. . [CS,Construct,Id=0] [Length=xx]     Private Honeywell Action
. . . [CS,Primitive,Id=0] [Length=01]     Code
. . . {xx}                                 test
. . . [CS,Construct,Id=1] [Length=xx]     Honeywell Action Info
. . . . [CS,Construct,Id=5] [Length=xx]     Test Info
. . . . . [CS,Construct,Id=7] [Length=xx]     LLC Test Info
. . . . . . [CS,Construct,Id=3] [Length=xx]     Send TEST frame
. . . . . . . [CS,Construct,Id=0] [Length=xx]     LLC Destination Addr
. . . . . . . . [CS,Primitive,Id=0] [Length=xx]     Mac Address
. . . . . . . . {xxxxxxxxxxxx}                       6 octets
. . . . . . . . [CS,Primitive,Id=1] [Length=xx]     Lsap Id
. . . . . . . . {integer, 0=>254 even values}
. . . . . . . [CS,Primitive,Id=1] [Length=xx]     MAC Service Class
. . . . . . . {undefined}
. . . . . . . [CS,Primitive,Id=2] [Length=xx]     LSDU
```

. . . . {data octetstring}