S U N Microsystems

Sun-2 Ethernet Board Diagnostic User's Document

Chad B. Rao

September 5, 1985

Revision B

Contents

1. Preface ---

The user document for Sun-2 ethernet(VMEBus and Multibus) board diagnostic is presented.

1.1. Purpose

It is the intention of this document to inform the reader how to use the Sun-2 ethernet board diagnostic. Also, to indicate how it works.

1.2. Audience

Members of any of the following departments may find this document of interest for various reasons: (1) Design Engineering, (2) Manufacturing, (3) Field Service, and (4) Documentation. Using the Sun-2 ethernet board diagnostic, the Design Engineer(s) of the Sun-2 ethernet board will be able to confirm the correctness of their design. Manufacturing and Field Service personnel will use Sun-2 ethernet board diagnostic for testing and/or troubleshooting purposes. Finally, the Documentation department will use this document as a basis for developing the User's Guide which will be shipped with the product itself.

2. Revision History

Revision A January 3, 1985 Initial release of this document.

Revision B September 5, 1985 Reorganized and seperated the commands for VME & Multibus.

3. Glossary

82586 EDLC (Ethernet Data Link Controller) -- it is an intelligent, high performance local communications controller by Intel.

MB502 -- it is an ethernet serial interface controller (encoder/decoder) by Fujitsu which is located between ethernet controller (82586) and ethernet transceiver cable. The major functions are to generate the 10MHz transmit clock for 82586, perform Manchester encoding/decoding of transmitted/received frame(s), and provide the electrical interface to the ethernet transceiver cable.

multibus -- it is a trademark of Intel corporation for standard bus structure.

VMEBus -- it is a trademark of Motorola for standard bus structure.

3Com black box -- transceiver by 3Com corporation (for external loopback test)

ethernet -- it is a trademark of Xerox corporation for local area network.

ethernet register space -- it is multibus memory space from 0x88000 to 0x88800 and contains 2K bytes of page map pointers to ethernet memory, 32 bytes of ID PROM, and control/status bytes.

ethernet memory space -- it is multibus memory space from 0x40000 to 0x80000.

ethernet channel attention (CA) -- writing a one to the CA bit in the control/status register causes the CA line of the EDLC (82586) to be asserted, which is the way the CPU (68010) catches the EDLC's attention.

ethernet board reset -- writing a one to the reset bit in the control/status register will cause the whole ethernet board to be reset. The board will remain in reset state until a zero is written to that register.

TDR -- Time Domain Relectometer.

RFD -- Receive Frame Descriptor.

TXD -- Transmit data line.

RXD -- Receive data line.

4. Introduction

This document is meant to help you understand how the Sun-2 ethernet board diagnostic works. It starts with a general description of hardware/software requirements, and moves on to a description of the user interface which includes error messages. Following is a description of the suggested testing sequence which will result in an accurate test. Also, included is a section with detailed descriptions of each module and how it works.

5. Requirements

5.1. Hardware Requirements

The minimum configuration of hardware required to run the Sun-2 ethernet board diagnostic is:

1). Working sun-2 system.
2). Sun video console (video board needed) or televideo-like terminal.
3). Boot device, local disk (disk controller needed), local tape (tape controller needed), and remote disk via ethernet (ethernet board needed).
4). 3Com black box or any transceiver (required to do external loop back test).

5.2. Software Requirements

The standard firmware (rom diagnostic/monitor) and standalone ethernet board diagnostic (ehter.diag) are needed.

6. General Information

This is a standalone Sun-2 ethernet board diagnostic package.

6.1. Hardware-related Information

The "heart" of ethernet board is an Ethernet Data Link Controller (EDLC - 82586 by Intel). It interfaces the Sun-2 ethernet board with the ethernet transceiver through an ethernet encoder/decoder (MB502 by Fujitsu) serial interface unit. Also, on the board are 256 Kbytes of dual-ported dynamic ram memory, one of whose ports is dedicated to the EDLC, the other to the Multibus (or VMEBus).

It is accessed through virtual addressing (using its own page map). The base of multibus memory space starts from system memory location 0xF00000. Ethernet memory (data) space starts from system memory location 0xF40000 to 0xF80000. Ethernet register space starts from system memory location 0xF88000 to 0xF88800.

Contents of ethernet register space are described as follows:

offset 0x000	page map word 0000
	(words 0001 - 1022)
offset 0x7FE	page map word 1023
offset 0x800	ID PROM byte 00
	(bytes 01 - 30)
offset 0x83E	ID PROM byte 32
offset 0x840	status/control register
offset 0x844	error latch register

Contents of ethernet status/control register are described as follows:

bit_15	reset
bit_14	normal/loopback(MB502)
bit_13	channel attention
bit_12	EDLC interrupt enable
bit_11	parity error interrupt enable
bit_10	unused
bit_09	parity error indicator
bit_08	EDLC interrupt indicator
bit_07	unused
bit_06	unused
bit_05	expansion enabled indicator
bit_04	on_board memory size indicator
bit_03 - 00	multibus base address (bits 19 - 16)

Contents of ethernet error latch register are described as follows:

first word

bit_15 - 08	unused
bit_07	EDLC or multibus error
bit_06	high byte indicator
bit_05 - 04	unused
bit_03 - 00	high order error address

second word

bit_15 - 00	low order error address

6.2. Software-Related Information

All I/O routines provided by the standalone libraries are linked to the ethernet board diagnostic. It is a sequencing, menu driven, and interruptible package and has parameterization capability. It is a useful tool for both engineers and technicians to troubleshoot and evaluate the operation of Sun-2 ethernet board.

7. Operating Instructions

7.1. Loading And Starting

When you turn on the system, after power-on rom diagnostics are run, the system automatically begins booting Unix. Then, break out of the boot sequence and return to the rom monitor by typing L1-a (hold down the "L-1" key while typing "a" key) on the Sun-2 console or by typing <break> key on the televideo-type terminal.

At this point, type K1 to the rom monitor to reset the memory maps to the initial state. We are now ready to boot the ethernet board diagnostic.

There are several ways of booting

1). from local disk

Assuming the diagnostic 'ether.diag' exists on your local disk in the /pub/stand (fileserver) or the /stand (standalone) directory, the ethernet diagnostic is loaded by typing

> b stand/ether.diag

2). from remote disk

Assuming the network fileserver has a partition reserved for the system under test (legitimate client) and the ethernet diagnostic exists in the /pub/stand on the fileserver, the ethernet diagnostic is loading by typing

> b stand/ether.diag

If the system under test is not a client of the fileserver where the diagnostic lives in the /pub/stand directory, the ethernet diagnostic is loading by typing

> b ec(,fileserver_host_net_#)stand/ether.diag

for example, (fileserver_host_net_# = 0x1a) indicates fileserver venus.

7.2. User Interface

After loading the ethernet diagnostic, the control of system is passed from the rom monitor to the ethernet diagnostic. First, it prints configuration messages on the screen as follows. Parts of them might be different for your system.

```
Boot: ec(0,1A,0) stand/ether.diag
Load: ec(0,1A,0) boot
Boot: ec(0,1A,0) stand/ether.diag
Size: 18940+ 7932+ 628 bytes
```

For Multibus ehernet board the following three messages will appear (for VMEBus these will not appear) :

```
Register @ 0xF88000
Memory @ 0xF40000
Memory size 0x40000
```

The following message will appear for both VMEBus and Multibus ethernet board.

My ethernet address = 1a:2b:3c:4d:5e:6f

"Register @ 0xF88000" indicates the starting address of register memory space. "memory @ 0xF40000" indicates the starting address of data memory space. "Memory size 0x40000" (256 Kbytes) is the total available data area. Then, it prints the ethernet diagnostic menu. For Multibus ethernet board the following menu will be printed:

MULTI-Bus Ethernet Board Diagnostics Rev. 1.12 85/07/17, Select test(s):

a - all tests (r,p,m,l,e,E,d)
r - Multibus register
p - Multibus page map
m - Multibus memory
l - local loopback
e - encoder loopback
E - external loopback
d - diagnose/TDR
D - dump control blocks
h - help
q - quit

Command :

For VMEBus ethernet board the following menu will be printed:

VME-Bus Ethernet Board Diagnostics Rev. 1.12 85/07/17, Select test(s):

a - all tests (l,e,E,d)
l - local loopback
e - encoder loopback
E - external loopback
d - diagnose/TDR
D - dump control blocks
h - help
q - quit

Command:

The prompt sign is shown by "Command :". After the sign is prompted, the user can issue a command and parameters separated by spaces. If the selected command is not supported by the ethernet diagnostic, the error message

no such test as (typed command)

is printed. (typed command) is whatever you have typed. Also, the menu is displayed again.

7.2.1. Special Notations

1). Separator (;)

The semicolon (;) mark is used between commands at the command line prompt. It must be isolated on the line and surrounded by spaces. It allows tests to be flexibly sequenced.

2). Default (.)

The period (.) mark is used as a place holder to indicate default values for some parameters.

3). Forever (*)

The star (*) mark is approximately equal to infinity or 0x7fffffff times for loop count parameters.

4). Null ( )

If a parameter is not supplied, then, the default values are used.

7.2.2. Command Line

The following message shows allowable commands and their parameters needed for VMEBus and Multibus ethernet Board.

```
a passcount(10)
r passcount(10)
p passcount(10)
m passcount(10) from(16) size(16)
l passcount(10) blocksize(16)
e passcount(10)
E passcount(10)
d passcount(10)
D flags
h
q
```

7.2.3. Parameters

1). passcount(10)

It is the loop count parameter which specifies how many times to run through the test, before passing control to the next test(s) or prompt.

```
default(.) = 1
forever(*) = 2147483647 in decimal
100        = 100 times in decimal
```

2). from(16)

This is the parameter which specifies the offset of the starting memory location to be tested.

default(.) = 0
forever(*) = 0
1f0 = 1f0 in hexdecimal (ie, 0xf401f0 in our case)

3). size(16)

It is the size parameter which specifies the length of the tested memory block.

default(.) = 0x4000 in hexdecimal
forever(*) = 0x4000 in hexdecimal
f00 = 0xf00 in hexdecimal

4). blocksize(16)

It is the size parameter which specifies the frame size of each transmit/receive block.

default(.) = 0x2000 in hexdecimal
forever(*) = 0x2000 in hexdecimal
1000 = 0x1000 in hexdecimal

5). flags

This is an index parameter which specifies the selected control block to be dumped.

= -1 in decimal (for all blocks)
1 = 1 in decimal (system control block)
2 = 2 in decimal (control block(s))
4 = 4 in decimal (TDR block)
8 = 8 in decimal (RFD (Receive Frame Descriptor) block)
16 = 16 in decimal (RBD (Receive Buffer Descriptor) block)
32 = 32 in decimal (STATS block)
64 = 64 in decimal (TBD (Transmit Buffer Descriptor))
128 = 128 in decimal (RXBUF (Receive Buffer(s))

7.2.4. Sequencing

You can specify several commands separated by separator mark (;) on a single command line. This is equivalent to traditional batch mode operation. The ethernet diagnostic program will fetch each command and sequentially execute them until the end of line mark is reached. This feature gives users flexibility.

For example, when the user types the following string,

command : r ; p ; m ; l ; E ; e ; d

first, the control of system is passed to the register check routine and exercises the multibus registers. It then sequentially checks page map memory and data memory and checks the transmission capability by local loopback, external loopback, and encoder loopback tests. Finally, the internal operations of 82586 and TDR are checked.

8. Message Handling

8.1. Messages For Help (h) Command

When you are confused about the format of an ethernet diagnostic command, typing "h" will help you confirm how to issue the correct command(s). After typing the "h" key, the following message is displayed for Multibus ethernet board:

```
a     all tests (r,p,m,l,e,E,d)   a passcount(10)
r     Multibus registers          r passcount(10)
p     Multibus page map           p passcount(10)
m     Multibus memory             m passcount(10) from(16) size(16)
l     local loopback              l passcount(10) blocksize(16)
e     encoder loopback            e passcount(10)
E     external loopback           E passcount(10)
d     diagnose/TDR                d passcount(10)
D     dump control blocks         D flags
h     help
q     quit
```

And for VMEBus ethernet board, the following message is printed:

```
a     all tests (l,e,E,d)         a passcount(10)
l     local loopback              l passcount(10) blocksize(16)
e     encoder loopback            e passcount(10)
E     external loopback           E passcount(10)
d     diagnose/TDR                d passcount(10)
D     dump control blocks         D flags
h     help
q     quit
```

The first column is the command itself, the second column is the explanation of the command, and the third column is the syntax of each command and parameter(s).

1). Passcount(10) is the loop count as a decimal number.

2). From(16) is the starting memory location as a hexdecimal number.

3). Size(16) is the length of the memory block to be tested in hexdecimal.

4). Blocksize(16) is the size of transmit/receive buffer(s) in hexdecimal.

5). Flags is an index indicating which block(s) should be dumped.

8.2. Messages For all tests (a) command

It prints the following message for Multibus ethernet board:

all multibus-ethernet board tests for xx passes

and for VMEBus ethernet board:

all vme-ethernet tests for xx passes

Then this command executes each test. The messages for each test are same as if they were

executed independently and are described later. For each pass if all tests are passed then, the following message will be displayed :

all tests ok, pass number = xx

otherwise, if any test is failed, (the individual test gives the corresponding error messages) the following message will be displayed:

test failed, in pass number = xx

It then jumps back to the main menu. The program can be stopped in between the passes by typing the 'q' couple of times. The program will jump back to the main menu after printing the following message:

Program Stopped

8.3. Messages For Multibus Register (r) Command

If the contents of the status register are correct, the message:

status register OK

is displayed on the screen. Otherwise the message:

status register read 0x[data]

is displayed. "[data]" is the contents of the status register. Then, the following messages are dumped.

```
expansion is disabled
rams are 64k
mbaddr = 4 (as in 0x40000)
Prom:
0x0: 0   0   0   0   0   0   0   0
0x8: 0   0   0   0   0   0   0   0
0x10:0   0   0   0   0   0   0   0
0x18:0   0   0   0   0   0   0   0
```

Mbaddr is the multibus high order address bits (bit_19 - bit_16). The dynamic rams are 64K devices (total memory is 256 Kbytes). If 256K devices are used, the maximum memory can be 1 Mbytes. With external memory the maximum memory can be expanded to 3 Mbytes. The above message will be displayed N times depending on the selected loop count. At the end of the loop, the message :

End of Multibus register pass N

is displayed on the screen. The diagnostic then jumps back to the menu and prompts for more tests.

8.4. Messages For Multibus Page Map (p) Command

It starts with a data bus check routine and prints:

Pass N Constant 0x[data]

on the screen, where "[data]" is one of the testing patterns (0, ffff, aaaa, 5555, 0101, 1010). If error(s) occurred, It prints:

Page map @ 0x[err loc] exp(0x[data]) obs(0x[data])

where "0x[err loc]" is the error address, "exp(0x[data])" is the written data (one of testing patterns) , "obs(0x[data])" is read data. It then executes an address bus check and prints:

Pass N Unique [data]

where "[data]" is one of incremental factors (1, -1, 16, -16, 256, -256). If error(s) occurred, it prints:

Page map @ 0x[err loc] exp(0x[data]) obs(0x[data])

The above sequence will loop N times until the selected loop count is reached. It then jumps back to the main menu.

8.5. Messages For Multibus Memory (m) Command

It prints:

Test 0x[starting loc] (0x[blk len]) N times

"[starting loc]" is starting tested memory location, "[blk len]" is the length of tested block. Then, it prints:

Pass N Constant 0x[data]

and executes a constant pattern test, where "[data]" is one of constant patterns (0, ffffff, aaaaaaaa, 55555555, 01010101, 10101010, 21983940). If parity error(s) occurred during the constant pattern test, the following message is displayed.

p 1 m 1 b 0 addr 0x20000 index ???

where "p 1" indicates parity error occurred, "m 1" indicates the error occurred on EDLC access or "m 0" indicates on Multibus access. "b 0" indicates the high byte was accessed and "b 1" low byte. "addr 0x20000" is the latched address at which the parity error occurred. If a memory error(s) occurred, the following message is shown.

Memory constant @ 0x[err loc] e(0x[data]) o(0x[data])

where "e(0x[data]" is the data being written and "o(0x[data])" is the data read. Then, it prints:

Pass N Unique [data]

and moves on executing the random data test, where "[data]" is the increment factor. If parity error(s) occurred during the random pattern test, the following message is displayed.

p 1 m 1 b 1 addr 0x2000 index ????
Parity unique @ 0x[err loc] e(0x[data]) o(0x[data])

If memory error(s) occurred, the following message is displayed.

Memory @ 0x[err loc] e(0x[data]) o(0x[data])

The above sequence (constant pattern test/random pattern test) loops N times until the selected loop count is reached. Then, the main menu is displayed.

8.6. Messages For Local Loopback (l) Command

It prints:

local loopback blk 0x[blk len] for N passes

and reconfigures the EDLC in internal loopback mode. If the 82586 fails to process the configure command, it prints:

localloop: config failed

and jumps back to display the user menu. Otherwise, it executes the loopback test. If error(s) occurred during test, it prints:

localloop: loopback failed at pass N

following the error messages of the loopback test which will be discussed later. Otherwise, it prints:

localloop: OK pass N of L

If the user suspends execution of the local loopback test, it prints:

localloop: stopped pass N of L

and jumps back to the main menu.

8.7. Messages For Encoder Loopback (e) Command

It prints:

encoder loopback for N passes

and reconfigures the EDLC into external loopback mode. If the 82586 fails to process the command, it prints:

encoderloop: config failed

and jumps back to the user menu. Otherwise, it executes the loopback test. If error(s) occurred during test, it prints:

encoderloop: loopback failed pass N

following the error messages of the loopback test which will be described later. Otherwise, it prints:

encoderloop: OK pass N of L

If the user suspends execution of the encoder loopback test, it prints:

encoderloop: stopped pass N

and jumps back to the main menu.

8.8. Messages For External Loopback (E) Command

It prints:

external loopback for N passes

and reconfigures the EDLC in external loopback mode. If the 82586 fails to process the command, it prints:

externalloop: config failed

and jumps back to the user menu.

Otherwise, it executes the loopback test. If error(s) occurred during test, it prints:

externalloop: loopback failed pass N

following the error messages of the loopback test which will be described later. Otherwise, it prints:

externalloop: OK pass N of L

If the user suspends execution of the external loopback test, it prints:

externalloop: stopped pass N

and jumps back to the main menu. (NOTE : For this test a closed end transceiver should be connected to the ethernet line, Otherwise the errors will be reported)

8.9. Messages For Loopback Module

Before starting the transmit/receive command, it checks the CUC, CU, and RU fields (in SCB block).

If the 82586 can't clear the CUC field, the following message is displayed.

loopback:CU begin command wait timeout

It then jumps back to the menu.

If either the CU or RU of the 82586 cannot enter the idle state, the following message is displayed.

loopback:CU/RU begin status wait timeout

It then jumps back to the menu.

If error(s) occurred during transmission, the message:

pass n:loopback check failed at 0x[err loc]

is displayed.

If a timeout occurred, the following message is displayed.

loopback:CU/RU end wait timeout

and it jumps back to the menu.

If the contents of the system control block (SCB) are wrong, then it prints one or more of the following messages.

loopback: scb CU command not NOP
loopback: scb RU command not NOP
loopback: scb CU status not IDLE
loopback: scb RU status not NO RESOURCES
loopback: scb cbl offset e(0x[cbl off]) o(0x[cbl off])
loopback: scb rfa offset e(0x[rfa off]) o(0x[rfa off])
loopback: scb errors in statistics

Both the CU and RU command fields should be in NOP state. The CU status field should be in IDLE state, and the RU status field should be in NO RESOURCES state. "e(0x[cbl off])" is an expected offset of the control block link (in SCB). "o(0x[cbl off])" is a read offset which indicates the link had been updated by the 82586 (not allowable). "e(0x[rfa off])" is an expected offset of the receive frame area in the SCB and "o(0x[rfa off])" is the read offset. "scb errors in statistics" indicates one or more of the error counters have been updated during command(s) execution as follows.

crcerrs counter - it contains the number of aligned frames discarded due to a crc error.
alnerrs counter - it contains the number of misaligned frames discarded due a to crc error.
rscerrs counter - it contains the number of good frames discarded due to a lack of resources to receive them.
ovrnerrs counter - it contains the number of frames that are known to be lost due to a lack of availability of the system bus.

If the contents of the transmit command block (CB) are wrong, it then prints one or more of the following messages.

loopback tx[i]: c bit not set
loopback tx[i]: b bit set
loopback tx[i]: ok bit not set
loopback tx[i]: abort bit set
loopback tx[i]: nocarrier bit set
loopback tx[i]: nocts bit set
loopback tx[i]: underrun bit set
loopback tx[i]: deferred bit set
loopback tx[i]: heartbeat bit set
loopback tx[i]: ? collisions
loopback tx[i]: el set in mid list
loopback tx[i]: el not set at end
loopback tx[i]: s bit set
loopback tx[i]: i bit set
loopback tx[i]: cmd wrong 0x?
loopback tx[i]: link wrong e(0x[link]) o(0x[link])
loopback tx[i]: bdptr wrong e(0x[bdptr]) o(0x[bdptr])
loopback tx[i]: ether address wrong
loopback tx[i]: type wrong e(0x[type]) o(0x[type])

The c bit is set by the 82586, indicating the completion of the command. The b bit is set and

cleared by the 82586 indicating the beginning and end of execution respectively. The ok bit is set by the 82586, indicating the command was executed without error. The abort bit is set by the 82586, indicating the command was abnormally terminated due to a CU abort command. The nocarrier bit is set by the 82586, indicating an unsuccessful transmission (transmission stopped when lack of carrier sense has been detected). The nocts bit is set by the 82586, indicating transmission was unsuccessful (stopped) due to lost of clear-to-send signal. The underrun bit is set by the 82586, indicating transmission was unsuccessful (stopped) due to DMA underrun, no data supplied by the system. The deferred bit is set by the 82586, indicating transmission was deferred. The heartbeat bit is set by the 82586, indicating the ethernet transceiver collision detect logic performs well. ? collisions is set by the 82586, indicating transmission attempt was stopped due to too many collisions, number of retries is exhausted. "el set in mid list" indicates EL is set by the 82586 in the mid of command link list. "el not set at end" indicates EL is cleared by the 82586 in the last command block. "s bit set" indicates that suspension of the CU upon completion of the current CB is not allowable. "i bit set" indicates that interrupt enable is not allowable. "cmd wrong 0x?" indicates that the command field should be 0x4.

If the contents of the transmit buffer descriptor (TBD) are wrong, it prints one or more of the following messages.

```
loopback[i]: tbd eof not set
loopback[i]:tbd blk e(0x[blk]) o(0x[blk])
loopback[i]:tbd addr e(0x[addr]) o(0x[addr])
```

"tbd eof not set" indicates eof field was cleared by the 82586 (should be 1).
"e(0x[blk])" is the expected block size and "o(0x[blk])" is the read block size which indicates the actual count field had been changed by the 82586.
"tbd addr e(0x[addr]) o(0x[addr])" indicates the pointer to data memory was changed.

If the contents of the received frame descriptor (RFD) are wrong, it prints one or more of the following messages.

```
loopback rfd[i]: c bit not set
loopback rfd[i]: b bit set
loopback rfd[i]: ok bit not set
loopback rfd[i]: crc bit set
loopback rfd[i]: align bit set
loopback rfd[i]: buffer bit set
loopback rfd[i]: overrun bit set
loopback rfd[i]: short bit set
loopback rfd[i]: el bit wrong
loopback rfd[i]: s bit set
loopback[i]:rfd link e(0x[link]) o(0x[link])
loopback[i]:rfd bdptr e(0x[bdptr]) o(0x[bdptr])
loopback[i]:rfd myaddr/source
loopback[i]:rfd myaddr/dest
loopback[i]:rfd type e(0x[type]) o(0x[type])
```

The c, b, ok, el, and s bits have the same meaning as above. The crc bit is set by the 82586, indicating a crc error in an aligned frame. The align bit is set by the 82586, indicating a crc error in a misaligned frame. The buffer bit is set by the 82586, indicating it ran out of buffer space. The overrun bit is set by the 82586, indicating a DMA overrun. The short bit is set by the 82586, indicating the frame was too short.

If the contents of the received buffer descriptor (RBD) are wrong, it prints one or more of the following messages.

loopback[i]:rdb eof not set
loopback[i]:rbd f not set
loopback[i]:rbd sz e(0x[count]) o(0x[count])
loopback[i]:rbd link e(0x[link]) o(0x[link])
loopback[i]:rbd addr e(0x[addr]) o(0x[addr])
loopback[i]:rbd size e(0x[size]) o(0x[size])
loopback[i]: data + 0x[index] e(0x[data]) o(0x[data])

The eof bit should be 1. The rbd f bit is set by the 82586 indicating the buffer has been used. "rbd sz e(0x[count]) o(0x[count])" indicates the receive byte count is different from the expected byte count. "rbd size e(0x[size]) o(0x[size])" indicates the buffer size field was changed by the 82586. "data + 0x[index] e(0x[data]) o(0x[data])" indicates the data in the transmit buffer is different from the receive buffer.

8.10. Messages For Diagnose/TDR (d) Command

First, it performs an ethernet board reset; if successful, it prints:

Pass N: Reset OK or

otherwise, it prints:

Pass N: Reset failed

If the self-test of the EDLC was successful, it prints:

Diagnose OK

otherwise, it prints:

Diagnose failed command

If the execution of the TDR command is successful, it prints:

TDR OK

Otherwise, it prints:

TDR failed command

8.11. Messages For Dump Control Block (D) Command

For some flags, the user is asked to type (self explanatory) a key to continue, so that the information will not be scrolled out of the screen. It prints one or more of the following messages. If the SCB flag is on, it prints the contents of the system control block.

SCB: stat 0x[status] cmd 0x[cmd]
cus(status) rus(status) cuc(cmd) ruc(cmd)
cbl offset 0x[cbl off]
rfa offset 0x[rfa off]

"stat 0x[status]" shows the status field and "cmd 0x[cmd]" the command field in hexdecimal.

"cus(status)" is the status of the command unit. "(status)" could be one of (idle), (suspend), (ready), (3), (4), (5), (6), or (7), but, (3), (4), (5), (6), or (7) should never occur. "rus(status)" is the status of the receive unit. "(status)" could be one of (idle), (suspend), (no resources), (3), (ready), (5), (6), (7), but, (3), (5), (6), or (7) should never occur. "cuc(cmd)" is the command in the command unit. "(cmd)" could be one of (nop), (start), (resume), (suspend), (abort), (5), (6), or (7), but, (5), (6), or(7) should never occur. "ruc(cmd)" is the command in the receive unit. "(cmd)" has the same meaning as the "cuc(cmd)" above. "0x[cbl off]" is a pointer to the first command block and "0x[rfa off]" is a pointer to the receive frame area.

If the CB flag is on, it prints the contents of the control block(s).

CB[i]: c [d] b [d] ok [d] abrt [d] stat 0x[]/[] el [d] s [d] i [d] cmd(cmd) link 0x[link]

Any field on this line has the same meaning as above, "[d]" could be either 1 - set, or 0 - reset.

If the TBD flag is on, it prints the contents of the transmit buffer descriptor block.

TXEXT: tbdptr 0x[tbdptr] addr 0x[addr] type 0x[type]

"tbdptr 0x[tbdptr]" is a pointer in hexdecimal to the next transmit buffer descriptor. "addr 0x[addr]" is a pointer in hexdecimal to the transmit buffer. "type 0x[type]" is a user defined type field.

If the TDR flag is on, it prints the contents of the TDR command block.

TDR: link [link], transceiver [tran], [open][short]time [time] 0x[time]

"link [link]" could be either "link ok" or "link BAD", "[ok]" indicates no serial link problem was identified. "transceiver [tran]" could be either "transceiver BAD" or "transceiver ok", "[ok]" indicates no transceiver link problem was identified. "[open]" could be either "CABLE OPEN, " or "" (open on the ethernet link). "[short]" could be either "CABLE SHORTED, " OR "" (short on the ethernet link).

If the RFD flag is on, it prints the contents of the receive frame descriptor block(s).

RFD[i]: c [d] b [d] ok [d] crc [d] align [d] buf [d] DMA [d] short [D]
RFD: el [d] s [d] link 0x[link] bdptr 0x[bdptr]
RFD: src [b0:b1:b2] dst [b0:b1:b2] type 0x[type]

The c [d], b [d], ok [d], crc [d], align [d], buf [d], DMA [d], short [d], el [d], and s [d] have the same meaning as above. "[d]" could be either 1 or 0. "link 0x[link]" is a pointer in hexdecimal to the next receive frame descriptor. "bdptr 0x[bdptr]" is a pointer in hexdecimal to the first receive buffer descriptor containing frame data. "src [b0:b1:b2]" and "dst [b0:b1:b2]" are ethernet source and destination addresses respectively.

If the RBD flag is on, it prints the contents of the receive buffer descriptor block(s).

RBD[i]: eof [d] f [d] count 0x[cnt] bdptr 0x[bdptr] addr 0x[addr] el [d] size 0x[size]

If the STATS flag is on, it prints the contents of error counters.

STATS: crc [crcerr] align [alignerr] resource [reserr] overrun [overr]

"crc [crcerr]" is the error counter of aligned frames discarded due to a CRC error. "align [alignerr]" is the error counter of misaligned frames discarded due to a CRC error. "resource [reserr]" is the error counter of good frames discarded due to a lack of available resources to receive them. "overrun [overr]" is the error counter of frames that are known to be lost because of a lack of system bus availability.

9. Suggested Testing Sequence

The command 'a' can be used, to execute the sequence of tests or in order to isolate error(s), it is recommended to follow the testing sequence below. For VMEBus system the tests 1, 2, and 3 does not exist. So, for VMEBus the valid tests are 4, 5, and 6 (for Multibus system all the tests are valid.).

1. Run the multibus register subtest to verify the multibus interface.
2. Run the page map subtest to exercise the static ram in order to have correct memory pointers.
3. Run the memory subtest to isolate and detect error(s) for both the data and address buses.
4. Run either the diagnose subtest or the local loopback subtest (isolate the 82586 with network) to test the internal functions and transmission capability of the 82586.
5. Run the external loopback subtest to isolate the network. (to run this test successfully a closed end transceiver box should be connected to the network line)
6. Run the encoder/decoder loopback subtest to exercise the transmission capability of the ethernet board without going through the network.

10. Description Of Modules

Ethernet communication board diagnostic program contains 12 different C language modules and several 68000 assembly modules as follows.

10.1. Module -- Ether.c

1. Routine -- main()
2. Function:
   It displays the user menu, parses command(s)/parameter(s), and dispatchs command(s) execution.
3. Input parameters:
   None.
4. Output parameters:
   None.
5. Test steps:

   a). It configures the VME or Multibus ethernet board.

   b). For Multibus it prints:

   ```
   Register @ 0xF8800
   Memory @ 0xF40000
   Memory size 0x4000
   My ethernet address = 1a:2b:3c:4d:5e:6f
   ```

   For VMEBus it prints:

   ```
   My ethernet address = 1a:2b:3c:4d:5e:6f
   ```

   c). It prints a menu on the screen and waits for a command(s)/parameter(s).

   ```
   VMEBus(or Multibus) Ethernet Board diagnostics Rev.1.12 85/07/17, Select te

   a - all tests (r,p,m,l,e,E,d)
   r - Multibus register
   p - Multibus page map
   m - Multibus memory
   l - local loopback
   e - encoder loopback
   E - external loopback
   d - diagnose/TDR
   D - dump control blocks
   h - help


   Command :
   ```

   d). It accepts command/parameter tokens from the keyboard, and parses them.

e). The selected function is executed.

f). Jump to step c). again.

6. Related functions:
gets(), tokenparse(), ether_help(), ether_vme_help(), ether_mb_all(), ether_vme_all().

10.2. Module -- Dump.c

This module contains two routines which dump the contents of the selected control block on the screen. Then the user can verify and troubleshoot the error(s) on the ethernet board.

A). Routine -- ether_dump_cmd(eregp, blockp, memsize, nametoken)

1. Function:
   It gets a parameter and executes the dump command.
2. Input parameters:

```
struct    ereg        *eregp,          /* pointer to register space */
struct    etherblock  *blockp,   /* pointer to control block */
int                   memsize,  /* dummy */
char                  **nametoken;   /* */
```

3. Return value:
   always one -- successful return.
4. Test steps:
   a). It gets the control block flag from the token buffer.
   b). It dumps the contents of the specified control block.
   c). Exit.
5. Related functions:
   eattoken(), etherdump().

B). Routine -- etherdump(blockp, level)

1. Function:
   It dumps the control block(s) based on the level parameter.
2. Input parameters:

```
struct    etherblock  *blockp,   /* pointer to control block */
int                   level;       /* specified block flag */
```

3. Return value:
   always one -- successful return.
4. Test steps:
   a). If the system control block flag is set, the contents of the system control block are dumped to the screen.
   b). If the command block flag is set, the contents of the command block are dumped to the screen.
   c). If the TDR (Time Domain Reflectometer) flag is set, the contents of the TDR block are dumped to the screen.

d). If the RFD (receiver file descriptor) flag is set, the contents of the RFD block are displayed on the screen.

e). If the RBD (Receiver Block Descriptor) flag is set, the contents of the RBD block are displayed on the screen.

f). If the STATS flag is on, the contents of the STATS block are dumped.

g). Exit.

5. Related functions:
   from_ieint(), maygetchar(), from_ieaddr().

10.3. Module -- Ether_register.c

This is the module which verifies the contents of the status/control register.

1. Routine -- ether_register(eregp, blockp, memsize, tokenname)
2. Function:
   This routine verifies the contents of the multibus status/control register, and prints the configuration of this board and contents of the prom on the screen.
3. Input parameters:

```
struct ereg        *eregp,          /* pointer to register space */
struct etherblock*blockp,    /* pointer to control block */
int                memsize,  /* dummy */
char               **tokenname;    /* */
```

4. Return value:
   0 -- successful return.
   1 -- status register error.
   2 -- statistics wrong.
5. Test steps:
   a). It gets the loop_count parameter from the token buffer.
   b). It verifies the contents of the status register.
   c). It prints the configuration of the board, for example whether expansion is enabled or disabled, the RAMs are 256k or 64k, mbaddr = 4 (as in 0x40000), etc.
   d). It prints the contents of the PROM (ID information).
   e). Jump to step b). until loop count=0.
   f). Exit.
6. Related function:
   eattoken().

10.4. Module -- Ether_pagemap.c

This is the module which exercises the page map memory (static ram).

1. Routine -- ether_pagemap(eregp, blockp, memsize, nametoken)
2. Function:
   It does an ethernet page map memory test (without parity).
3. Input parameters:

```
struct ereg        *eregp,          /* pointer to register space */
struct etherblock*blockp,    /* pointer to control block */
int                memsize,  /* dummy */
char               **nametoken;   /* */
```

4. Return value:
   0 -- successful return.
   1 -- data error(s).
   2 -- address error(s).
5. Test steps:
   a). It gets the loop_count parameter.
   b). It does a data bus and memory check by writing constant patterns in the sequence: 0, FFFF, AAAA, 5555, 0101, 1010, through whole page map array. If an error occurs, it loops at that error location forever until the "q" (suspension) key is typed.
   c). It does an address bus test by writing incremented data (delta = 1, -1, 10, -10, 100, -100) through the whole page map memory. If an error occurs, it loops at the error location forever until "q" key is typed.
   d). Jump to step b). until loop_count=0.
   e). Exit.
6. Related functions:
   eattoken(), wfill(), wcheck(), wloop(), wunique().

10.5. Module -- Ether_memory.c

This is the module which exercises the dynamic memory array.

1. Routine -- ether_memory(eregp, blockp, memsize, nametoken)
2. Function:
   It does an ethernet memory test (with parity enable).
3. Input parameters:

```
struct ereg        *eregp,          /* pointer to register space */
struct etherblock*blockp,    /* pointer to control block */
int                memsize,  /* length of tested memory */
char               **nametoken;   /* token buffer */
```

4. Return value:
   0 -- successful return.
   1 -- data error(s).
   2 -- address error(s).
5. Test steps:
   a). It gets the loop count, starting address, and length parameters of the memory to be tested.
   b). It sets page map pointer.
   c). It does a data bus and memory check by writing constant patterns in the sequence: 0, FFFFFFFF, AAAAAAAA, 55555555, 01010101, 10101010, 21983940, through the specified memory space. If an error occurs, it loops at the error address until the "q" key is typed.
   d). It does an address bus check by writing incremented data (delta = 1, -1, 10, -10, 100, -100) through the specified memory space. If an error occurs, it loops at the error location until the "q" key is typed.
   e). Jump to step c). until loop_count=0.
   f). Exit
6. Related functions:
   eattoken(), lfill(), lcheck(), lloop(), lunique().

10.6. Module -- Ether_support.c

This module provides several common routines which are used by other modules.

A). Routines -- etherca(blockp, eregp)

1. Function:
   It sends the channel attention signal to the ethernet board (actually to the 82586), and indicates commands should be executed.
2. Input parameters:

```
struct etherblock*blockp,   /* pointer to control block */
struct ereg       *eregp;         /* pointer to register space */
```

3. Return value:
   None.
4. Related functions:
   None.

B). Routine -- ether_reset(eregp, blockp, memsize)

1. Function:
   It resets the ethernet board (reset the 82586 chip, and put it into a ready state).
2. Input parameters:

```
struct etherblock*blockp,   /* pointer to control block */
struct ereg       *eregp,         /* pointer to register space */
u_long                memsize;  /* dummy */
```

3. Return value:
   o -- bad return.
   1 -- successful return.
4. Related functions:
   to_ieaddr(), to_ieoff(), setpgreg(), delay(), etherca(),

C). Routine -- etherconf(eregp, blockp, intloop, extloop, cable, fifolim, framelen)

1. Function:
   It resets and configures the ethernet controller chip (82586).
2. Input parameters:

```
struct etherblock*blockp,   /* pointer to control block */
struct ereg       *eregp,       /* pointer to register space */
int               intloop,  /* internal loopback indicator */
int               extloop,  /* external loopback indicator */
int               cable,    /* */
int               fifolim,  /* fifo limit value */
int               framelen; /* min # of bytes in a frame */
```

3. Return value:
   0 -- bad return.
   1 -- successful return.
4. Related functions:
   ether_reset(), ethercommand(), bcopy().

D). Routine -- ethercommand(eregp, blockp, cmd)

1. Function:
   It executes one of the 82586's commands (nop, individual address set up, configure, multicast address set up, transmit, TDR, dump status, diagnose).
2. Input parameters:

```
struct etherblock*blockp,   /* pointer to control block */
struct ereg       *eregp,       /* pointer to register space */
int               cmd;      /* command indicator */
```

3. Return value:
   0 -- bad return.
   1 -- successful return.
   Related functions:
   etherdump(), to_ieoff(), etherca().

E). Routine -- to_ieaddr(blockp, cp)

1. Function:
   It converts a cpu virtual address into an ethernet virtual address. For the multibus environment, it is assumed that the address is in the ethernet's memory space.
2. Input parameters:

```
struct    etherblock *blockp        /* pointer to control block */
caddr_t              cp;        /* */
```

3. Return value:
   n -- converted ethernet address.
4. Related functions:
   None.

F). Routine -- from_ieaddr(blockp, n)

1. Function:
   It converts an ethernet virtual address back to a cpu virtual address.
2. Input parameters:

```
struct etherblock*blockp,   /* pointer to control block */
ieaddr_t          n;        /* */
```

3. Return value:
   n -- converted cpu address.
4. Related functions:
   None.

G). Routine -- to_ieoff(blockp, addr)

1. Function:
   It converts a cpu virtual address into a 16-bit offset for the ethernet chip.
2. Input parameters:

```
struct etherblock*blockp,   /* pointer to control block */
caddr_                addr;     /* */
```

3. Return value:
   s -- converted 16-bit ethernet offset.
4. Related functions:
   None.

H). Routine -- to_ieint(n)

1. Function:
   It converts a cpu short integer into an EDLC short integer.
2. Input parameters:

```
short             n;    /* converted integer */
```

3. Return value:
   s -- converted integer.
4. Related functions:
   None.

10.7. Module -- Ether_diagnose.c

This module provides a diagnose command check, and TDR (Time Domain Reflectometer) command check.

The Diagnose command triggers an internal self test procedure of backoff related registers and counters. Also, it checks the exponential Backoff random number generator internal to the chip (82586).

The TDR command performs a Time domain reflectometer test on the serial link. By performing the command, the user is able to identify shorts or opens and their location. When the 82586 transmits All Ones, it triggers an internal timer. The timer measures the time elapsed from transmission start until the echo is obtained. Echo is indicated by either Collision Detect going active or a Carrier Sense signal drop.

1. Routine -- ether_diagnose(eregp, blockp, memsize, tokenname)
2. Function:
   This routine provides chip self_test and network self_test.
3. Input parameters:

```
struct    ereg        *eregp,          /* pointer to register block */
struct    etherblock *blockp,   /* pointer to control block */
int             memsize,   /* dummy */
char            **nametoken;   /* */
```

4. Return value:
   None.
5. Test steps:
   a). It gets the loop_count parameter.
   b). It does an ethernet board reset.
   c). It executes the diagnose command.
   d). Now, it put cable on.
   e). It executes the TDR command.
   f). Jump to b). until loop_count=0.
   g). Exit.
6. Related functions:
   ether_rest(), etherdump(), ethercommand, eattoken().

10.8. Module -- Ether_localloop.c

This is the module which executes the internal loopback test of the 82586 chip.

1. Routine -- ether_localloop(eregp, blockp, memsize, tokenname)

2 Function:
When the 82586 is configured in internal loopback mode, it is disconnected from the serial interface unit (transmit data to receive data and transmit clock to receive clock). Any frame transmitted is immediately received. This allows the users to check the transmission capability of the Intel 82586 chip.

3. Input parameters:

```
struct    ereg        *eregp,          /* pointer to register space */
struct    etherblock *blockp,    /* pointer to control block */
int             memsize,   /* dummy */
char            **nametoken;   /* */
```

4. Return value:
0 -- bad return.
1 -- successful return.

5. Test steps: ---

a). It gets the loop_count and size of the frame block parameters from the token buffer.

b). It configures the 82586 to be in internal loopback mode.

c). It fills the transmit buffer(s) and receive buffer(s) with constant data patterns.

d). It issues an internal loopback test by calling the loopback function.

e). It checks for any user suspension.

f). Jump to step c). until loop_count=0.

g). Exit.

6. Related functions:
eattoken(), etherconf(), lfill(), loopback(), maygetchar().

10.9. Module -- Ether_enloop.c

This is the module which executes the encoder/decoder loopback test.

1. Routine -- ether_enloop(eregp, blockp, memsize, nametoken)
2. Function:
   When the ethernet serial interface chip (MB502) is configured in internal loopback mode, the serial data is routed from TXD input, through its transmit logic (retiming and manchester encoding), through the receive logic (manchester decoding and receive clock generator) to RXD output. Under these circumstances, all of the transmit logic and receive logic, including the noise filter, is tested except for the transceiver cable output driver and input receivers.
3. Input parameters:

```
struct    ereg        *eregp,          /* pointer to register space */
struct    etherblock *blockp,   /* pointer to control block */
int             memsize,   /* dummy */
char            **nametoken;   /* */
```

4. Return value:
   0 -- bad return.
   1 -- successful return.
5. Test steps:
   a). It gets the loop_count parameter from the token buffer.
   b). It configures the ethernet serial interface controller chip (MB502) to be in internal loopback (encoder/decoder) mode.
   c). It issues an encoder/decoder loopback test by calling the loopback routine.
   d). It checks for any suspension by the user.
   e). Jump to step c). until loop_count=0.
   f). Exit.
6. Related functions:
   eattoken(), etherconf(), loopback(), maygetchar().

10.10. Module -- Ether_exloop.c

This is the module which executes the external loopback test of the 82586 chip.

1. Routine -- ether_extloop(eregp, blockp, memsize, nametoken)
2. Function:
   When the 82586 is configured in external loopback mode, it permits users to test all the external logic between the 82586 and the link itself. This allows the 82586 to check for correct operation of the carrier-sense and collision-detect signals from the transceiver for every frame transmitted.
3. Input parameters:

```
struct    ereg        *eregp,          /* pointer to register space */
struct    etherblock  *blockp,   /* pointer to control block */
int                   memsize,  /* dummy */
char                  **nametoken;   /* */
```

4. Return value:
   0 -- bad return.
   1 -- successful return.
5. Test steps:
   a). It gets the loop_count parameter from the token buffer.
   b). It configures the chip (82586) to be in external loopback mode.
   c). It does the external loopback test by calling the loopback routine.
   d). It checks for any suspension by the user.
   e). Jump to step c). until loop_count=0.
   f). Exit.
5. Related functions:
   eattoken(), etherconf(), loopback(), maygetchar().

10.11. Module -- Ether_loopback.c

This is the module which provides ether_enloop.c, ether_inloop.c, and ether_exloop.c modules to handle transmission tests.

1. Routine -- ether_loopback(eregp, blockp, blocksize)
2. Function:
   It does the common loop_back test depending on the configuration of the ethernet board. It checks the transmit and receive buffer(s) and control blocks.
3. Input parameters:

```
struct    ereg        *eregp,          /* pointer to register space */
struct    etherblock  *blockp,   /* pointer to control block */
int             blocksize;  /* frame block size */
```

4. Return value:
   0 -- bad return.
   1 -- successful return.
5. Test steps:
   a). It clears all control lists.
   b). It sets the transmit command block.
   c). It sets the transmit buffer descriptor block.
   d). It sets the receive frame descriptor block.
   e). It sets the receive buffer descriptor block.
   f). It requests channel attention.
   g). It compares the transmit buffer(s) with the receive buffer(s).
   h). It sends an acknowledge command.
   i). It checks the lists/buffers (system control block, transmit command block, transmit buffer descriptor block, receive frame descriptor block, receive buffer descriptor block).
6. Related functions:
   wfill(), to_ieoff(), bcopy(), to_ieaddr(), etherdump(), etherca(), lfill(), lcheck(), bcmp(), lcmp().

11. References

Ethernet Board Engineering Manual for the Sun-2/120 Deskside SunStation, Revision 50, August 23, 1983.

Intel 82586 Reference Manual, January, 1983.

Intel MicroSystem Components Handbook, 1984.

The Design Document of Sun-2 Ethernet Board Diagnostic, Revision A, 1984.

Sun-2/120 Ethernet Board Diagnostic User's Document, Revision A, January 3, 1985.