27-2-81
2-2-81
2-2-81
R1633
SAND80-1999
Unlimited Release

MASTER

SADIST **(The Sandia Data Index Structure)**

A Stand-Alone Data Base for Computer-Aided Design and General Use

Jerry D. Stauffer

Prepared by Sandia National Laboratories, Albuquerque, New Mexico 87185
and Livermore, California 94550 for the United States Department of Energy
under Contract DE-AC04-76DP00789

Printed November 1980

Sandia National Laboratories

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Issued by Sandia National Laboratories,
operated for the United States Department
of Energy by Sandia Corporation

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Printed in the United States of America

Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

Price: Printed Copy \$4.50; Microfiche \$3

SAND80-1999
Unlimited Release
Printed November 1980

DISCLAIMER
This book was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

SADIST
(the SANDia Data Index STtructure);

A Stand-Alone Data Base for Computer-Aided
Design and General Use

Jerry D. Stauffer
Computer-Aided Design Division 2113
Sandia National Laboratories
Albuquerque, New Mexico 87185

ABSTRACT

A file structure has been designed which fills the needs of multi-level hierarchical design of integrated circuits (ICs). Since the structure is actualized by a stand-alone FORTRAN program, it is applicable to general-purpose use in situations where the structure of the data modeled is similar to that of IC data. Though the structure itself is a FORTRAN direct-access file, its interface with user programs is a small sequential subfile accessible to FORTRAN, PASCAL, and most other languages. This facilitates linkup to systems already in use and requires minimal recoding. Backup, restore, and other data base recovery and integrity operations are automatic, but may be initiated by the user if desired.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

CONTENTS

	<u>Page</u>
I. Introduction.	5
II. Logical Structure	6
III. Physical Structure.	6
IV. Data Base Accessing Language.	10
V. Files Used by SADIST.	11
VI. Timing and Tuning	13
VII. Structure of the Backup File.	13
VIII. Creating the Data Base--Initial File Load	14
IX. DASTUB--An Interactive Superset	15
 APPENDIX A--Error Codes in SADIST.	 16
APPENDIX B--I-O Unit Names and Numbers	18
APPENDIX C--Sample Listing of Backup File.	19
APPENDIX D--Sample Listing of Work File.	21
APPENDIX E--Sample Listing of Data Base File	22
APPENDIX F--A Sample Terminal Session with DASTUB.	24
APPENDIX G--A Structured Description of the Data Base.	26
APPENDIX H--Calling Structure and Descriptions of Subroutines.	28

SADIST

(the SANDia Data Index STRUCTure)

I. Introduction

The need has long existed in computer-aided design of large-scale digital circuits for a small-scale data base intermediate between, on the one hand, the huge relational data base structures that require the continual services of a data base manager to retain coherence and, on the other, the simple sequential file, which costs so much in I/O time and in programmer resources in the attempt to stretch its capabilities.

SADIST (SANDia Data Index STRUCTure) is such a system. A FORTRAN program of approximately 1000 lines, it creates a structured FORTRAN direct-access file for the user, allowing subfiles of data to be inserted and retrieved by name. The structure modeled by SADIST is an approximation to the logical data structure conceived by the designer during the creation of an LSI circuit.

SADIST creates an artifact called a file structure. It imposes a more detailed structure onto a FORTRAN file, effectively dividing it into subsets of arbitrary length, containing one or more logical records. With a small (50 lines) calling program, it is stand-alone. Normally, however, it is simply linked to an existing FORTRAN or PASCAL program to provide I/O. When activated, SADIST searches the file structure for the requested subset and returns it to the user in the form of a small sequential file in a standard work file area. User programs must deal with this data in the normal fashion.

For example, a program which previously searched a sequential library file for a given model description was linked with SADIST. Instead of the time-consuming sequential search, one call to SADIST returned the requested model description (five lines of data out of five

hundred). Since the rest of the program was already set up to read those five lines and handle their internal format, no other changes were necessary to the user program. Less than one day sufficed to convert the library file from sequential to SADIST form.

II. Logical Structure

The SADIST file may be considered as an acyclic network in two dimensions and a tree in the third. Nodes on the structure are subfiles of an arbitrary length. Accessing by individual data field would be a degenerate single-record form of this structure.

In its incarnation as a model of a large-scale integrated circuit, the file would be structured as in Figure 1: nodes would represent subcircuits with the network operator being containment. At each node there would be a tree of data types, each containing one subfile of data (see Figure 1). "Nodes" or points on the graph are analogous to subcircuits or circuit elements in an IC data base.

III. Physical Structure

A file structure is a method of imposing additional structure on an existing file.

In this case, the file structure handled by SADIST is a FORTRAN direct-access file containing indices, record counts, and the like. SADIST effectively subdivides it into subfiles of arbitrary length and keeps a set of pointers with indexing names at the head of the file.

The record types handled by SADIST consist of node headers (Type 1), node pointers (Type 2), data headers (Type 4), data pointers (Type 3), and data records (Type 5).

**CIRCUIT DESCRIPTION -
LOGICAL STRUCTURE**

Figure 1

Node headers give the names of the nodes of the circuit-subcircuit tree. They are used for redundancy and to provide an area in which system accounting data may be held. Also, when the structure is backed up as a sequential file, the node headers allow programmer modification and checking of the file, a task that would be difficult if the only identification of data areas were record position counts (the method the data base uses internally).

Node pointers are associated with node headers. Each node header may have up to 100,000 node pointers below it and the same number of data types as well (this is an addressing limitation). A node pointer is a record containing the name of a subnode of the given node and the record-count location of the node header record for that node.

Another category of records subsidiary to a given node are data pointers. These records (containing name of data type and record count location of data header) define a third dimension of the tree whose operation is not "contains," but "is described by."

Each subfile of data (Type 5 records) is preceded by a data header (Type 4) which not only repeats the name of the data but includes a count of number of lines of data in the subfile.

All nodes (subcircuits) are described in the same fashion; a node header (with counts of open slots for writing, etc.) followed by a number of node pointers (to subcircuits contained in this circuit) and a number of data pointers (to different types of data that describe the circuit). All nodes (circuit elements) follow this pattern.

All leaves of data (subcircuit descriptions) consist of a data header (with record count) followed by a number of data records.

Node pointers point down the tree to other nodes (subcircuits) while data pointers point sideways to leaves of data (circuit descriptions) as in Figure 2.

Figure 2

IV. Data Base Accessing Language

There are eight possible actions that may be performed upon the file structure. Each is described by one verb (INSERT, DELETE, etc.).

SADIST is commonly used as a callable subroutine by programmers building additional power into user-level programs. As such, it is not interactive, but is called with a list of parameters which must be passed from the calling program. One of these parameters is the verb which may be:

INSERT: Adds a leaf (description of circuit element) to tree.
EXTRACT: Copies a leaf to the standard work area.
DELETE: Flags a leaf of data as deleted.
LOOK: Makes list of subelements of an element.
SYNONYM: Adds a name to be a synonym of another name.
BACKUP: Backs up the entire structure sequentially.
RESTORE: Restores the structure from its sequential form.
EXIT: Closes all files and exits.

The seven parameters that must be passed to SADIST from the accessing program are

- First, the name of the data base file itself (no longer than six characters).
- Second, the verb.
- Third, an integer in which SADIST may return an error flag.
- Fourth and fifth, the names of the including node and the node (including node used only when adding a new node or a synonym).
- Sixth, the name of the type of data to deal with.

Seventh, the name of the sequential work file for interface (if not given, default is DBWORK.DAT).

A sample call to SADIST from a user program might look like this:

```
CALL SADIST ('CHIP33','INSERT',IERROR,'MULTIPLEXER','FLIP-  
FLOP', 'SDL-DATA', 'DBWORK.DAT'))
```

which would request SADIST to insert the description of FLIP-FLOP that is already in the work file DBWORK.DAT into the description of CHIP33. The description is to be SDL-DATA data type and is a sub-circuit of MULTIPLEXER. If any error occurs, its identifying number (see Appendix A) is to be returned in the integer field IERROR.

V. Files Used by SADIST

SADIST itself uses only three files, one of which is transparent to the user (see Figure 3).

User data is inserted into the sequential file DBWORK.DAT, and data returned to the user by SADIST appears in the same area. This is the file that will be input to any user program.

Should a backup of the entire file be requested, a sequential version of the entire file structure is written to the sequential file CADFIL.BAK, and a restore operation will take data from CADFIL.BAK.

The actual file structure, whose name is determined by the user, is kept entirely by SADIST. The user provides a six-character name. The data structure itself is put into the user's directory with the name XXXXXX.DAT, where XXXXXX is the name provided by the user. All other names within the SADIST structure may be from one to twenty characters. Only the name of the file structure itself must be six characters.

Figure 3

VI. Timing and Tuning

On a sample file structure containing a library of 100 models (average 5 lines each), under TOPS-20 operating system on the DEC PDP-10 (with KL processor) the following times obtain:

to insert a new model	1.93 seconds
to extract a model	.54 seconds
to delete a model	.47 seconds
to look at structure	.46 seconds
to add a synonym	.45 seconds
to back up the structure	4.14 seconds
to restore the structure	9.30 seconds

These are CPU times and include the time needed for the interactive calling program DASTUB to access the file-structuring program itself.

When the file structure is originally created, ten slots are created for subnodes and five for data types for each node. On restoration, the maximum of these or 20 percent of the number of subnodes or data types already in the file is used. This default may be changed by the programmer in charge of SADIST at a given installation to reflect knowledge about the structure of the circuits or other data being stored. Higher default values will use file space but reduce the number of times that the file structure will be automatically backed up and restored to restructure index pointer space.

VII. Structure of the Backup File

Upon execution of the BACKUP command, SADIST creates a sequential version of the structure which we will call the backup file.

The backup file has a format which differs slightly from the physical format of the master structure. In particular, there are no pointers, and no Type 3 records. In addition, Type 4 and 5 records immediately follow the Type 2 records for their nodes.

All records begin with 1X, A1, 1X. Then there is a 20-character name field, 1X and 10 for all records except Type 5 which are 80 characters.

VIII. Creating the Data Base--Initial Load

Though an entire data base can, of course, be created through the use of INSERT, this will be inefficient due to the continual BACKUP/RESTORE cycles forced on SADIST by the overflow of pointer tables.

The easiest method to create a large structure at once, then, is to create a sequential file in SADIST backup format, run the interactive calling program DASTUB, and instruct SADIST to RESTORE the structure. Backup files may be created by the system utilities (COPY, etc.) and altered by the system editors. Line numbers added by editor programs should be removed before RESTOREing the structure.

To assure a properly constructed data base, the file being used as backup must be edited before RESTORE to conform to SADIST's idea of what a backup file should look like. No error messages should occur during the RESTORE. The old copy of the data base will be overwritten.

It is recommended that the sequential file in backup format be preserved, perhaps by RENAMEing, as a simple typo could create an invalid backup file that, nonetheless, would be picked up by a RESTORE resulting in an improperly structured data base. Special attention should be paid to the only decimal numbers required by backup format; the counts of numbers of lines in each subfile or leaf of data.

IX. DASTUB--An Interactive Superset

A program has been written to allow direct user interface with SADIST for the purpose of creating libraries of data for future use and for looking at the present contents of such a library or data base. That program is called DASTUB.

DASTUB asks the user two to five questions about the structure of the data base he/she wishes to create or use:

First, the name of the file structure itself (no more than six characters). This will be the name of the FORTRAN direct access file containing the file structure and also will usually be the name of the circuit or other data contained in the data base.

Second, the verb (INSERT, EXTRACT, DELETE, LOOK, SYNONYM, BACKUP, RESTORE, or EXIT).

Then DASTUB will ask for up to three more parameters depending on choice of verb. The parameters are node, containing or outer node, synonym, or type of data.

DASTUB then calls SADIST and the operation is performed. Note that BACKUP and RESTORE, if needed to restructure the data base pointers, are performed automatically in a user-transparent fashion (except for the 10-15 CPU seconds needed to perform the cycle).

DASTUB will then ask again for the operation to perform. EXIT causes file closure and allows a graceful end to the update process.

APPENDIX A

Error Codes in SADIST

There are three levels of error in SADIST/DASTUB. A Level "3" error is a disaster (such as a file not being there or the structure of the data base compromised). In the case of a Type "3" error, SADIST will print a message on the user's TTY and then stop. A Level "2" error is also fatal, but will not cause immediate printout. It is the classic "error message." If DASTUB is used, Level "2" error messages will cause a printout of their numeric value, but no alphabetic explanation.

Level "1" errors are warnings. Like Level "2" errors, they will be printed by DASTUB, but if the data base is called by another program, it is the responsibility of the calling program to handle error conditions.

** Note--First character is level: 1 is warning, 2 is error, 3 is disaster, stop run.

** Note--Second and third characters are error number.

** Note--Fourth and fifth characters are subroutine number.

ERROR LEVEL 1:

<u>Error Code</u>	<u>Message</u>
01	Circuit data base does not exist. It is created.
02	Data type not found in the given node.
03	Node has no data and no subnodes. Node deleted.
04	No data in this node at all.
05	Node not found.
06	Data type already exists in the node. Overwritten.
07	Synonym not allowed for top node (name of circuit).

ERROR LEVEL 2:

<u>Error Code</u>	<u>Message</u>
01	Work file open error.
02	Data base open error.
03	Node to insert new node into not found.

ERROR LEVEL 3:

<u>Error Code</u>	<u>Message</u>
01	Read error on data base.
02	Write error on data base.
03	Read error on work file.
04	Write error on work file.
05	Data base structure bad.
06	Backup data base structure bad.
07	Internal table overflow.

APPENDIX B

I/O Unit Names and Numbers

Logical unit number 50.

Symbolic name in program--ICERKT.

Access is RANDOM or RANDIN.

Record length is 83.

File name is CAD:SCADDB.LIB if SCADDB is given by the user,
CAD:DBEXEC.LIB if DBEXEC is given by the user, or
XXXXXX.DAT otherwise, where XXXXXX is given by user.

Logical unit number 51.

Symbolic name in program--ISEQ.

Access is SEQIN, SEQOUT, or SEQINOUT.

Record length is 80 or 83.

File name is CADFIL.BAK for BACKUP or RESTORE operations,
DBWORK.DAT in all other cases
unless overridden by the program call.

APPENDIX C

Sample Listing of Backup File

```
1  INVERT
2  LEVEL 1
4  SALOGS                      9
5  INPUT IN
5  OUTPUT OUT1 OUT2 OUT3 OUT4 OUT5 OUT6
5  C1310 OUT6 IN
5  LEVEL5 OUT5 OUT6
5  LEVEL4 OUT4 OUT5
5  LEVEL3 OUT3 OUT4
5  LEVEL2 OUT2 OUT3
5  LEVEL1 OUT1 OUT2
5  END
1  LEVEL1
2  LEVEL2
4  SALOGS                      6
5  INPUT 1
5  OUTPUT 4
5  C1310 2 1
5  LEVEL2 3 2
5  DEL 4 3 1 1
5  END
1  LEVEL2
2  LEVEL3
4  SALOGS                      6
5  INPUT 1
5  OUTPUT 4
5  C1310 2 1
5  LEVEL3 3 2
5  DEL 4 3
5  END
1  LEVEL3
```

```

2  LEVEL4
4  SALOGS                      6
5  INPUT 1
5  OUTPUT 4
5  C1310  2  1
5  LEVEL4  3  2  2  3
5  DEL  4  3  1  1
5  END
1  LEVEL4
2  LEVEL5
4  SALOGS                      6
5  INPUT 1
5  OUTPUT 4
5  C1310  2  1
5  LEVEL5  3  2  2  -3
5  DEL  4  3  -1  -2
5  END
1  LEVEL5
4  SALOGS                      6
5  INPUT 1
5  OUTPUT 4
5  C1310  2  1
5  INV 3  2
5  DEL  4  3  -1  -2
5  END

```

APPENDIX D

Sample Listing of Work File

Description of data contained in the subcircuit description just accessed; in whatever language. The work file below is the SALOGS data for Model C1130 from a master CAD library.

C1130	1		1		3		4		0		1
*		X	A	B	C						
2	3			4		5					
NOR	2			5		4		3			
END	C1130										

APPENDIX E

Sample Listing of Data Base File

1	INVERT	1	3	10	1	13	5	97	194	240
2	LEVEL1	17								
3	SALOGS	195								
1	LEVEL1	1	19	10	1	29	5			
2	LEVEL2	33								
3	SALOGS	205								
1	LEVEL2	1	35	10	1	45	5			
2	LEVEL3	49								
3	SALOGS	212								
1	LEVEL3	1	51	10	1	6	5			
2	LEVEL4	65								
3	SALOGS	219								
1	LEVEL4	1	67	10	1	77	5			
2	LEVEL5	81								
3	SALOGS	226								
1	LEVEL5	0	82	10	1	93	5			
3	SALOGS	233								
4	SALOGS	9								
5	INPUT IN									
5	OUTPUT OUT1 OUT2 OUT3 OUT4 OUT5 OUT6									
5	C1310 OUT6 IN									
5	LEVEL5 OUT5 OUT6									
5	LEVEL4 OUT4 OUT5									
5	LEVEL3 OUT3 OUT4									
5	LEVEL2 OUT2 OUT3									
5	LEVEL1 OUT1 OUT2									
5	END									
4	SALOGS	6								
5	INPUT 1									
5	OUTPUT 4									
5	C1310 2 1									
5	LEVEL3 3 2									

```

5 DEL 4 3 1 1
5 END
4 SALOGS 6
5 INPUT 1
5 OUTPUT 4
5 C1310 2 1
5 LEVEL3 3 2
5 DEL 4 3
5 END
4 SALOGS 6
5 INPUT 1
5 OUTPUT 4
5 C1310 2 1
5 LEVEL4 3 2 2 3
5 DEL 4 3 1 1
5 END
4 SALOGS 6
5 INPUT 1
5 OUTPUT 4
5 C1310 2 1
5 LEVEL5 3 2 2 -3
5 DEL 4 3 -1 -2
5 END
4 SALOGS 6
5 INPUT 1
5 OUTPUT 4
5 C1310 2 1
5 INV 3 2
5 DEL 4 3 -1 -2
5 END

```

APPENDIX F

A Sample Terminal Session with DASTUB

[PHOTO: Recording initiated Wed 16-Jul-80 10:43AM]

LINK FROM STAUFFER, TTY 34

TOPS-20 Command processor 3A(415)-1

%Structure already mounted

End of PS:COMAND.CMD.2

@

@

@DASTUB

ENTER NAME OF CIRCUIT--EXACTLY 6 CHARACTERS

INVERT

ENTER OPERATION (INSERT, EXTRACT, DELETE, EXIT, LOOK, BACKUP,
SYNONYM, OR RESTORE)

EXTRACT

ENTER NODE NAME

LEVEL5

ENTER DATA TYPE

SPICE

SADIST RETURNS ERROR NUMBER 10208

ENTER OPERATION (INSERT, EXTRACT, DELETE, EXIT, LOOK, BACKUP,
SYNONYM, OR RESTORE)

LOOK

ENTER NODE NAME

LEVEL4

ENTER OPERATION (INSERT, EXTRACT, DELETE, EXIT, LOOK, BACKUP,
SYNONYM, OR RESTORE,

EXIT

STOP

END OF EXECUTION

CPU TIME: 1.00 ELAPSED TIME: 52.95

EXIT

@TYpe (FILE) DBWORK.DAT

1 1

LEVEL5

SALOGS

@pop

[PHOTO: Recording terminated Wed 16-Jul-80 10:45AM]

APPENDIX G

A Structured Description of the Data Base

```

<data base>                := <pointer area> <data area>
<pointer area>              := <master node> {<subnode>} [0..infinity]
<data area>                 := {<data set>} [0..infinity]
<master node>               := <master header> <node body>
<master header>             := <file name> <node flags> <data flags>
                             <file flags>
<file name>                 := A6
<node flags>                := <# of node pointers> <location of next
                             node pointer slot> <# of node pointer
                             slots>
<data flags>                := <# of data pointers> <location of next
                             data pointer slot> <# of data pointer
                             slots>
<subnode>                   := <node header> <node body>
<data set>                  := <data header> <data file>
<node header>               := <node name> <node flags> <data flags>
<data header>               := <data type> <data record count>
<file flags>                := <location of next node header slot>
                             <location of last node header slot>
                             <location of next data header slot>
<node name>                 := 2A10
<data type>                 := 2A10
<data record count>         := I5
<# of node pointers>         := I5
<node body>                 := {<node pointer>} [0..infinity] {<data
                             pointer>} [0..infinity]
<node pointer>              := <node name> {<file name> 1 <null>}
                             <location of node header>
data pointer                 := <data type> {<file name> 1 <null>}
                             <location of data header>
<location of next node
  pointer slot>              := I5

```

```

<# of node pointer
  slots>                := 15
<# of data pointers>    := 15
<location of next data
  pointer slot>          := 15
<# of data pointer
  slots>                := 15
<location of next node
  header slot>          := 15
<location of last node
  header slot>          := 15
<location of next data
  header slot>          := 15
<location of node
  header>               := 15
<location of data
  header>               := 15
<null>                 :=
<data file>            := {<data record>} [1..infinity]
<data record>          := 8A10

```

APPENDIX H

Calling Structure and Descriptions of Subroutines

```
main line--|----backup
           |----restor
           |----synnym
           |----look   -----search
           |----extrac -----search
           |----dflag  -----search
           |----insert --|-----search
                       |-----leaf
                       |-----backup
                       |-----restor
```

Subroutine BACKUP: Copies a specified random-access data base file to sequential format for restructuring and to provide a sequential form of the file for portability. Deleted data from the random file and dummy slots (existing in the random file for inserts) are not copied. The sequential form of the structure is normalized, in that all the data pertaining to a node in the tree-structured graph follows the node info.

Subroutine RESTOR: Rebuilds the random CAD data base from a sequential file created by subroutine BACKUP. In the process, it inserts dummy record slots for use in inserting new data.

Subroutine SYNNYM: Inserts synonym records, allowing a given data leaf to be accessed by more than one name.

Subroutine LOOK: Provides a list of all subnodes and types of data for a given node. List appears in the work file DBWORK.DAT. Names are 20 characters and

format of the work file is one name per record, with one header line in format 2I5, giving number of pointers and number of data types.

Subroutine EXTRAC: Makes a copy of a selected leaf of data into the work file DBWORK.DAT.

Subroutine DFLAG: Flags a given data leaf as deleted, effectively removing it from the tree.

Subroutine INSERT: Handles all insertions of leaves into the data structure. If there is insufficient room in the structure to do an insertion, the file will be backed up and then restored. If the node to put data into does not exist, it is created. If the node that this node is to be a subnode of does not exist, that is a fatal error.

Subroutine SEARCH: Finds a given node in the tree.

Subroutine LEAF: Inserts a leaf of data at a specified node in the data base. The data to be inserted is found in the file DBWORK.DAT. To do this insertion, LEAF rewrites record #1 of the file so its pointers reflect the insertion of the data record, and the appropriate node header so its pointers reflect the insertion of a pointer to the data, and inserts a new data pointer record as a sub-record of the node header.

JDS:pml:5064A:02/13/81X

Distribution:

Georgia Sutherland
Hewlett-Packard
11000 Wolfe Road
Cupertino, CA 95014

A. B. C. Sampaio
3267 Sepulveda, #3
Los Angeles, CA 90034

John R. Yatchak
EG&G Idaho, Inc.
P.O. Box 1625
Idaho Falls, ID 83415

Graeme R. Boyle
Tektronix, Inc.
MS 50-454
P.O. Box 500
Beaverton, OR 97077

Sam Stevens
Hughes Aircraft
Fallbrook & Roscoe
Bldg. 270/055
Canoga Park, CA 91304

T. J. Lie
Burroughs Corp.
San Diego, CA 92127

Harm Rombeek
Manager, Systems Support
MITEL Corp.
350 Leggett Drive
P.O. Box 13089
Kanata, Ontario, Canada K2k 1X3

L. Vincent Corbin
Boeing Aerospace
P.O. Box 3999/Mailstop 84-53
Seattle, WA 98124

William I. Henry
Electronic CAD Development Manager
Boeing Aerospace
P.O. Box 3999
Seattle, WA 98124

2110	C. W. Gwyn
2113	M. E. Daniel
2113	J. M. Acken
2113	E. L. Mathey
2113	B. T. Preas
2113	J. D. Stauffer (25)
2113	M. M. Wada
2115	J. L. Jorgensen
2116	W. J. Barnard
2140	B. L. Gregory
3141	T. L. Werner (5)
3151	W. L. Garner, for DOE/TIC (3)
3154-3	R. P. Campbell, DOE/TIC (25)