010261-5-T

A Structurally Oriented Simulation System

Z. ARAN

under the direction of
Professor J. F. Meyer

July 1973

Prepared under
NASA Grant NGR23-005-463

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SYSTEMS ENGINEERING LABORATORY

THE UNIVERSITY OF MICHIGAN, ANN ARBOR

THE UNIVERSITY OF MICHIGAN

SYSTEMS ENGINEERING LABORATORY

Department of Electrical and Computer Engineering
College of Engineering

SEL Technical Report No. 70

A STRUCTURALLY ORIENTED SIMULATION SYSTEM

by

Zvi Aran

Under the direction of
Professor John F. Meyer

Programmers: J. G. Bravatto
W. E. Bully

July 1973

Prepared under

NASA Grant
NGR23-005-463

CONTENTS

A Structurally Oriented Simulation System

1. General Description

1. 1. Introduction

A Structurally Oriented Simulation System (SOSS) is a computer program, designed to be used as an experimental aid in the study of reliable systems. Basically, SOSS can simulate the structure and behavior of a discrete-time, finite-state, time-invariant system at various levels of structural definition. SOSS enables the user to simulate systems incorporating many input and state variables which cannot, generally, be solved with a pencil and pad. This is especially true when one is interested in the effects of local structural faults on the overall behavior of a large system.

The structure of a simulated system is specified as a network of sequential machines. Further, local changes in structure can be specified, which correspond to faults in the original system. This ability to "insert" faults through simulation and observe their effects on the behavior or the original system is one of the distinguishing features of SOSS in its application to the study of reliable systems. Another feature is that SOSS is not confined to the simulation of logic systems with a binary alphabet. The object of SOSS is to obtain experimental results pertaining to properties of reliable system such as fault tolerance, diagnosability and reconfigurability. Such results can lend insight to both the theory and design of reliable systems.

SOSS has been designed to run on-line on the Michigan Terminal System, (MTS), i.e., in a conversational mode via a terminal, but it can be easily modified to run on other IBM 360/370 facilities with terminal input.

1.2. Description

A general description of the systems that can be simulated by SOSS is given in Figure 1. The simulator is table driven (see section 5) and is, therefore, circuit and device independent. The combinational network is a finite acyclic network (no feedback loops are allowed). Each network node realizes a general combinational function of n variables ($n \geq 1$), that is, a function from an n-fold cartesian product of finite sets into a finite set. Such functions can be simple one or two variable switching functions or very complicated ones having as many as 255 variables, each of which can assume up to 255 values. SOSS dynamically allocates memory according to the core size required by the simulated system (limited only by the storage capability of the host computer). In order to simplify the use of SOSS and make it more intuitive each function is identified with, and is referred to by the name of the node representing it. The component machines $M_1, M_2, \ldots M_\ell$ are state machines, i.e., sequential machines having an output function equal to the identity function. These state machines provide the memory for the simulated system, and are referred to as machine nodes.

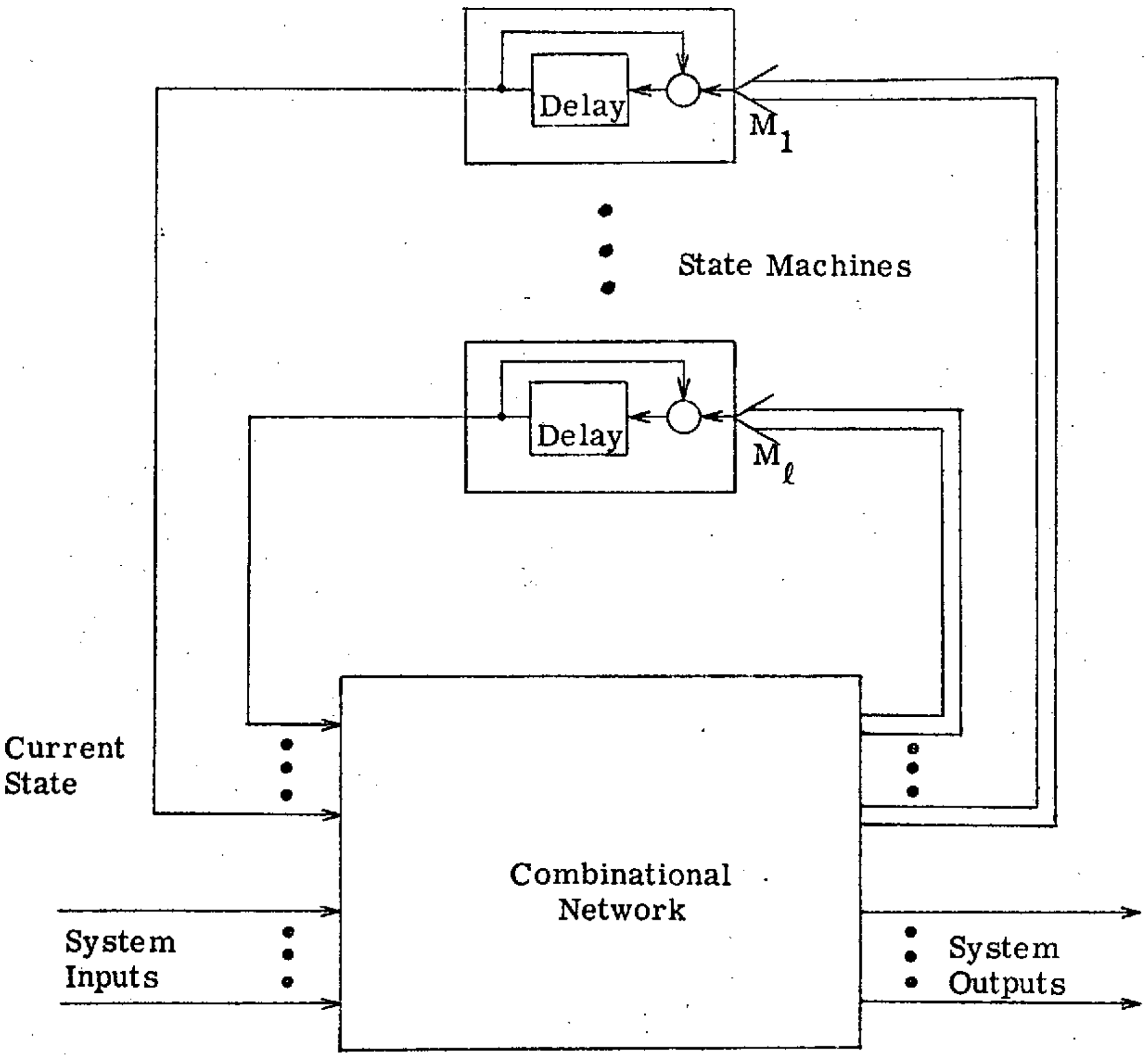
Delay
M_1
State Machines
Delay
M_ℓ
Current State
Combinational Network
System Inputs
System Outputs

Figure 1.

The simplest function a machine component can represent is a pure delay, or it can become as complicated as a combinational network function. In addition, each machine component may contain a feedback loop. The inputs to the combinational network are the system inputs and the "current state" of the system provided by the state machines. The system outputs are the outputs of any node the user chooses to monitor including the input and machine nodes.

As an example we will use an S-R flip-flop whose function table is given in Figure 2. a. In this table S and R are the control inputs, Q_t is the current state and Q_{t+1} is the next state of the flip-flop. A realization of the S-R flip-flop using AND, OR, NOT gates, and machine component Q1 is given in Figure 2. b. In this example, A1, O1 and N1 are the nodes of the combinational network. The function realized by node A1, for example, is the switching function AND. This function is assigned to node A1 while specifying the system for simulation. The transfer function δ associated with Q1 is used to detect the condition $R = S = 1$ for which the output of the S-R flip-flop is undefined. When $R \cdot S = 0$ the output of the OR gate O1 is passed to the delay. The inputs are X1(=S), X2(=R), and Q1 (as noted before the ambiguity in Q1 being both the function and the node is intentional).

The Function Table:

$$Q_{t+1} = S \vee \bar{R} \cdot Q_t$$

$$R \cdot S = 0$$

Q_t	S	R	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	undefined
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	undefined

Figure 2. a.

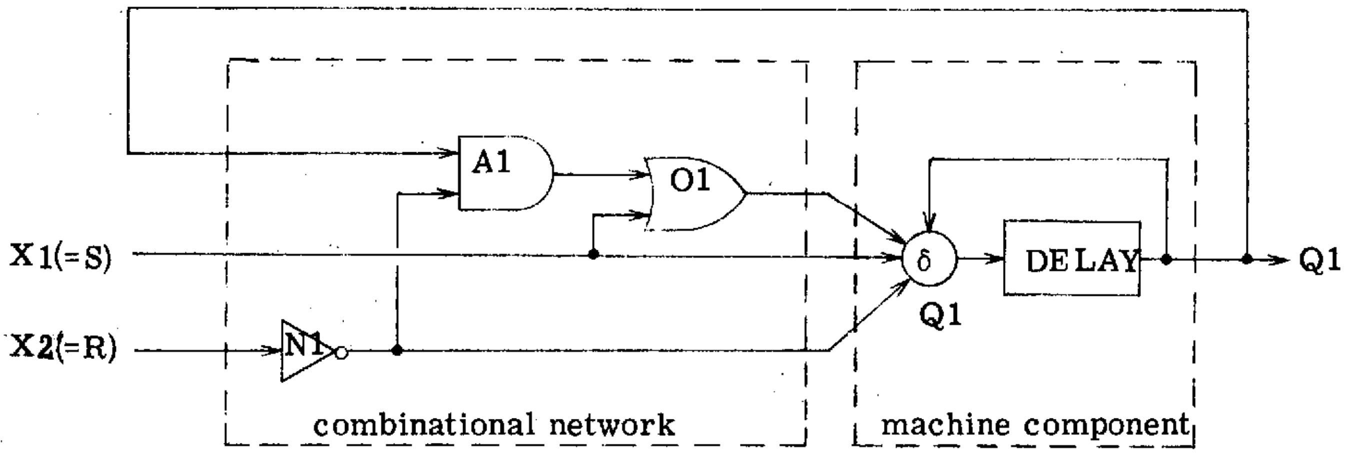

Figure 2. b.

The output set of each function may contain up to 255 distinct values. These values are represented by the integers 0 thru 254, associated with the output line of each node. The integer 255 is reserved for unspecified function values when an incompletely specified function table is assigned to some function. The cardinality of the output set associated with each node is referred to as the alphabet size (AS) of the node. In our example, Q1 has three values associated with it, (0, 1, u), hence its AS is equal to three, represented by (0, 1, 2). A1, O1, and N1 represent binary functions (0, 1) and have an AS equal to two each. Note that Q1 can assume one of three values (0, 1, 2) while the input alphabet set of A1, which represents the binary function AND, contains only the values (0, 1). SOSS will execute the simulation as long as the input to A1 assumes a legal binary value but will terminate the execution and issue an error statement if the value 2 is passed to A1 as an input.

SOSS stores the function of each node in a canonical function table. The order of the tables is the natural order defined by the order in which the input connections to each node are entered during the system specification. In the case of machine nodes the first variable in the table is the output of the machine (the feedback loop) if a function other than a delay is specified. In order to save storage, only the "value" column of each function table is stored. The number of entries in each function table is equal to the product of the AS's of all the input variables to a particular node, that is, it is the cardinality of the cross product of the alphabet sets of all the variables of that particular function.

The command language of SOSS has been designed to enable the user to employ a simple yet powerful set of commands in order to specify the structure of a system, alter structure (insert faults), and simulate the behavior of the original and altered systems.

Since structure is specified as a network of sequential machines, the user may describe a system at a variety of levels of structural refinement. He may choose to describe the detailed structure of a combinational or sequential switching network, he may describe a system as a composition of several subsystems, or he may describe only the state-transition and output functions of a system by regarding it as a single component network. The network given in Figure 2. b. , for example, can be alternatively specified in SOSS as the function F1 in Figure 2. c which realizes the function table given in Figure 2. a. Still another specification, stressing the versatility of the machine nodes, is given in Figure 2. d which containes only the machine component Q1.

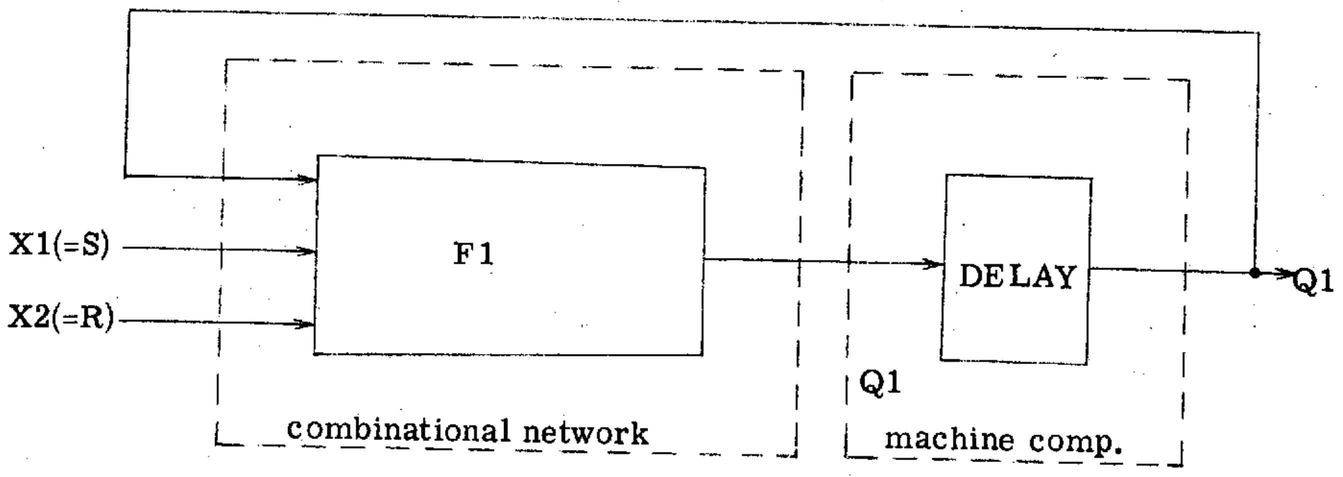

Figure 2. c.

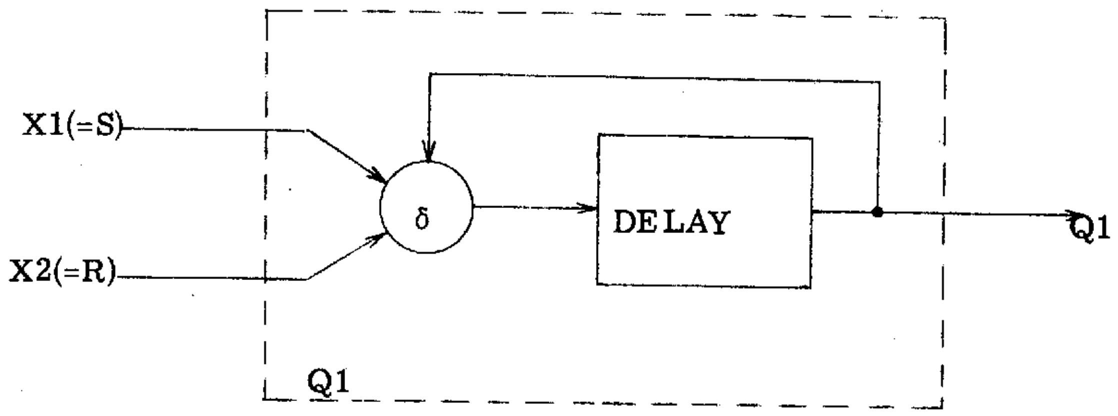

Figure 2. d.

As for fault insertion, any permanent fault, beginning with simple "stuck at" faults through functional changes in combinational and machine components may be simulated via alterations in the original structure.

1. 3. Modes of Operation

SOSS has three basic modes of operation. These are CREATE, SIMULATE and ALTER. A fourth mode is the COMMAND mode which enables the user to transfer from one mode into another. Entering data to SOSS is done in free format statements (see section 2). In these statements the letters A through O, R through V, and Y and Z are assigned to combinational network components. The letters P and Q are reserved for the component machines. The letters W and X are reserved for system inputs. The letters serve as node names. Each letter is followed by a number in the range 0-255. The number of possible system inputs is 512. This is also the number of possible machine nodes. The number of possible nodes in the combinational network is 5632. The following describes the basic modes of operation.

a) CREATE mode.

In this mode the user creates the system to be simulated. SOSS is initialized upon entry to CREATE and is ready to accept a new system. There are three types of information that the user need supply to SOSS. The first is the alphabet size (AS) of the different components, inputs, and component machines. The user then provides the functions of the combinational network components and the transition functions of the state machines. The user must also supply the inter-connections list for the whole system. Specifying the AS actually creates the node and must be done prior to assigning a function to a node or specifying its interconnections. SOSS assembles the data provided into a complete system. While data is entered, SOSS monitors the created system and issues warnings if specification errors are detected. Such error could occur in specifying functions, improper connections, feedback loops in the combinational network, repetitions of information, etc.

The system can be displayed by a display feature whenever the user wishes to inspect it while in CREATE, SIMULATE, or ALTER.

b) SIMULATE mode.

This is the mode in which the system that was created by the user is simulated. When a SIMULATE command is issued, SOSS first enteres into a test phase. In this phase it checks the created system for unspecified connections or functions, and warnings are issued if any such failure is found. Upon completion of the test phase, SOSS transfers into the SIMULATE mode and proceeds to simulate the system.

Simulation is done sequentially. First, the "current state" of the system is initialized either to an initial state given by the user or to the "next state" that resulted from the last simulation. SOSS simulates the system one clock period at a time (see section 5 for a detailed description). The user may specify an input string of any length (for each input variable) or a single symbol at a time. If, during simulation, an error in referencing the function tables is detected (e.g., a reference exceeding the table size or to an unspecified entry) simulation stops.

The output information of a simulation can be obtained by employing the display provision. All information concerning a simulation can be printed out, i.e., current state, inputs, and values of the nodes of the combinational network. Printout can be done for each clock period or following a given number of clock periods.

c) ALTER mode.

The ability to change a given system and insert faults is one of the most important features of SOSS. This is done in the ALTER mode. This mode is actually a subset of the CREATE mode in that an identical syntax is used. The difference is that the system is not initialized upon entry into the ALTER mode, but changes are made in the structure of the original system. To alter an existing system the user merely restates the required information which is to be changed.

The user may store his original system in an MTS file and make alterations on an identical copy. Both can then be run, separately, with the same input strings. In particular, in the intended application where alterations are interpreted as faults, the behavior of the faulty (altered) system can thus be compared with that of the original (fault-free) system.

2. The Command Language of the Basic System

SOSS has been designed to be versatile and as easy to operate as possible. Currently, however, only an essential subset of its designed features has been incorporated. The currently implemented version will be described in this section. The additional features, to be implemented in the future, are described in Section 3.

The following is a description of the command language of SOSS. In each command statement only the first three characters are significant. After each statement SOSS prompts the user with a special prefix character assigned to each mode of operation.

2.1. Command Mode

This is the control mode from which the user may transfer into any of the other modes. The prefix for the command mode is the "greater than" symbol (>). The statements in the command mode are listed below.

2.1.1.	CREATE	transfers the user into the CREATE mode.
2.1.2.	SIMULATE	transfers the user into the SIMULATE mode.
2.1.3.	ALTER	transfers the user into the ALTER mode.
2.1.4.	MTS	transfers the user back to the Michigan Terminal System. SOSS can be resumed where stopped by issuing $RESTART.
2.1.5.	STOP	transfers the user back to MTS. SOSS execution is terminated.
2.1.6.	HELP	lists the available commands.

2.2 CREATE Mode.

The CREATE mode is used for the specification of the system to be created. This mode is prefixed by a question mark (?). Upon entering this mode SOSS is initialized and any existing system is destroyed. The commands in this mode are given by specifying the alphabet size (AS), the connections, and the functions of all the nodes and inputs of the system. The AS of a node must be specified before any other reference to that node is made. The S-R flip-flop of Figure 2.b with its function table (Figure 2.a) will be used as an example.

2.2.1. Specifying the alphabet size (AS) is done by using the "at" sign (@). For the S-R flip-flop:

```
A1 @2
N1 @2
O1 @2
X1 @2
X2 @2
Q1 @3
```

To make writing easier the left hand side of an AS specification statement can be given as a list of nodes with the same AS, using commas to separate the node names. For example, the above specification could be entered as

```
A1, N1, O1, X1, X2 @2
Q1 @3
```

If the user wishes to create, say, ten nodes with the same AS he can use ellipses, e.g.,

A1...A10 @ 5

This statement will create ten nodes, A1 through A10, each with an AS equal to 5.

The alphabetic order within a statement is not important. The following statement, for example, will create twenty nodes, K0 through K8, C1 through C6 and Z13 through Z17, all with an AS of 3.

K0...K8,C1...C6,Z13...Z17 @ 3

The letters A-O, R-V, and Y and Z are reserved for combinational network nodes. The letters, P and Q are reserved for machine nodes and the letters W and X are reserved for system inputs. The integer following each letter can assume any value in the range 0-255. Inputs, machine nodes and combinational network nodes can be entered in the same list regardless of order, e.g.,

Q5...Q12,X1,X2,B3...B5 @ 2

2.2.2. Specification of connections between nodes is done by using the greater than symbol (>). In the case of the S-R flip-flop:

A1,X1 > O1

Q1,N1 > A1

O1,X1,X2 > Q1

X2 > N1

On the left hand side of each statement are the nodes whose outputs serve as inputs to the node(s) on the right hand side. For the purpose of specifying the connections, the left hand side of each statement must include the list of <u>all</u> the nodes which serve as input to the node on the right hand side of the statement.

If some set of inputs feeds more than a single node, the user may specify that fact as follows:

X1, P3, B5. . . B10 > Z2, Q1. . . Q6, T5

The order within each list is not important. (However, the order in which the inputs to a node are specified determines the canonical order of its function table). Each node must have at least one input connected to it, whether a system input or the output of some other node.

2.2.3. Specification of the function of a node is done by using the equal sign (=). Two types of functions currently exist in SOSS. One type is that of a SOSS predefined function. These are the logic functions for two variables AND, OR, NOT, and EXCLUSIVE-OR which can be specified for both combinational network and machine nodes. For machine nodes pure delay may be specified by using the predefined DELAY function.

The second type of function specification consists of a list of values. These values are ordered according to the "value" column of the function table. The function table must be ordered canonically, according to the order in which the connections to a particular node were specified. The function values are represented by integers in the range 0-254. The integer 255 is reserved for incompletely specified tables. An incompletely specified table is legal in SOSS as long as an unspecified entry is not referenced. It is the responsibility of the user to ascertain that such unspecified entries are not referenced. The length of a completely specified table is the same as the product of the AS's of all the inputs to that particular node.

Returning to the S-R flip-flop:

A1 = AND

N1 = NOT

O1 = OR

Alternately we could specify O1, for example, as follows:

O1 = 0, 1, 1, 1

where its table would look as follows:

A1	X1	O1
0	0	0
0	1	1
1	0	1
1	1	1

The function for node A1 would be specified as

A1 = 0, 0, 0, 1

Examining the connections to A1 we see that its inputs are Q1 and N1. Since the AS of Q1 is 3 and that of N1 is 2, the table length of A1 should be 6, and the table should look as follows:

Q1	N1	A1
0	0	0
0	1	0
1	0	0
1	1	1
2	0	undefined
2	1	undefined

The reader may note that the first four entries in the table are those of a logical AND. As long as the output of Q1 is a legal binary symbol (0, 1) SOSS will simulate the behavior of the S-R flip-flop. If the output of Q1 becomes 2 simulation will terminate. The user could alternately declare the AS of A1 to be three and the function value in the two last entries to be the integer 2, representing the undefined value. The same could be done for O1.

As for Q1, since the connections of Q1 were specified in the order O1, X1, X2 the function table for Q1 should propagate the output of O1 to the delay element whenever R•S = 0, and look as follows:

O1	X1(=S)	X2(=R)	$Q1_{t+1}$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	undefined
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	undefined

This table has eight entries and should be specified as

Q1 = 0, 0, 0, 2, 1, 1, 1, 2

The reader should recall that in the case of machine nodes, the first (leftmost) column of the table is implicitly the output of that particular machine (i.e., the feedback loop). Since the AS of Q1 is three, its table contains 24 entries and looks as follows:

Q1	O1	X1(=S)	X2(=R)	$Q1_{t+1}$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	2
0	1	0	0	1
⋮	⋮	⋮	⋮	⋮
2	1	1	0	1
2	1	1	1	2

The specification of Q1 should, therefore, be given by the statement

$$Q1 = \underbrace{0, 0, 0, 2, 1, 1, 1, 2}_{8}, \underbrace{0, 0, 0, 2, 1, 1, 1, 2}_{8}, \underbrace{0, 0, 0, 2, 1, 1, 1, 2}_{8}$$

which is exactly the earlier specification repeated three times. This happens because the feedback loop is automatically assumed if Q1 is not defined to be a pure delay. In this case the feedback is not needed at all. The best way to specify this system, then, is to pull the function of Q1 out of the component machine, and consider it as part of the combinational network. The revised network is given in Figure 3.a.

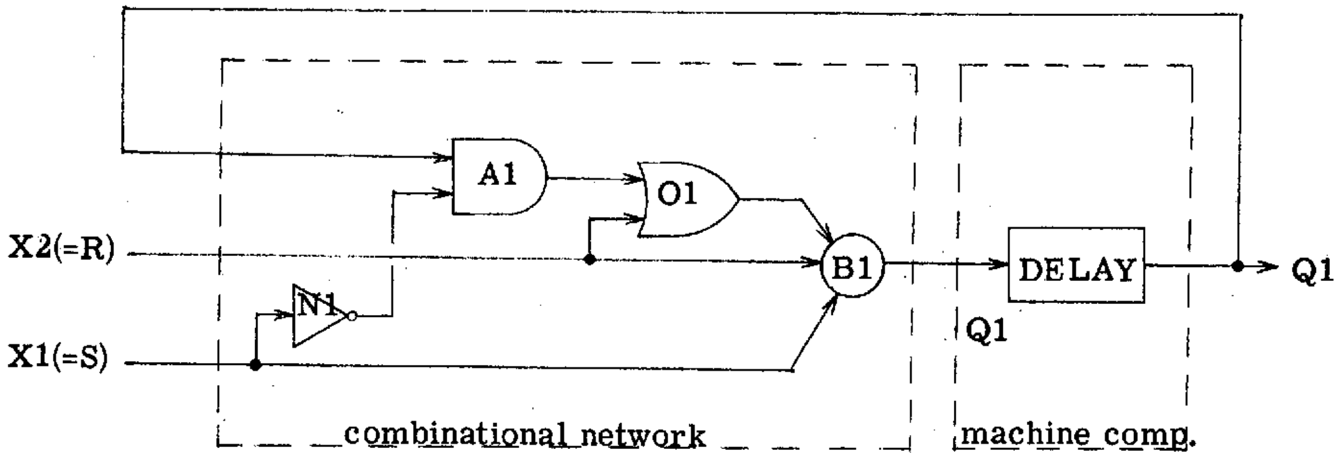

Figure 3. a.

The above example has deliberately been made elaborate in order to present some insight to the specification of a system in SOSS. To summarize the creation of a system in SOSS, the printout of the specification of the S-R flip flop, as implemented in Figure 3. a., is given in Figure 3. b.

```
>CREATE
?A1,N1,O1,X1,X2@2
?B1,Q1@3
?X2>N1
?Q1,N1>A1
?A1,X1>O1
?O1,X1,X2>B1
?B1>Q1
?A1=AND
?N1=NOT
?O1=OR
?B1=0,0,0,2,1,1,1,2
?Q1=DELAY
?END
```

Figure 3.b.: CREATE phase for the S-R flip flop of Figure 3. a.

The END statement terminates the CREATE mode and returns the user to the COMMAND mode.

Because of the algorithms used in the simulation and feedback loop checking, processor time will be saved (for large systems) if the connections are specified in the order of signal propagation, that is, from the inputs, through the combinational network to the machine nodes.

2.2.4. A display feature is incorporated in SOSS and enables the user to monitor the simulated system. The display command may be given at any time in each of the different modes. The command statement is:

DISPLAY $_\wedge$ parameter-list $_\wedge$ node-list

where $_\wedge$ represents one or more blanks.

SOSS will then display the parameters given in the parameter-list for each of the nodes specified in the node-list. The nodes in the node-list are separated by commas. The parameter-list may be any combination, in any order (separated by commas) of the following:

AS	display the alphabet size
FUNCTION	display the function
INPUTS	display the nodes serving as inputs to each of the specified nodes
OUTPUTS	display the nodes for which a specified node serves as input
VALUE	display the functional value assumed by the node during the simulation of the last clock period. When first created, all the machine component functions assume the value zero unless specifically otherwise defined.

If for some node in the node-list the requested parameter has not yet been defined, SOSS will declare it to be UNDEFINED. In the case of INPUTS into the X or W nodes (system inputs) the written message will read NONE. If the user wishes to examine all the parameters he can issue the command

DISPLAY ∧ ALL ∧ node-list

This can be useful during the CREATE phase where the user can thus verify that a node is completely specified. The user can also inspect the size of the system he has created (in MTS pages) by issuing the statement

DISPLAY ∧ SIZE.

As an example, the system specified in Figure 3.b is displayed in Figure 3.c.

```
?DIS FUNCTION B1,O1
 NODE   FUNCTION
 B1       F=0,0,0,?,1,1,1,?
 O1       F=OR
?DIS INPUTS A1,B1,N1,O1,X1,X2,Q1
 NODE   INPUTS
 A1       I=Q1,N1
 B1       I=O1,X1,X2
 N1       I=X2
 O1       I=A1,X1
 X1       I=UNDEF.
 X2       I=UNDEF.
 Q1       I=B1
?DIS OUTPUTS A1,X2,Q1
 NODE   OUTPUTS
 A1       O=O1
 X2       O=N1,B1
 Q1       O=A1
?DIS ALL A1
 NODE OAS VAL  FUNCTION  INPUTS   OUTPUTS
 A1    2    0    F=AND   I=Q1,N1   O=O1
?DIS OAS Q1,N1 *
 NODE OAS
 Q1    3
 N1    2
```

NOT REPRODUCIBLE

Figure 3. c. DISPLAY-ing the created system for for the flip flop of Figure 3. a.

* The OAS has been replaced by the AS specification.

2. 3. SIMULATE Mode.

In this mode the user-created system can be simulated. To enter this mode the statement

SIMULATE

is issued while in the COMMAND mode. Upon entry to the SIMULATE mode, a TEST phase is automatically initiated which tests the system for unspecified connections or functions. If no such failure exists SOSS prints out on the terminal the following statements:

```
k   INPUT NODES
l   MACHINE NODES
m   LOGIC NODES
```

where k, l, and m are the numbers of the corresponding nodes specified. This serves as a check for the user. SOSS then enters the SIMULATE mode. The prefix for this mode is an asterisk (*). As in CREATE, the statement END returns the user to the COMMAND mode, and the DISPLAY command (section 2. 2. 4) can be issued at any time. The following commands are available in SIMULATE mode:

2. 3. 1 STATE ∧ list of values for all the machine nodes.

This command allows the user to define the initial "current state" of the system to be simulated. This is done by specifying the values for all the machine nodes, separated by commas. Each specified value must be a legal symbol of the alphabet set of the corresponding machine node. For example, in a system that has three machine nodes

Q0, Q2, Q5, the statement

STATE 3, 1, 2

assigns the value 3 to node Q0, the value 1 to node Q2 and the value 2 to node Q5. A STATE statement may be issued prior to each simulation, otherwise, the "current state" will be defined by the outputs of the machine nodes determined at the last simulation. Initially the "current state" defaults to zero for all the machine nodes.

2.3.2 STEP $_\wedge$ n

This statement causes a simulation of n consecutive clock periods, or until a logical end-of-file (signifying the end of the input data) is encountered. If the simulated system has k inputs, for example, then an input sequence must be specified as follows (where the equal (=) sign is a prompting character issued by SOSS):

$$=\underbrace{0, 5, 2, \ldots, 1}_{\text{k symbols}}$$

SOSS proceeds to execute one clock period and prompts the user for another input sequence. This is repeated for n steps. The user may then examine the system using DISPLAY.

2.3.3 CONTINUE $_\wedge$ n

This command is similar to the STEP command, again SOSS executes the simulation one step at a time (returning to the user for more input each step) until time period n is reached. An example of a simulation of the S-R flip-flop in Figure 3.a is given in Figure 3.d.

```
>SIMULATE
    2 INPUT NODES.
    1 MACHINE NODES.
    4 LOGIC NODES.
*DIS VAL Q1
 NODE VAL
 Q1    0
*STEP 1
=0,0
*DIS VAL X1,X2,Q1
 NODE VAL
 X1    0
 X2    0
 Q1    0
*STEP 1
=0,1
*DIS VAL X1,X2,Q1
 NODE VAL
 X1    0
 X2    1
 Q1    0
*STEP 1
=1,0
*DIS VAL X1,X2,Q1
 NODE VAL
 X1    1
 X2    0
 Q1    1
*STEP 2
=0,0
=1,0
*DIS VAL X1,X2,Q1
 NODE VAL
 X1    1
 X2    0
 Q1    1
*STEP 1
=0,1
*DIS VAL X1,X2,Q1
 NODE VAL
 X1    0
 X2    1
 Q1    0
*STEP 1
=1,1
*DIS VAL X1,X2,Q1
 NODE VAL
 X1    1
 X2    1
 Q1    2
```

Figure 3. d. Simulation of the S-R flip flop.

```
*STE 1  COMMENT: SOSS IS GOING TO DECLARE THE INPUT TO NODE Q1 ILLEGAL
=0,0
 Q1 COMPUTED INDEX FALLS OUTSIDE TABLE.
*STATE 1  COMMENT: THE STATE OF MACHINE NODE Q1 MUST BE REDEFINED
*DIS VAL Q1
 NODE VAL
 Q1    1
*STE 1
=0,1
*DIS VAL X1,X2,Q1
 NODE VAL
 X1    0
 X2    1
 Q1    0
*END
```

NOT REPRODUCIBLE

Figure 3. d. Simulation of the S-R flip flop (continued).

2. 4. ALTER Mode.

The ALTER mode is the mode which enables the user to modify the structure of a given system and "insert" faults. Essentially it is part of the CREATE mode. The difference is that transferring into the ALTER mode does not destroy the existing system. In this mode nodes, connections, and functions can be added to the current system or altered as the user pleases.

It is the responsibility of the user to keep track of the changes and make sure that the altered system is legal. He may check this by issuing the END statement and transferring into the SIMULATE mode. This will initiate the TEST phase and he can then transfer back to ALTER.

A summary of the basic version of SOSS is given below. A($\wedge$) represents one or more blanks. Each statement is terminated by a carriage-return. Parameters in brackets are optional.

SOSS (basic system)

I. Command Mode: (prefix: >)

Commands available:

CRE(ATE)	to transfer into CREATE mode
ALT(ER)	to transfer into ALTER mode
SIM(ULATE)	to transfer into SIMULATE mode
HEL(P)	to list available commands
MTS	to return to MTS control, SOSS may be $RESTART-ed
STO(P)	to stop SOSS. Control returns to MTS and the current system is destroyed.

II. Create mode: (prefix: ?)

Commands available:

"AS" specifications, e.g., A1@2

"connection" specifications, e.g., A1 > B1

"function" specifications, e.g., A1 = 0, 1, 1, 0

DIS(PLAY) ∧ ALL ∧ node-list

DIS(PLAY) ∧ [VAL] [,AS] [,FUN] [,INP] [,OUT] ∧ node-list
The nodes in the node-list must be separated by commas.

DIS(PLAY) ∧ SIZE   to display size of current system

END terminates the CREATE mode. Returns the user to the COMMAND mode.

III. Simulate mode: (prefix: * for commands, = for input)

Commands available:

STA(TE) $\wedge$ list of	values for all the machine nodes to define the initial state of the system. Defaults to the "next-state" of the last simulation or to zero when first transferring into SIMULATE mode.
STE(P) $\wedge$ n	to simulate n consecutive clock cycles.
CON(TINUE) $\wedge$ n	to simulate starting from the currect clock period until clock period n. SOSS returns to the user for a new input set after each step.
DIS(PLAY)	same as in CREATE mode.
END	terminates SIMULATE mode. Returns the user to the COMMAND mode.

IV. ALTER mode.

Same as CREATE mode with the exception that it does not initialize a new system.

Hitting the ATTENTION or BREAK key terminates the current operation and returns the user to the current mode.

3. Future Features to be Incorporated in SOSS.

The current implementation of SOSS is a basic one in the following sense:

i) the display system is very convenient for creating a system but is awkward for displaying the results of simulations,

ii) an ability to store a permanent copy of the simulated system in an MTS file is desired,

iii) more versatile function specification capabilities are desired,

iv) an ability to store input specifications for consecutive clock period simulations is required.

These and additional features are listed below. They are part of the design of SOSS and should be implemented in the future.

3. 1. COMMAND Mode.

3. 1. 1. Line-Continuation character.

The logical record length in MTS is 256 characters. This limits the length of the function tables that can be specified. Since the values in the specifications are separated by commas, the maximum function table for AS $\leq$ 10 (0-9) is 128. A line continuation character (+) will eliminate this problem.

3. 1. 2. TERSE.

To shorten the messages issued by SOSS for the well versed user. This will decrease the terminal time costs.

3.2. CREATE Mode.

3.2.1. DELETE $_\wedge$ node-list

Currently, the outputs of a specified node can be disconnected, but the node cannot be deleted. In order to disconnect a node, the user must redefine the input connections to all the nodes which were fed by this particular node, eliminating this node as an input to some other node but not from the node list. The DELETE command will enable the user to delete nodes from the system.

3.2.2. TEST

To enable the user to test his created system for unspecified connections and undefined functions. This is currently done, automatically, only when the user transfers into the SIMULATE mode.

3.2.3. Additional function specification capabilities are proposed.

3.2.3.1. Additions to the system predefined functions include:

i) NAND - logic $\overline{\text{AND}}$.

ii) NOR - logic $\overline{\text{OR}}$.

iii) CONSTANT - a node with a single output value.

iv) CLOCK - to provide a clock for the user's system.

v) COUNTER (modulo n) - to provide an internal COUNTER node.

vi) RANDOM (modulo n) - to generate a random integer in the range 0-n.

vii) MAX - to determine the maximal number in a given set.

viii) MIN - to determine the minimal number in a given set.

ix) SUM (modulo n) - to determine the sum (mod n) of a given set of inputs.

x) PRODUCT (modulo n) - to determine the product (mod n) of a given set of inputs.

4. An Example

In this section an example of an autonomous machine is given. The transition graph of this machine is given in Figure 4. a.* This machine generates all binary sequences of length two. It is an example of a machine that tolerates all errors of length 1. An implementation of it is given in Figure 4. b. The printout of the simulation is given. First the system is created and displayed, then its behavior is simulated. At each state, a clock period (or more) is simulated and the resulting "next state" displayed.

*J. F. Meyer, "Sequential behavior and its inherent tolerance to memory faults," Proc. of the 5th Hawaii International Conference on System Sciences, Jan. 1972, pp. 476-478.

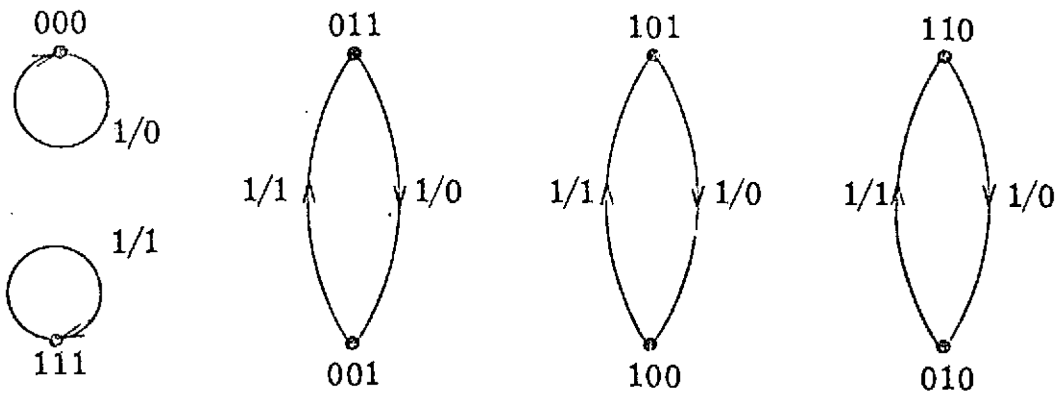

Figure 4. a.

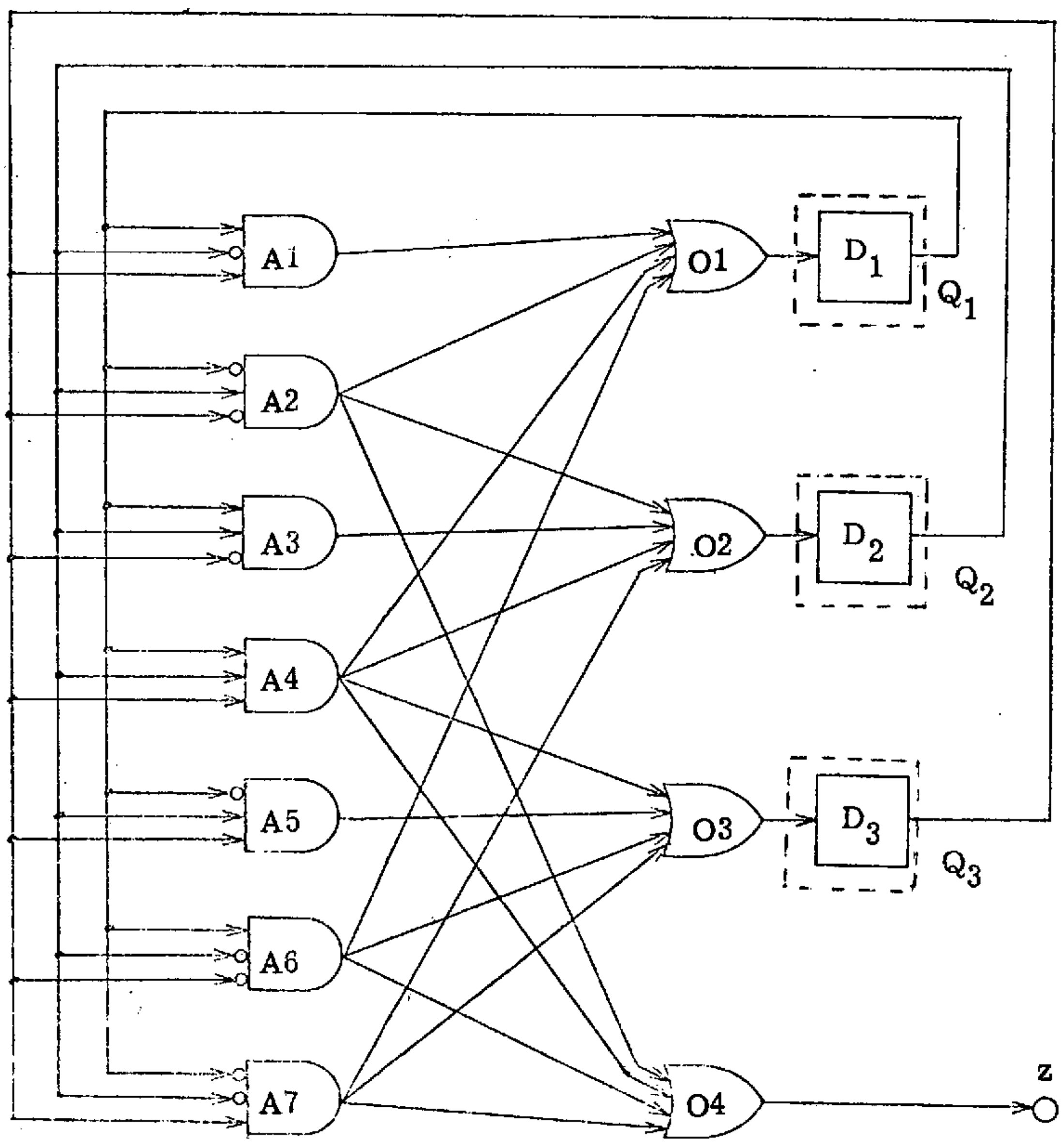

Figure 4. b.

```
>CRE
?A1...A7,01...04,Q1...Q3O2
?01...O4=0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
?Q1...Q3=DELAY
?A1=0,0,0,0,0,1,0,0
?A2=0,0,1,0,0,0,0,0
?A3=0,0,0,0,0,0,1,0
?A4=0,0,0,0,0,0,0,1
?A5=0,0,0,1,0,0,0,0
?A6=0,0,0,0,1,0,0,0
?A7=0,1,0,0,0,0,0,0
?Q1...Q3>A1...A7
?A1,A2,A4,A6>01
?A2...A4,A7>O2
?A4...A7>O3
?A2,A4,A6,A7>O4
?O1>Q1
?O2>Q2
?O3>Q3
?


?DISPLAY INPUTS A1...A7,01...04,Q1...Q3
 NODE  INPUTS
 A1    I=Q1,Q2,Q3
 A2    I=Q1,Q2,Q3
 A3    I=Q1,Q2,Q3
 A4    I=Q1,Q2,Q3
 A5    I=Q1,Q2,Q3
 A6    I=Q1,Q2,Q3
 A7    I=Q1,Q2,Q3
 O1    I=A1,A2,A4,A6
 O2    I=A2,A3,A4,A7
 O3    I=A4,A5,A6,A7
 O4    I=A2,A4,A6,A7
 Q1    I=O1
 Q2    I=02
 Q3    I=O3
?DIS OUT A1...A7,O1...O4,Q1...Q3
 NODE  OUTPUTS
 A1    O=O1
 A2    O=01,02,O4
 A3    O=O2
 A4    O=O1,02,03,04
 A5    O=O3
 A6    O=O1,03,O4
 A7    O=O2,03,04
 O1    O=Q1
 O2    O=Q2
 O3    O=Q3
 O4    O=NONE
```

NOT REPRODUCIBLE

```
 Q1     0=A1,A2,A3,A4,A5,A6,A7
 Q2     0=A1,A2,A3,A4,A5,A6,A7
 Q3     0=A1,A2,A3,A4,A5,A6,A7
?DIS SIZE
 SIZE IS 2 P0GES.
?END


>SIM
    0 INPUT N0DES.
    3 MACHINE N0DES.
   11 LOGIC N0DES.
*DIS VAL Q1...Q3
 N0DE VAL
 Q1    0
 Q2    0
 Q3    0
*STE 1
*DIS VAL Q1...Q3
 N0DE VAL
 Q1    0
 Q2    0
 Q3    0
*STE 100
*DIS VAL Q1...Q3
 N0DE VAL
 Q1    0
 Q2    0
 Q3    0
*STA 1,1,1  COMMENT:  SET INITIAL STATE T0 111
*DIS VAL Q1...Q3
 N0DE VAL
 Q1    1
 Q2    1
 Q3    1
*STE 1
*DIS VAL Q1...Q3
 N0DE VAL
 Q1    1
 Q2    1
 Q3    1
*STE 100
*DIS VAL Q1...Q3
 N0DE VAL
 Q1    1
 Q2    1
 Q3    1
*STA 0,1,1
*DIS VAL Q1...Q3
 N0DE VAL
 Q1    0
 Q2    1
 Q3    1
```

NOT REPRODUCIBLE

```
*STE 1  COMMENT: SHOULD TRANSFER TO STATE 001
*DIS VAL Q1...Q3
 NODE VAL
 Q1   0
 Q2   0
 Q3   1
*STE 1  COMMENT: SHOULD TRANSFER BACK TO STATE 011
*DIS VAL Q1...Q3
 NODE VAL
 Q1   0
 Q2   1
 Q3   1
*STA 1,0,1
*STE 1  COMMENT: SIMILARLI TRANSFER BETWEEN STATES 101 AND 100
*DIS VAL Q1...Q3
 NODE VAL
 Q1   1
 Q2   0
 Q3   0
*STE 1
*DIS VAL Q1...Q3
 NODE VAL
 Q1   1
 Q2   0
 Q3   1
*STE 1
*DIS VAL Q1...Q3
 NODE VAL
 Q1   1
 Q2   0
 Q3   0
*STA 1,1,0
*DIS VAL Q1...Q3
 NODE VAL
 Q1   1
 Q2   1
 Q3   0
*STE 1
*DIS VAL Q1...Q3
 NODE VAL
 Q1   0
 Q2   1
 Q3   0
*STE 1
*DIS VAL Q1...Q3
 NODE VAL
 Q1   1
 Q2   1
 Q3   0
*STE 1
*DIS VAL Q1...Q3
 NODE VAL
 Q1   0
 Q2   1
 Q3   0
*END
```

5. SOSS as a Computer Program.

The purpose of this section is to let the reader gain some insight into the structure of SOSS as a program.

5. 1. How SOSS is Stored.

SOSS uses a data base for the created user system. This data base occupies contiguous core locations. All the pointers in the data structure are relative addresses, relating to the beginning of the storage block allocated, so that the data base may be relocated in core. This enables SOSS to allocate more storage area for the user as the need arises. Initially the created system is assigned two MTS pages (a page is 1024 32 bit words divided into 4 bytes each). If more space is required, SOSS requests four pages from MTS, copies the information into the first two pages of the new block and frees the current block. This process is repeated each time the user runs out of space, and each time the block assigned to the data base is doubled. The total number of pages available for users on MTS is 256. The whole process is transparent to the user and is done automatically. The created system is stored in tables. The structure of the data base elements is given below.

STRUCTURE OF DATA-BASE ELEMENTS

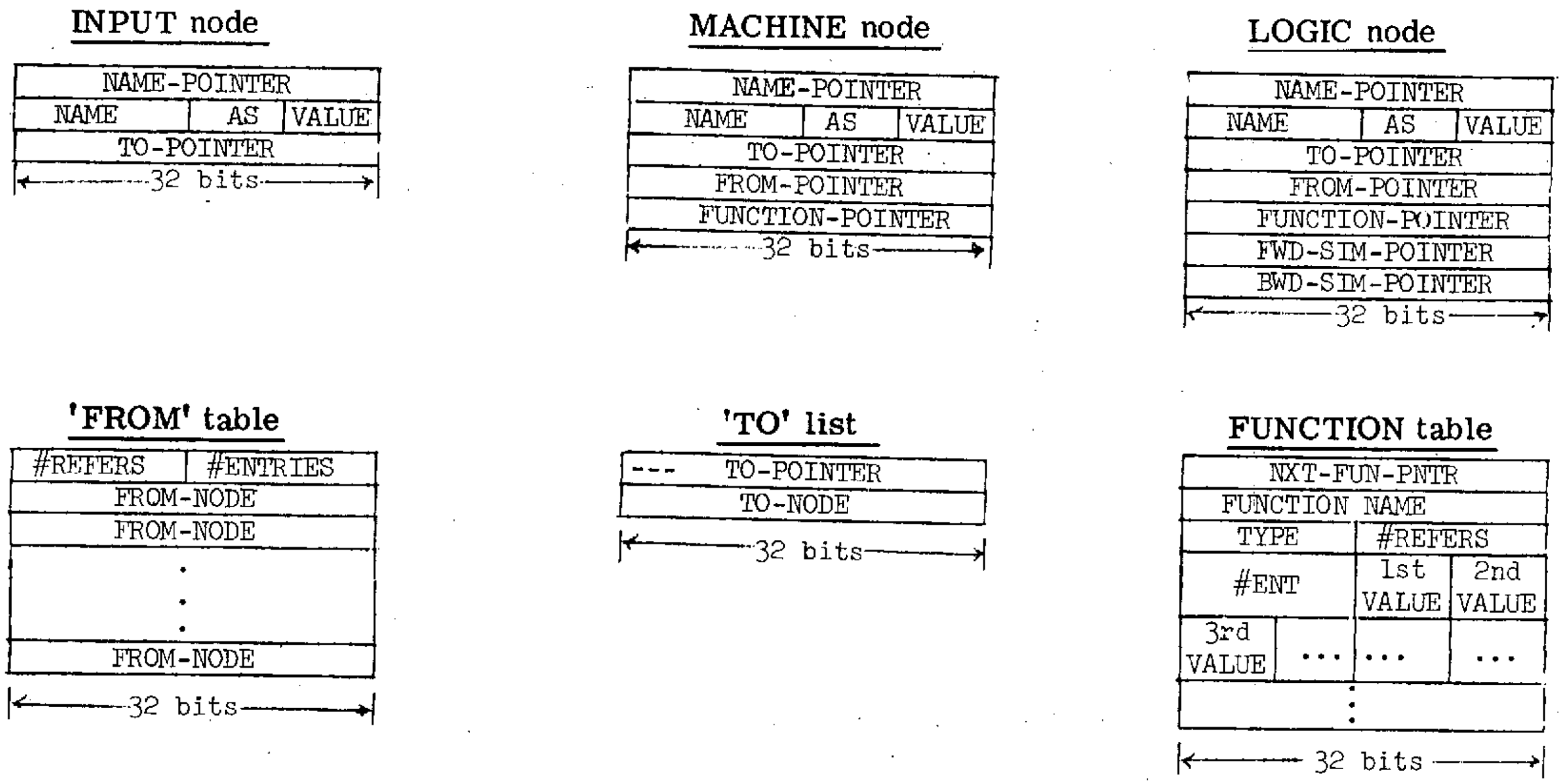

Figure 5.

An explanation of the terms used follows.

5. 1. 1. Node Elements

NAME POINTER	points towards the next node with the same name (i. e., the alphabetic character in the node name, e. g., the A in A28).
NAME	encoded node name and number
AS	the alphabet size
VALUE	the value assumed during the last clock period simulation of the function the node represents.
TO-POINTER	points to a list of nodes for which a particular node serves as an input.
FROM-POINTER	points to a list of nodes serving as inputs to a particular node.
FUNCTION-POINTER	points to the function table of a node.
FWD-SIM-POINTER BWD-SIM-POINTER	these pointers link all the combinational network nodes on a two way list. This list serves to detect illegal feedback loops. In the the process of checking for feedback loops the nodes on the list are automatically ordered for simulation.

5. 1. 2. 'FROM' table

Such a table is associated with each node, except for the system input nodes. These tables are of variable length. Each table contains pointers to the nodes which serve as inputs to the node associated with a particular table. This is useful in the ALTER mode where a node has to be disconnected or some connection deleted. The table also contains:

#REFERS	the number of nodes which share the same table. Several nodes can share the same table if, while in the CREATE mode, a statement of the form D2, A1, B5, Q2 > C1, C5, B1 is issued. C1, C5, and B1 will have the same 'FROM' table, with entries pointing towards D2, A1, B5, and Q2.
#ENTRIES	the number of nodes which serve as input to the node associated with a particular 'FROM' table. In the above example, the number of entries will be four. The maximum number of entries is 256.
FROM-NODE	each word contains a pointer to the nodes which serve as inputs to the node associated with that particular table.

5. 1. 3. 'TO' list

Such a list is associated with each node. It contains pointers to the nodes for which the node, associated with a particular table, serves as an input. The lists are of variable length. There is no limit on the fan out of any node. Each element in the list contains:

TO-POINTER	link information to the next entry in the list.
TO-NODE	pointer to a node for which the node, associated with a particular list, serves as an input.

5.1.4. FUNCTION table

Each function table contains the "value" column of a given function. A function table of a system function can be shared by more than a single node, saving data base space. In order to define his own common 'value' functions the user must, currently, specify such a function in the CREATE mode as a statement of the form

A1. . . A7, C2, B3 = 0, 1, 15, 255, 10, 2, 3, 4.

A1 through A7, B3, and C2 will then share the same function table. The entries in a function table are:

NXT-FUN-PNTR	a pointer to the next function. This entry is present only for named functions (functions not specified by a list of values). All the named functions are linked on a list.
FUNCTION-NAME	the name given to a named function. This entry is present for functions not specified by a list of values. A name can be made of up to 11 alphanumeric characters.

TYPE	type of function (named or list of values).
#REFERS	contains the number of nodes sharing a particular function table. This is necessary in the case where the value of a function, represented by a node associated with a particular table, is to be altered.
#ENT	number of entries in the table.
VALUE	one entry in the function table. The values consist of integers in the range 0-254. The integer 255 is reserved to indicate an unspecified value in an incompletely specified table.

5.2. How SOSS Operates.

This section presents an overview of the data processing and handling done by SOSS at the data base level. It is not, however, intended to be a detailed description of the processing and error checking done during execution.

5.2.1. Data Base Initialization

Data base initialization is performed upon entry into the CREATE mode. SOSS releases the memory associated with the last user system, if such exists. It then allocates space (2 pages) for the new system. Next, it proceeds to initialize and restore all the required constants and pointers in the data base.

5. 2. 2. Creation of a Node

The user creates a node by defining the alphabet size (AS) associated with it, e.g.,

B12@2.

SOSS first scans the data base to check whether such a node has been previously defined. If such a node already exists, the new AS value replaces the current AS and a message is issued, informing the user that the AS has been redefined.

If the node has not been previously defined, the appropriate node table (see Figure 5) is generated; the name of the node and its AS are inserted into the table and the pointers are initialized to zero. A node pointer that contains a zero signifies that this pointer has not, as yet, been defined.

Next, the node table is linked to the list of nodes having the same name (i.e., the same first alphabetic character, e.g., all the nodes starting with an A are on one list). The name lists are ordered lexicographically.

Finally, if the node is a logic node, i.e., part of the combinational network, it is linked to a two-way list which contains all the logic nodes.

5. 2. 3. Assignment of a Function to a Node

Assigning a function to a node is achieved by the "=" command. E.g.,

B12 = AND; C5 = 0, 1, 1, 3, 5, 2

If such a node does not exist SOSS informs the user of the fact, and no assignment is made. If the function is defined using a name, (e.g., AND) rather than a list of values, SOSS sets the function pointer of the node to the requested function. If the specified name is not found a message is printed out.

If a function is defined by a list of values, SOSS allocates space for the function table and inserts the values into the table in the order in which they were specified. It then sets the pointer in the node to the function table. If the node had a previously defined function associated with it, the user is informed that the function has been redefined.

5.2.4. Assignment of Input Connections to a Node

Specifying an input connection to a node is achieved by the ">" command. For example,

```
A3, B2, B5 > B6, Q1
```

establishes A3, B2, and B5 as the inputs to both B6 and Q1.

SOSS verifies that all the nodes exist and that no more than 255 inputs are assigned to a node. In the case such failures are detected a warning is issued and no assignment is made.

Next, a common 'FROM' table is generated for all the nodes specified on the right hand side of the statement. The table contains pointers to all the nodes serving as inputs to the nodes on the right, i.e., the nodes specified on the left hand side of the statement.

If the connections to some node have been previously defined, a message is printed out; the connections are disconnected by deleting the pointer to the old 'FROM' table; also, the nodes on the right hand side are deleted from the 'TO' list associated with each node in the old 'FROM' table.

If the nodes on the right hand side are part of the combinational network SOSS establishes that no feedback loops exist (see section 5. 3.). Next, the forward and backward simulation pointers, and the pointer to the new 'FROM' table are established. Finally, the nodes on the right hand side of the statement are linked to the 'TO' lists of each node in the new 'FROM' table.

5. 2. 5. Simulation

Simulation is executed one clock period at a time. The following steps are invoked for each clock period.

i) Whenever input nodes are defined, SOSS verifies that all the inputs are present and do not exceed their defined AS. It then inserts these values in the appropriate input nodes.

ii) Using the forward simulation list (established during CREATE-ion) the values for each combinational network node are calculated.

iii) The values of the outputs of the machine nodes are calculated.

iv) The clock is advanced.

The algorithms used to order the combinational network nodes for simulation and to evaluate the node outputs are discussed in section 5. 3.

5. 3. The Algorithms Used in SOSS

5. 3. 1. The Feedback Loop Test

No feedback loops are allowed in the combinational network. The following algorithm tests the created system for feedback loops and simultaneously arranges the nodes of the combinational network for fast processing during simulation. While in CREATE, each time a connection between two nodes is specified it is immediately tested. If an illegal feedback loop is detected no connection is made, and an error message is printed out.

An illegal feedback loop is defined, for our purpose, as a closed chain of connected nodes which contains only combinational network nodes. The test involves scanning down a tree-like structure. A path is terminated in one of three cases:

i) No connection eminates from some node in the path which means that it has not been specified yet. This is a legal path.

ii) A machine node is encountered. This is a legal path.

iii) A combinational network node which has been encountered previously is encountered again. This is an illegal path.

The search terminates when all the possible paths have been scanned and found valid or when a feedback loop has been found. The process makes use of a two way list which contains all the logic nodes that have already been connected. The node which inputs have just been specified is put at the endof the two way list. The test is executed as follows:

1) set a "ring-pointer" indicating the node whose inputs have just been specified.

2) Get the first entry in the 'TO' list associated with the node indicated by the ring-pointer.

3) Do one of the following:

a) If the 'TO' list is empty go to 6.

b) If the entry is a machine node get the next entry in the 'TO' list.

c) If the entry is a logic node verify that it is not listed in the 'FROM' table of the node which inputs have just been specified. If it is listed in it, a feedback loop exists, so go to 5. Otherwise, move the node referenced in the 'TO' table to the end of the two-way list of combinational network nodes. Get the next entry in the 'TO' list. Go to 3.

d) If the 'TO' list associated with the node referenced by the ring-pointer has been exhausted, go to 4.

4) If the ring-pointer points at the last node in the combinational network nodes list then go to 6. Otherwise, advance the ring-pointer to the next node on the list and go to 2.

5) A feedback loop has been detected. Print out error message. Go to 6.

6) Test terminated. SOSS prompts the user for the next CREATE (or ALTER) command.

This test algorithm simultaneously arranges the logic nodes for simulation. The nodes are evaluated one at a time according to the two way list. The algorithm assures that when a node output is to be evaluated, all the nodes serving as inputs to that particular node have already been evaluated.

5. 3. 2. Evaluating the Output of a Node

The output of a node is evaluated by using the function table associated with that node. Given a node the process of output evaluation is executed as follows:

1) the alphabet size (AS) and current value of each node referenced in the 'FROM' table of the given node is picked up.

2) An index into the function table of the given node is calculated using the formula

$$\text{Index} = V_1(AS_2 \times \ldots \times AS_N) + \ldots + V_i(AS_{i+1} \times \ldots \times AS_N) + V_{N-1}(AS_N) + V_N$$

where:

V_k is the current value of the k-th node in the 'FROM' table

AS_K is the alphabet size of the k-th node in the 'FROM' table.

N is the number of entries in the 'FROM' table.

3) If the calculated index falls outside the function table an error message is issued. If the index is legal, the function value associated with it is picked.

4) If the function value obtained is within the AS specified for the function it is inserted in the table of the node. Otherwise an error message is issued.

While in simulation, all the calculated function values for the machine nodes are stored temporarily. Only when all the values have been obtained and no errors detected are these values inserted into the appropriate node tables.

5) The clock is advanced.

5.4. Comments on Statistics and Efficiency.

Statistics ran on SOSS while in execution show that for small systems, most of the processor time is spent on input/output operations. The time spent on simulation becomes larger, as expected, when the simulated system grows. However, for a large system the overhead becomes insignificant and the processor time spent on simulation is almost a linear function of the system size.

Because of the test algorithm described above, the efficiency of SOSS in the CREATE mode increases if the simulated system is specified according to the signal flow through it. Less time is spent then on testing for feedback loops.

6. Conclusion

SOSS has been designed to accommodate the need for a versatile simulation system with a powerful command language that is not limited to binary systems. Currently only the basic SOSS has been implemented, but the provisions have been made for easy completion of all the options. In addition, the need for an associate program that will handle comparison of output data for large systems is foreseen. Such a program, however, should not be part of SOSS.