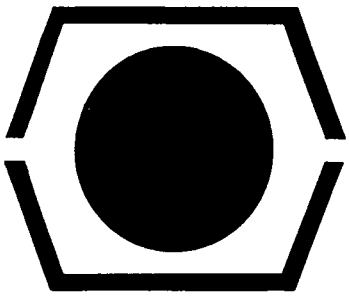

DECUS
PROGRAM LIBRARY

DECUS NO.	8-420
TITLE	LOGSIM-8
AUTHOR	Robert Stolarz
COMPANY	Princeton University Princeton, New Jersey
DATE	April 7, 1971
SOURCE LANGUAGE	

LOGSIM-8

DECUS Program Library Write-up DECUS No. 8-420

Abstract

LOGSIM-8 is an interactive digital logic simulation program for the simulation of combinational and sequential logic circuits at the gate level. The language is simple, and allows logical units such as flip-flops to be called as functions. The output consists of a table of the values of selected variables during each pass through the circuit description.

Language

The simulation language consists of two types of statements: logic statements and control statements. Statements are numbered, the line number <n> being an integer number from 0 to 999, or an integer number from 0 to 999 preceded by an asterisk in the case of a control statement. LOGSIM does not, however, order statements by line number.

The logic statements are a description of the gates and other functional elements making up a circuit; the connections between elements are described by means of variables which have only one value associated with them at any time, either 0, 1 or undefined; any change in the value of a variable occurs immediately. A variable is initially undefined and remains undefined until some statement is executed which would cause it to be set to a 0 or 1, or until it is used as an input argument, when LOGSIM will ask for its value. Variable names may be one to three alphanumeric characters in length, the first of which must be alphabetic. The constants 0 and 1 are also permissible as input arguments to gates or functions. An expression consists of a variable, a constant, or a gate reference. A gate is referenced by giving its name followed by its arguments (expressions) separated by commas and enclosed in parentheses. An input argument to a gate may thus contain references to other gates, and recursive calls are also permitted. LOGSIM supplies the following gates:

NOT, OR, NOR, AND, NAND, XOR

NOT is the only gate which has a restriction on the number of inputs; it may only have one.

Logic Statements

Logic statements are of two types: the assignment statement and the function reference. The general form of an assignment statement is as follows: (any quantity in slashes may be repeated as many times as necessary, or omitted entirely)

<n> <var> = <gate> (<expr> /, <expr>/)

For example:

```
5  C=NOR(A,B)
2  Y=AND(NOT(X1),NOT(X2))
7  OP7=NAND(NAND(OP1,OP5),NAND(OP2,OP3))
```

The variable named on the left-hand side of the equal sign will be assigned the output of the gate named on the right. The line number <n> can be used to identify a given element on a circuit diagram with the corresponding statement in the circuit description.

The second type of logic statement is the function reference. Its form is:

<n> <function> (<expr> /,<expr>/ ; <var> /,<var>/)

where <function> can be one of the functions listed in the table below. The semicolon separates input expressions from output variables. The significance of each of the arguments is dependent upon its position, as can be seen in the table: (in this table only, the slashes indicate that the second output argument, <Qbar>, is optional)

HA(<inp1>,<inp2>;<sum>,<carry>)

FA(<inp1>,<inp2>,<inp3>;<sum>,<carry>)

SR(<set>,<reset>;<Q>/,<Qbar>/)

CSR(<clock>,<set>,<reset>;<Q>/,<Qbar>/)

D(<clock>,<inp>;<Q>/,<Qbar>/)

JK(<clock>,<J>,<K>;<Q>/,<Qbar>/)

SRJK(<set>,<reset>,<clock>,<J>,<K>;<Q>/,<Qbar>/)

For example:

```
11  HA(A,B;SUM,CAR)
52  SR(AND(F1,CLK),RST;F2,NF2)
21  JK(CLK,1,1;L21)
```

The proper number of input arguments must be supplied for each function; only HA and FA require two output variables. The second output argument for the other functions is just the complement of the first; it may omitted if it is not needed. It should be noted, however, that <Q> and <Qbar> are totally independent variables.

The <set> and <reset> inputs to SR, CSR and SRJK have the following effect on <Q> (and a corresponding effect on <Qbar> when it is specified): when both are zero, no change in <Q>; when <set> is one, <Q> is set to one; when <reset> is one, <Q> is reset to zero; when both are one, an error condition exists. The <set> and <reset> inputs to SRJK have precedence over the other inputs.

The clock inputs to the functions are not edge-triggered; whenever a function statement containing a clock input is executed, and its clock input is one, the indicated action, if any, is performed. In order to simulate an edge-trigger, the value of the clock input can be saved after the function statement so that the old value of the clock can be compared with the new value on the next pass. For example, the following sequence will produce a leading-edge trigger:

```
10  JK(AND(X,NOT(XS)),1,1;FLG)
99  XS=OR(X)
```

For a trailing-edge trigger:

```
10  JK(AND(NOT(X),XS),1,1;FLG)
99  XS=OR(X)
```

To simulate sequential circuits, any of the flip-flops can be used as memory elements. Corresponding to the general model of a sequential circuit, such memory elements should generally be placed at the end of the circuit description, so that the new values of their outputs will not take effect until the next pass.

Control Statements

The control statements are used to input and output variables or set them to certain values. The general format of a control statement is:

<n> <keyword> <var> /, <var> /

where <keyword> in this format may be INPUT, OUTPUT or CYCLE.

INPUT

When this statement is encountered, the program will type the names of the variables in the statement list followed by a question mark, after which the user should type the value (0 or 1) he wishes the variable to assume. The program will ignore any characters typed until a 0 or 1 is encountered, and then ask for the next value, if any. When all variables have been assigned, the program will issue a carriage return and line feed and will continue the simulation with the next statement.

OUTPUT

On first encountering this statement, the program will print a heading consisting of the variable names in the list. On this and each consecutive time that it executes the OUTPUT statement, the program will print the values of the variables under their names in table form. If there is more than one OUTPUT statement in a circuit description, a heading will be produced only for the first OUTPUT statement encountered.

CYCLE

On first encountering this statement, the program will assign 0 to any variables in the list that have not already been assigned values. After the program performs the last statement in the circuit description, control will return to the CYCLE statement. Starting with the left-most variable in the list, CYCLE will search for the first variable with a value of 0 and assign it a value of 1 and the simulation will continue with the next statement. Any variables with values of 1 that CYCLE passed over in searching for a 0 value are assigned 0 values. When control returns to the CYCLE statement and the values of each of the variables in the list is 1, the simulation is terminated. There should be only one CYCLE statement in a circuit description.

Thus CYCLE may be used to supply the values necessary to print a truth table for a given circuit, the first variable in the CYCLE list varying most rapidly. CYCLE may also be used simply to cause the simulation statements to be repeatedly executed by using dummy variables in the statement list (for example, in conjuction with an INPUT statement to allow the testing of a sequential circuit).

SET

There is another control statement whose format differs slightly from that of the other control statements. Its keyword name is SET and its format is:

<n> SET <var> = 0 | 1 /, <var> = 0 | 1 /

Upon encountering this statement, the program will set the named variables to the given values.

Operation

LOGSIM is loaded by means of the binary loader; its starting address is octal 200. LOGSIM is similar in operation to FOCAL, with the exception that it uses a different line number-editing scheme. LOGSIM starts in command mode, in which it will accept commands or simulation statements. Any line beginning with a letter will be interpreted as a command, and any line beginning with a number or an asterisk followed by a number will be interpreted as a simulation statement to be stored in the buffer. Every line is terminated with a carriage return. Lines may be longer than 72 characters; LOGSIM automatically accounts for line overflow by issuing a carriage return-line feed.

LOGSIM maintains an internal pointer which determines where a simulation statement entered from the keyboard will be placed in the buffer. Initially, the pointer is set at the beginning of the buffer (when there is nothing in it). When a line is input, it is placed in the buffer at the position indicated by the pointer, and then the pointer is moved immediately after the line just added. The line numbers used in simulation statements thus have no relation at all to the way the lines are stored in the buffer or to the order in which the simulator will look at them.

Should one wish to insert a line or group of lines before a certain line presently in the buffer, the insert command, I<n>, is given. When this command is given, the pointer will be moved before the line with the given number. To add lines at the end of the buffer, the append command, A, is given. To delete the entire buffer or a specific line, one may give the commands D or D<n>, respectively.

To list the entire buffer, L is given. If L<n> is given, the listing will start at the given line and continue until the last line in the buffer has been printed. To save simulation statements on paper tape, type L, turn on the punch, and type carriage return. (LOGSIM only uses low-speed paper tape equipment.)

To replace a line, one may simply retype the line; the replacement will occur and the pointer will not be affected.

Blanks are ignored everywhere in general, with the exception that there must be a blank following the keyword in a control statement. Note also that only the first character of a control statement keyword is significant; thus OUTPUT may be abbreviated OUT or O, or extended to OUTPUTS.

There are two characters used for error correction when typing a line. The rubout (echoed as a backslash) will erase a single character to the left each time it is typed, while the back-arrow will erase all of the line just typed in. LOGSIM will respond to an invalid command or simulation statement with a question mark.

To run the simulation, the command G is given. Upon completion, LOGSIM will return to command mode. If any variable used as an input argument does not have a value when it is referenced, the program will ask for its value as with INPUT. To interrupt LOGSIM at any time and return to command mode, type CTRL/C. A question mark will be printed to indicate interruption. There is no immediate mode execution of simulation statements available.

Should the simulator detect an error while running the simulation, it will print an error message of the form ? <errno> @ <n> , and return to command mode. The meaning of the error numbers is given in the table below.

LOGSIM's internal pointers have been set to allow 128 variables, 128 simulation statements and 4000 characters of buffer storage. The user should note that no warning will be given if these limits are exceeded, although this is not likely to occur.

Comments

The author is currently planning a second version of LOGSIM-8 which would implement features desirable for the simulation of sequential circuits. The author would appreciate receiving any comments or suggestions that a user may have about LOGSIM-8.

Command Summary

A	Append after last line
D	Delete all lines
D<n>	Delete line <n>
I<n>	Insert before line <n>
L	List all lines
L<n>	List from line <n>
G	Begin simulation

Error Messages

1 - name too long
2 - illegal character
3 - improper number, sequence or type of argument
4 - invalid character sequence
5 - invalid gate or function name
6 - invalid control statement keyword
7 - invalid SR, CSR or SRJK input
8 - mismatched parentheses

Examples

(1) verifying DeMorgan's law

```
*1  CYCLE Y,X
 1  A1=NOR(X,Y)
 2  A2=AND(NOT(X),NOT(Y))
*2  OUTPUT X,Y,A1,A2
```

(2) comparing LOGSIM's XOR with one constructed with NANDs

```
*1  CYCLE B,A
 1  X1=NAND(A,B)
 2  X2=NAND(A,X1)
 3  X3=NAND(B,X1)
 4  XR1=NAND(X2,X3)
 5  XR2=XOR(A,B)
*2  OUTPUT A,B,XR1,XR2
```

(3) simple ALU implemented with NAND/NOR logic to perform the following functions:

when X1=0 and X2=0, C=AND(A,B)
when X1=0 and X2=1, C=OR(A,B)
when X1=1 and X2=0, C=NOT(A)
when X1=1 and X2=1, C=0

```
*1  CYCLE B,A,X2,X1
11  NX1=NAND(X1)
12  NX2=NAND(X2)
13  NA=NAND(A)
21  O21=NOR(A,B)
22  O22=NOR(O21)
31  O31=NAND(NX1,A,B)
32  O32=NAND(NX1,X2,O22)
41  O41=NAND(X1,NX2,NA)
42  C=NAND(O31,O32,O41)
*2  OUTPUT X1,X2,A,B,C
```

(4) sequential machine to recognize the sequence 1101 with the possibility of overlap (sets Z to 1 after receiving this sequence)

```
*0  SET Y1=0,Y2=0,NY2=1
*1  CYCLE A,B,C,D,E,F,G,H
*2  INPUT X
31  Z=AND(X,Y1,NY2)
11  JK(1,AND(X,Y2),NY2;Y1)
21  JK(1,X,NOT(X);Y2,NY2)
*3  OUTPUT Y1,Y2,Z
```

(5) BCD counter

```
*0  SET RST=1,D4=0,ND4=1
*1  SET XS=0,G1S=0,D1S=0,G4S=0
*2  CYCLE X,Z1,Z2,Z3,Z4
 0  SRJK(0,RST,AND(NOT(X),XS),1,1;D0)
90  XS=OR(X)
10  G1=AND(D0,ND4)
 1  SRJK(0,RST,AND(NOT(G1),G1S),1,1;D1)
91  G1S=OR(G1)
 2  SRJK(0,RST,AND(NOT(D1),D1S),1,1;D2)
92  D1S=OR(D1)
11  CAR=AND(D0,D4)
12  G4=OR(D2,CAR)
 4  SRJK(0,RST,AND(NOT(G4),G4S),1,1;D4,ND4)
94  G4S=OR(G4)
*3  OUTPUT D4,D2,D1,D0,CAR
*4  SET RST=0
```

LogSim-8 DECUS 8-420

Line # ϕ to 999 logic Statement

Line # *ϕ to 999 control Statement

line # identifies but does not indicate order of execution

Logic Statements and Variables

functional elements - gates, flip flops etc

connected by

Variables with values of 0 or 1 or 'undefined'

Variable - initially undefined - ① is defined by use in ~~requiring a value~~ execution of a statement

② or if used as an input argument - program asks for value

NAMES - up to 3 characters, first is alphabetic

Constants - 0 or 1

Expression (argument of gate)

Variable, constant, or gate reference

Reference Gate

NAME (expression, expression, ---, ---, ...)

- input arguments to a gate may contain references to other gates are permitted

NOT ← only one input
OR
NOR
AND
NAND
XOR ← exclusive OR.

Logic STATEMENT

assignment or function reference

Assignment type:

line # → n _ variable = gate name (—, —, — ...)

2 _ Y = AND (NOT(X1), NOT(X2))

Function reference

line # function (input: expr, expr ; output: var, var)

(3) simple ALU implemented with NAND/NOR logic to perform the following functions:

when X1=0 and X2=0, C=AND(A,B)
when X1=0 and X2=1, C=OR(A,B)
when X1=1 and X2=0, C=NOT(A)
when X1=1 and X2=1, C=0

```
*1  CYCLE B,A,X2,X1
11  NX1=NAND(X1)
12  NX2=NAND(X2)
13  NA=NAND(A)
21  O21=NOR(A,B)
22  O22=NOR(O21)
31  O31=NAND(NX1,A,B)
32  O32=NAND(NX1,X2,O22)
41  O41=NAND(X1,NX2,NA)
42  C=NAND(O31,O32,O41)
*2  OUTPUT X1,X2,A,B,C
```

(4) sequential machine to recognize the sequence 1101 with the possibility of overlap (sets Z to 1 after receiving this sequence)

```
*0  SET Y1=0,Y2=0,NY2=1
*1  CYCLE A,B,C,D,E,F,G,H
*2  INPUT X
31  Z=AND(X,Y1,NY2)
11  JK(1,AND(X,Y2),NY2;Y1)
21  JK(1,X,NOT(X);Y2,NY2)
*3  OUTPUT Y1,Y2,Z
```

(5) BCD counter

```
*0  SET RST=1,D4=0,ND4=1
*1  SET XS=0,G1S=0,D1S=0,G4S=0
*2  CYCLE X,Z1,Z2,Z3,Z4
 0  SRJK(0,RST,AND(NOT(X),XS),1,1;D0)
90  XS=OR(X)
10  G1=AND(D0,ND4)
 1  SRJK(0,RST,AND(NOT(G1),G1S),1,1;D1)
91  G1S=OR(G1)
 2  SRJK(0,RST,AND(NOT(D1),D1S),1,1;D2)
92  D1S=OR(D1)
11  CAR=AND(D0,D4)
12  G4=OR(D2,CAR)
 4  SRJK(0,RST,AND(NOT(G4),G4S),1,1;D4,ND4)
94  G4S=OR(G4)
*3  OUTPUT D4,D2,D1,D0,CAR
*4  SET RST=0
```