NASA Contractor Report 177939

NASA-CR-177939
19860010477

The Development of An Interim Generalized Gate Logic Software Simulator

J.G. McGough and S. Nemeroff

Allied/Bendix Aerospace
Flight Systems Div.
Teterboro, New Jersey

Contract NAS1-15946

December 1985

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1.0 SUMMARY AND CONCLUSIONS

1.1 Summary

An interim version of GGLOSS (called IGGLOSS) was developed and tested for the purpose of obtaining an early assessment of the predicted performance of GGLOSS. IGGLOSS omitted only those features of GGLOSS which have little or no affect on the essential performance capabilities of GGLOSS. Five circuits were simulated and IGGLOSS performed exactly as anticipated.

IGGLOSS was hosted on a VAX 11/780 computer and was programmed in Fortran and Bliss, the latter being used exculsively for high speed logic computations.

Memory Requirements

IGGLOSS required 3504 Bytes of VAX memory including the library of Bliss-coded primitive macros.

Simulation Speed

Non-Faulted	:	9.5 x 10**6 Gates/Sec of VAX 11/780 CPU time
Faulted	:	4.4 x 10**6 Gates/Sec of VAX 11/780 CPU time

1.2 Conclusions

- IGGLOSS performed exactly as anticipated.
- The simulation techniques of GGLOSS, as described in (Ref. 3), appear to be sound.
- The memory requirements of IGGLOSS (3504 Bytes) are modest.
- The simulation speed of IGGLOSS is at least comparable to that of BGLOSS.
- It is recommended that GGLOSS be completed. The additional tasks required to transform IGGLOSS into GGLOSS are given in Section 5.0.

2.0 INTRODUCTION

In the Fall of 1979 Bendix was awarded a contract by NASA Langley Research Center to perform a fault simulation study to determine fault latency in a digital avionics processor (ref. 1). Prior to the award, Bendix had developed a gate logic software simulator (BGLOSS) for its BDX-930 digital computer. The study provided the opportunity to not only establish fault latency statistics but to test BGLOSS in a variety of scenarios, not the least of which included the simulation of software programs and their interaction with hardware faults. Prior simulation experience lead to the conclusion that, next to reasonable accuracy, simulation speed was the most important characteristic of a simulator. Current, commercially available varieties are too slow for the types of fault experiments envisioned, being of the order of 1000 gates/sec of host computer cpu time. The impact of such speeds can readily be appreciated by considering the application in which it is desired to determine the detectability of a fault by a self-test program consisting, typically, of 1000 assembly language instructions. In the BDX-930, an assembly language instruction requires, on the average, four passes through the cpu, which consists of 5000 equivalent gates. Thus, a single fault, together with a complete execution of self-test, requires the simulation of 20 million gates. At 1000 gates/sec it would require 5.55 hours of cpu time! Subsequently, after thousands of simulated faults, the speed of BGLOSS was established at 2.86 million gates/sec on a Vax 11/780 host computer. As a consequence of the success of BGLOSS in the context of the Fault Latency Study, NASA Langley Research Center awarded Bendix a follow-on contract to determine the feasibility of developing a generalized gate logic software simulator (GGLOSS) based upon the BGLOSS model (ref. 3).

As a result of this contract it was concluded that a GGLOSS simulator, which was at least comparable in speed to BGLOSS, was definitely feasible. As a consequence, NASA Langley Research Center awarded Bendix a contract to develop an interim version of GGLOSS (hence forth referred to a "IGGLOSS") for the purpose of validating the predicted performance of the final version before initiating a full-scale development program. This report describes the subsequently developed IGGLOSS and its performance.

Because IGGLOSS is an interim version of GGLOSS it is appropriate to include here an overview of GGLOSS (a detailed description is given in (ref. 3)).

2.1 Objectives of GGLOSS

GGLOSS is a fault simulator to be used for the purpose of:

- Conducting failure modes and effects analyses
- Designing and validating self-test programs
- Obtaining fault latency data for use in advanced reliability prediction programs such as CARE III.

It is important to emphasize that GGLOSS was never intended as a circuit design tool. Consequently the reader will observe, subsequently, that it lacks many of the features normally contained in a commercially available simulator.

The objectives of GGLOSS are:

- Gate level software simulator
- Generalized to the extent necessary to simulate virtually any representable by gate logic
- Reasonably transportable
- Very high speed, comparable to that of BGLOSS
- User-friendly
- Employs "standard" circuit specifications

2.2 Requirements of GGLOSS

Based on previous and extensive fault injection experiments (ref. 1,2) with BGLOSS the following requirements for GGLOSS were established:

a. The simulator must be capable of simulating software. This was a basic requirement since the objectives include the design and validation of self-test and the evaluation of fault latency when comparison-monitoring is the method of fault detection.

b. The simulator must yield results in a timely manner. The simulation of self-test and flight control applications programs requires many passes through the CPU. Considering the quantity of faults that were to be simulated it was the judgement of the BGLOSS design team that the simulation time, on whatever computer BGLOSS was hosted on, should not exceed 25000 times real time. Assuming 5000 gates in the CPU, at a clock cycle of 250 nanoseconds, this would be equivalent to simulating 801,753 gates/second (In any event, BGLOSS, simulated on a VAX 11/780, did not exceed 7000 times real time, which was equivalent to 2.86x10**6 gates/second).

c. The simulator must be capable of simulating multiple CPU's, with different software programs, concurrently. Because many of the envisioned simulation experiments involved redundant channels of a flight control system, it was desired that the simulation should be capable of modeling the concurrent operation of synchronous and asychronous channels, i.e., processors which, effectively, execute different software programs concurrently.

In summary, the simulator must be capable of simulating software. The simulator must yield results in a timely manner and the simulator must be capable of simulating multiple CPU's, operating synchronously or asynchronously with each, possibly executing different software programs.

2.3 Design Specifications of GGLOSS

In order to meet the objectives and satisfy the requirements of GGLOSS, the following specifications were established (ref. 3):

- Programmed in FORTRAN and BLISS
- Control and executive functions programmed in FORTRAN
- Hosted on a VAX 11/780
- Arithmetic and logic functions programmed in Bliss
- Employs a reasonably standard circuit specification syntax
- User - interactive
- Parallel mode simulation, exclusively (i.e., 32 circuits in parallel)
- Fixed order of node evaluations
- Orders combinational networks (P-ordered)
- 2-valued logic, exclusively
- Accommodates functional-level modules, e.g., memories, flip-flops
- User must initialize the network
- Stuck-at faults, exclusively
- Automatic fault selection if desired
- Simulates multiple faults in a single circuit
- Simulates ROM and RAM memories, 32 different copies of each, if required
- Simulates a fictitious clock and multiples thereof, in any desired quantity
- Simulates faults in ROM and RAM
- Simulates intermittent faults
- Collapses faults
- High speed of the order of 2 million gates/second of VAX 11/780 CPU time.

3.0 IGGLOSS

IGGLOSS was intended as a scaled-down version of GGLOSS for the purpose of obtaining an early assessment of the predicted performance of GGLOSS. IGGLOSS omitted those features of GGLOSS which were judged to have little or no affect on the essential performance capability of GGLOSS. Moreover, anticipating the success of IGGLOSS and to preclude a rework of the IGGLOSS software, IGGLOSS was designed to be the "core" element of the future GGLOSS. Thus, GGLOSS would be the result of adding the omitted features to IGGLOSS.

3.1 Objectives of IGGLOSS

An interim version of GGLOSS for the purpose of obtaining an early assessment of the essential performance of GGLOSS.

3.2 Requirements of IGGLOSS

- IGGLOSS must comprise the "core" element of GGLOSS
- IGGLOSS must employ the identical simulation techniques of GGLOSS
- IGGLOSS must be capable of simulating a variety of circuits, including memories, of sufficient complexity to allow an assessment of GGLOSS performance capabilities

3.3 Design Specifications of IGGLOSS

IGGLOSS incorporates all of design features of GGLOSS with the following exceptions:

- IGGLOSS interacts with the User to a limited extent
- IGGLOSS does not contain automatic fault insertion features
- IGGLOSS can only simulate 31 faults at a time. The User is required to recompile IGGLOSS for each set of 31 faults
- Although IGGLOSS allows for an arbitrary number of clocked nets, the clocks must have the same frequency. GGLOSS can accommodate clocks (i.e. fictitious) which are multiples of a basic clock and in any desired quantity
- IGGLOSS did not partition the network, i.e. all gates were replaced by their corresponding fault models at the start of each run
- Hierarchical networks could not be constructed
- IC's could only be set to logic 0s
- IGGLOSS could not simulate intermittent faults
- IGGLOSS could not simulate multiple faults in the same circuit
- IGGLOSS could not collapse faults
- IGGLOSS could simulate 32 different RAM's in parallel but could only simulate 32 identical ROM's.

4.0 PERFORMANCE OF IGGLOSS

The simulation techniques employed by IGGLOSS are identical to those of GGLOSS (ref. 3). Indeed, it was not found necessary to change or modify a single design specification. As indicated previously, a detailed description of GGLOSS is given in (ref. 3). However, as an aid to the reader, we give here an overview of these simulation techniques and the User-procedures in setting up a simulation.

4.1 Overview of an IGGLOSS Simulation

1) The User creates a Partslist for the circuit (see User's Manual). For illustrative purposes, the circuit could be the 4-BIT downcounter of Figure 3. The User may specify the parts in any order whatsoever.

2) In the Partslist the User must identify the location of fictitious clocks, in effect, by treating them as special buffer oates.

   Fictitious clocks are placed in order to break feedback paths. In the case of the downcounter, the User should break the circuit at the D-inputs of the 4 flip flops. Thereafter, IGGLOSS assumes that the resultant circuit is a collection of disjoint, combinational circuits. It treats the inputs to a clock gate as an external output and the output of the clock gate as an external input.

3) By prompts IGGLOSS will request

   - identity of faulted components and the type of fault
   - contents of ROM memory
   - identity of memory faults
   - number of clock cycles in a run
   - input as a function of clock cycle
   - output options

   IGGLOSS is now ready to compile.

4) IGGLOSS reorders the circuit, placing all gates in ranks: the first rank containing gates whose only inputs are external inputs. The second rank containing gates whose only inputs are outputs from gates in the first rank. This ordering continues until all gates are assigned to a rank. If IGGLOSS discovers a feedback path it will print an error message and stop. If the ordering is successful, IGGLOSS will convert the reordered partslist into a Bliss-coded program and print the results.

   IGGLOSS is now ready to execute.

5) IGGLOSS simulates, in parallel, 32 circuits, one of which is always non-faulted. IGGLOSS makes a single pass through the network, simulating every gate as it does so. At the end of each pass IGGLOSS compares the values of all test pins with those of their counterparts in the non-faulted circuit. If there is a difference the test pin, the fault and the clock cycle are noted. The "detected" fault, however, is no longer tracked. Thus, if subsequently another test pin "detects" this fault the fact will be ignored.

6) IGGLOSS repeats the above process for the next input vector. However, before executing the next pass, IGGLOSS transfers the input of all clock gates to the outputs. Effectively, the clock gates are treated as if they were D-flip flops.

4.2 Test Circuits

IGGLOSS simulated 5 circuits

- Test Circuit (Figure 1)
- Arithmetic Logic Unit (Figure 2)
- 4-Bit Downcounter (Figure 3)
- BCD Adder (Figure 4)
- Memory Circuit (Figure 5)

The corresponding partslists, P-ordering, test pin locations, faults, input sequences and results are described in the User's Manual. The partslists for the ALU and the BCD adder were already contained in the Bendix Circuit Library and only required the addition of test pin locations to be compatible with IGGLOSS.

The responses of IGGLOSS were manually checked in both the faulted and non-faulted runs.

4.3 Simulation Results

Memory Required

The IGGLOSS compiler, including the Library of Bliss-coded macros required only 3504 bytes of host computer memory.

Simulation Speed

To estimate the simulation speed of IGGLOSS the downcounter was simulated with and without faults. In the non-faulted case, IGGLOSS made 1000 passes through the circuit; in the faulted case 31 gates were faulted and IGGLOSS made another 1000 passes through the network. The results were:

Non-faulted, 1000 passes CPU time = 0.34 seconds
Faulted, 1000 passes CPU time = 0.74 seconds

The CPU times were measured from the start to finish of the BLISS program and did not include compile time.

We estimated the number of gates per second of CPU time as follows:

Non-Faulted Circuit

The downcounter contains 101 gates. Since 32 circuits are executed in parallel, the total number of gates simulated were:

1000 x 32 x 101 in 0.34 seconds

= 9.5 x 10**6 gates/sec of VAX 11/780 CPU time

Faulted Circuit

As indicated previously, IGGLOSS replaced every gate by its fault model. This was a very inefficient procedure since only 31 gates were faulted. Without counting these additional gates, the total number of gates simulated was taken to be, conservatively,

1000 x 32 x 101 in 0.74 seconds

= 4.4 x 10**6 gates/sec of VAX 11/780 CPU time.

ROM/RAM Circuit Timing

The introduction of memory elements in a circuit significantly reduces simulation speed since these devices cannot be simulated in parallel. Moreover, each execution of a memory requires two transformations:

1) A transformation from parallel to serial
2) A transformation from serial back to parallel

IGGLOSS simulated the memory circuit of Figure 5, which contained a ROM and RAM memory.

IGGLOSS made 1000 passes through the circuit, which required a total of 2.79 seconds of VAX 11/780 CPU time, exclusive of compile time. Since the transformations used in IGGLOSS are identical to those used in BGLOSS (the simulation of the Bendix BDX-930) including the BLISS code, it can be expected that the simulation speed of IGGLOSS, with respect to memory, is comparable to that of BGLOSS.

5.0 TASKS REMAINING TO COMPLETE GGLOSS

As indicated previously, IGGLOSS was developed as an interim version of GGLOSS and, consequently, lacks many features which were intended to be included in GGLOSS. Since IGGLOSS was essentially designed as the "core" element of GGLOSS, IGGLOSS can be transformed into GGLOSS by adding those omitted features. The additional features are:

1) Network partitioning
2) IC specifications
3) Extended test pin coverage
4) Identification of device pins
5) Simulation of intermittent faults
6) Simulation of multiple faults in a single circuit
7) Fault collapsing
8) Simulation of 32 different ROMS
9) Simulation of RAM faults
10) Accommodation of multiple fault models
11) Addition of multiple, fictitious clocks
12) Interfacing with circuit capture programs
13) Expanded library of Bliss-coded primitive circuits
14) Hierarchial network construction
15) Error diagnostic routines
16) Output options
17) Menu option
18) Graphical displays
19) Expanded user's manual
20) Automatic selection of faults
21) Automatic statistical analysis

1. Network Partitioning

In order to simulate a faulted gate IGGLOSS replaces the gate by a fault model (Figure B-3) which contains several gates, the quantity depending upon the type of gate and the number of nodes.

Since IGGLOSS simulates 32 circuits in parallel, one of which is always the non-faulted circuit, replacing more than 31 gates during a run could be inefficient. In fact, IGGLOSS simultaneously replaces all gates by their fault models, irrespective of the number and location of the faults actually simulated. This increased the simulation time proportionately. On the other hand, there is a real time penalty associated with the replacement of gates during a simulation. Consequently, there is a trade-off between the number of faulted gates in each run versus the overhead penalty. GGLOSS should provide the option of optimally partitioning the network into faulted and non-faulted gates during each run.

2. IC Specifications

IGGLOSS automatically sets all nodes to logic 0's at the start of a simulation run. GGLOSS should allow the User to specify the IC's and provide default modes which set the IC's to either all logic 0's or all logic 1's.

3. Extended Test Pin Coverage

IGGLOSS tracks each fault until the fault is detected, at which time it identifies the test pin and the clock cycle. Thereafter it no longer tracks the coverage of the fault (possibly by different test pins). GGLOSS should continuing tracking the fault, identifying all test pins which detected the fault and, of course, the clock cycle during which the fault was detected.

4. Identification of Device Pins

GGLOSS should provide the capability of restricting faults to device pins. It should, therefore, allow the User to identify device pins, preferably in the Partslist.

5. Simulation of Intermittent Faults

IGGLOSS can only simulate permanent faults; i.e., a fault must remain in place for the duration of the run. GGLOSS should provide for the insertion of intermittent faults.

6. Simulation of Multiple Faults In A Single Circuit

IGGLOSS can only simulate one fault at a time in the same circuit. GGLOSS should provide the option of inserting multiple faults in the same circuit.

7. Fault Collapsing

IGGLOSS simulates every fault designated by the User. This is somewhat inefficient because different faults of the same gate could produce identical effects at the output; e.g., a s-a-o of an input to an AND gate produces the same effect as a s-a-o of the output. It would be more efficient to simulate only one and these "equivalent" faults and multiply the results by the number of equivalent faults. GGLOSS should identify such faults and eliminate their redundant simulation.

8. Simulation of 32 Different ROMS

While IGGLOSS can simulate 32 ROMS in parallel it requires that all of the ROMS have the same contents. GGLOSS should provide the option of 32 different ROMS.

9. Simulation of RAM Faults

IGGLOSS simulates 32 RAMS in parallel but does permit the simulation of faults in the RAMS. GGLOSS should provide this option.

10. Accommodation of Multiple Fault Models

It is well known (ref. 3) that a single, gate-equivalent circuit cannot model all faults of a real device by single stuck-at faults. In general, several models of the circuit are required. GGLOSS should allow the User to define several different fault models for the same device.

11. Addition of Multiple, Fictitious Clocks

Fictitious clocks are placed in order to break feedback paths. IGGLOSS allows the placement of an arbitrary number of fictitious clocks but requires that they have the same frequency and phase. The User should have the option of specifying fictitious clocks and different frequencies derived from a master clock.

12. Interfacing With Existing Circuit Capture Program

There are several commercially available circuit capture programs which allow the User to create a circuit on a CRT screen, which is then translated into a Partslist. GGLOSS should have the capability of interfacing with one or more of these Partslists.

13. Expanded Library of Bliss-Coded Primitive Circuits

As noted previously, IGGLOSS contains a library of Bliss-coded primitive macro; e.g., AND gates, OR gates, etc. (see Appendix B). The existing library should be expanded to include a greater variety of primitives. In this connection, it is important to observe that the only time a knowledge of Bliss is required is in the creation of these primitives. It is entirely possible to eliminate even this dependence by allowing the User to define primitives via PartsLists (see the Hierarchical Network Construction Feature).

14. Hierarchical Network Construction

IGGLOSS does not allow a Partslist to contain another Partslist; i.e., a component device cannot be defined by a Partslist. When it is desired to combine circuits the User must manually create a single Partslist for the combined circuits.

GGLOSS should provide the option of defining a circuit component by a Partslist. This provision would, in addition, allow the User, without any knowledge of Bliss, to create new primitives.

15. Error Diagnostic Routines

IGGLOSS provides almost no clues to the source of errors committed by the User in setting up a simulation. GGLOSS should provide a reasonable level of error diagnosis. A "Help" routine could also be added.

16. Output Options

GGLOSS should provide the User with a variety of selectable outputs, including, as a minimum:

- Number of s-a-o, s-a-1 faults detected in each User-designated circuit component or test pin versus clock cycle
- Identification and location of undetected faults
- Fault list
- Output vector of any User-designated component as a function of clock cycle*
- Input vector as a function of clock cycle*
- Failure detection coverage of s-a-o, s-a-1 and combined faults for any User-designated component
- Directory of the library of Bliss-coded primitives
- Contents of memories at a User-designated clock cycle or as a function of clock cycle*

*Caution should be exercised when outputting vectors as functions of clock cycle since this could significantly increase simulation time.

17. Menu Option

While GGLOSS is intended to be interactive with the User (e.g., by prompts), it would be desirable to provide the option of a graphically displayed menu. This would provide the User with a more compact and comprehensive overview of the data required. A "Help" menu would also be desirable.

18. Graphical Displays

If a graphics screen is available, the User should have the option of receiving output data in graphical formats. The formats would include:

- Histograms of latency, perhaps dynamically changing as a function of clock cycle*
- Input vector as a function of clock cycle*
- Output vector as a function of clock cycle*
- Histograms of test pin coverage
- Graphical representation of the circuit, including location of faults, coverage of components, etc.

*See note in "Output Options"

19. Expanded User's Manual

The existing User's Manual for IGGLOSS should be greatly expanded.

20. Automatic Selection of Faults

IGGLOSS requires that the User select each fault manually. For large-scale fault insertion experiments, in which the objective is to determine coverage, a manual selection of faults is not practical. In these cases the User should have the option of specifying only the number of faults, with GGLOSS making the actual selection. The method of selection would be based on:

- Output data requirements
- Distribution of failure rates over the gates
- Sampling strategy, e.g., stratified sampling

Automatic selection of faults requires that the Partslist be expanded to include failure rates of components.

21. Automatic Statistical Analysis

GGLOSS will have the capability of producing large quantities of data. To be useful, the data must be analyzed and reduced and the results presented to the User in a comprehensible form. The statistical analysis of data is inseparable from the sampling strategy employed. Given a sampling strategy (e.g., stratified sampling), statistical analysis would provide the following data:

- Histograms of latency by components
- Combined histograms
- Detection coverage
- Confidence levels of detection coverage (confidence levels vs. quantity of faults presented to User during set-up)
- Maximum likelihood estimates of time to detection
- Most efficient set of test pins

Statistical analysis could conceivably be used to automatically design an efficient self-test program with perhaps some User intervention and prompts.

6.0 REFERENCES

1. McGough, J., Swern, F., "Measurement of Fault Latency in a Digital Avionic Mini Processor," NASA CR-3462, NASA Langley Research Center, Hampton, Va., October, 1981.

2. McGough, J., Swern, F., "Measurement of Fault Latency in a Digital Avionic Mini Processor," NASA CR-3651, NASA Langley Research Center, Hampton, Va., January, 1983.

3. McGough, J., "Feasibility Study for a Generalized Gate Logic Software Simulator," NASA CR-172159, NASA Langley Research Center, Hampton, Va., July, 1983.

ER13A

APPENDIX A

THE SOFTWARE STRUCTURE OF IGGLOSS

The Software Structure of IGGLOSS is shown in Figure A-1. The basic sub-programs are:

- Inquire. Com
- GLOSS. For

INQUIRE.COM

INQUIRE.COM is the executive for IGGLOSS. It is a command procedure which does the following:

1) Asks the User for the name of the Partslist and copies it into the appropriate file for use with the GGLOSS program.

2) Asks the User for the name of the file containing the inputs to the circuit, if any such file exists, and copies it into the appropriate file for use with the GGLOSS program. All I.C.'s are initialized to zero.

3) Asks the User for the name of the file containing the faulted input data. It then copies this data into the file FLTVAL.DAT which is read in by SET_FLT.

4) Executes GLOSS.FOR

5) Compiles each of the modules created by GLOSS.FOR:

- EXEC.FOR	- PASPIN.FOR
- MAIN.B32	- PASADD.FOR
- DWN.B32	- FLTPRN.FOR
- ZND.B32	- DETECT.B32
- TIM2.FOR	- RWN.B32
- PRINT.FOR (optional)	- PRM.B32
- TIM3.FOR	- MEM.B32

6) Links together the modules just compiled.

7) Starts the simulation by running EXEC.FOR

8) Prints the results of the simulation on the terminal. (optional)

GLOSS.FOR

GLOSS.FOR creates the sub-program which does the actual simulation of the circuit. GLOSS.FOR is considered the 'preprocessor' part of IGGLOS, because it prompts for all the inputs, creates the necessary Bliss modules and sets up the result and output files. GLOSS.FOR accomplishes this in the following way:

1) Reads in and stores LIBRARY.DAT

2) Reads in and stores the Partslist.

3) Parses the Partslist, extracting the following information:

   - types of gates used in the circuit
   - external inputs to the circuit
   - external outputs from the circuit

4) Associates a gate type to each component in the Partslist.

5) Determines the input nets for each component.

6) Determines the output nets for each component.

7) Reads in and stores fault data. Substitutes faulted types for gates to be faulted.

8) For each component, substitutes the component name, its input nets and its output nets into the logical equation for its type as found in the library.

9) P-orders the list created in step (8).

10) Creates EXEC.FOR, which prints out the output column headings and calls MAIN.B32.

11) Creates MAIN.B32, with the information received interactively with the User.MAIN.B32 is a Bliss-coded sub-program which sets up the parameters and calls DWN.B32 and PRINT.FOR

12) Creates DWN.B32, a Bliss-coded sub-program containing the P-ordered list. This sub-program does the actual simulation of the circuit.

13) Creates PRINT.FOR, a FORTRAN sub-program which prints the results of the simulation in a file named OUTPUT.DAT

14) Creates MEM.B32, a Bliss routine which loads the contents of the User-specified memory input file into the simulated memories. If the circuit has a RAM, then the routine INTRAM (contained in MEM.B32) makes 32 copies of the RAM in a "scratch pad" memory.

15) Creates PASPIN.FOR and PASADD.FOR, two routines which copy the Bliss "fault detection" vectors into fortran arrays so that the detected data may be printed out.

16) Creates FLTPRN.FOR, a fortran routine to print the results of the fault detection routines.

17) Creates PRM.B32, a Bliss module which emulates a "read only" memory.

18) Creates RWM.B32, similar to PRM.B32 this module emulates a random access memory.

19) Creates ZND.B32, a Bliss routine to load the faulted inputs for the expanded gates in memory so that they can be substituted for unfaulted inputs during emulation.

20) Creates DETFLT.B32, a Bliss module that contains the routine DETECT which checks the specified "detect points" to see if any faults can be detected.

APPENDIX B

BLISS-CODED PRIMITIVES

The Bliss-Coded Library contains Bliss Macros for the following devices:

DEVICE	INPUTS
Buffer	1
Inverter	1
AND Gate	2,3,4
NAND Gate	2,3,4
OR Gate	2,3
NOR Gate	2,3,4,5
Exclusive OR Gate	2
D-Flip Flop	4
RAM	User-Specified
ROM	User-Specified

The Bliss Library also contains the fault model for each of the above devices. The primitive devices and their fault models are shown in Figures B-1 through B-9.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

USER'S MANUAL

TABLE OF CONTENTS

1.0 SOFTWARE ORGANIZATION

1.1 INQUIRE.COM

INQUIRE.COM is the executive for IGGLOSS. It is a command procedure which does the following:

(1) Asks the user for the name of the partslist and copies it into the appropriate file for use with the GLOSS program.

(2) Asks the user for the name of the file containing the inputs to the circuit, if any such file exists, and copies it into the appropriate file for use with the GLOSS program. All I.C.'s are initialized to zero.

(3) Asks the user for the name of the file containing the faulted input data . It then copies this data into the file FLTVAL.DAT which is read in by SET_FLT.

(4) Executes GLOSS.FOR (see GLOSS).

(5) Compiles each of the modules created by GLOSS.FOR:

- EXEC.FOR
- MAIN.B32
- DWN.B32
- ZND.B32
- TIM2.FOR
- PRINT.FOR (optional)
- TIM3.FOR
- PASPIN.FOR
- PASADD.FOR
- FLTPRN.FOR
- DETECT.B32
- RWM.B32
- PRM.B32
- MEM.B32

(6) Links together the modules just compiled.

(7) Starts the simulation by running EXEC.FOR

(8) Prints the results of the simulation on the terminal. (optional)

1.2 GLOSS.FOR

GLOSS.FOR creates the subprogram which does the actual simulation of the circuit.GLOSS is considered the 'preprocessor' part of IGGLOS, because it prompts for all the inputs, creates the necessary bliss modules and sets up the result and output files. GLOSS accomplishes this in the following way:

(1) Reads in and stores LIBRARY.DAT

(2) Reads in and stores the partslist.

(3) Parses the partslist, extracting the following information:

- types of gates used in the circuit
- external inputs to the circuit
- external outputs from the circuit

(4) Associates a gate type to each component in the partslist.

(5) Determines the input nets for each component.

(6) Determines the output nets for each component.

(7) Reads in and stores fault data. Collapses faults and subtitutes faulted types for gates to be faulted.

(8) For each component, substitutes the component name, its input nets and its output nets into the logical equation for its type as found in the library.

(9) P-orders the list created in step (8).

(10) Creates EXEC.FOR, which prints out the output column headings and calls MAIN.B32.

(11) Creates MAIN.B32, with the information received interactively with the user (see RUNNING IGGLOSS). MAIN.B32 is a BLISS coded subprogram which sets up the parameters and calls DWN.B32 and PRINT.FOR.

(12) Creates DWN.B32, a BLISS coded subprogram containing the P-ordered list. This subprogram does the actual simulation of the circuit.

(13) Creates PRINT.FOR, a FORTRAN subprogram which prints the results of the simulation in a file named OUTPUT.DAT.

(14) Creates MEM.B32 a bliss routine which loads the contents of the user specified memory input file into the simulator memories. If the circuit has a RAM, then the routine INTRAM (contained in MEM.B32) makes 32 copies of the RAM in a 'scratch pad' memory.

(15) Creates PASPIN.FOR and PASADD.FOR, two routines which copy the bliss 'fault detection' vectors into fortran arrays so that the detected data may be printed out.

(16) Creates FLTPRN.FOR ,a fortran routine to print the results of the fault detection routines.

(17) Creates PRM.B32, a bliss module which emulates a read only memory.

(18) Creates RWM.B32, similar to PRM.B32 this module emulates a random access memory.

(19) Creates ZND.B32 , a bliss routine to load the faulted inputs for the expanded gates in memory so that they can be substituted for unfaulted inputs during emulation.

(20) Creates DETFLT.B32 ,a bliss module that contains the routine DETECT which checks the specified 'detect points'(see fault detect list page) to see if any faults can be detected.

2.0 RUNNING IGGLOSS

INQUIRE.COM is the command file which executes the program.

To execute, type:

```
$ @INQUIRE
```

The program will then prompt the user with:

```
Enter full name of file containing partslist:
```

The partslist must be supplied by the user. The partslists presently on file are:

ALU.PRT	(Arithmetic Logic Unit)
BCD.PRT	(BCD Adder)
DWN.PRT	(Downcounter)
DWNOCN.PRT	(Downcounter, reordered, clocked)
MEM3.PRT	(Memory circuit, Rom and Ram)

Respond with the name of one of the partslists.
The program will then prompt the user with:

```
Enter full name of file containing input data, if any:
```

Input data is on file for the ALU and the BCD circuits. The data must be input manually for the DWN circuit. Data on file is:

ALU.DAT	(Arithmetic Logic Unit)
BCD.DAT	(BCD Adder)
MM3INP.DAT	(Memory circuit)

If the user wishes to enter the data manually he should hit the RETURN key. If he wishes to use a file, he should enter the file name.

The program will next prompt as follows:

```
Enter full name of file containing fault input if any:
```

Fault input data is on file for the ALU, DOWNCOUNTER and the MEMORY circuits. Data on file is:

ALUFLT.DAT	(ALU)
DWNFLT.DAT	(Downcounter)
MEM2FLT.DAT	(Memory circuit)

If the user wishes to enter the faulted data by file he should enter the file name. Otherwise hit the return key.

The program will then prompt as follows:

DO YOU WISH A FAULTED RUN ? (Y OR N)

If you do not want a faulted run type <N> otherwise type <Y> and the program will then ask:

FAULT TABLE INPUT MANUAL OR FROM A FILE (M OR F) ?

If you specified a file of faults before and you wish to run those faults then type < F >. If you want to enter your faults manually, you should type < M >. The program then asks you to enter the faults as you would in the partslist.

The program will next prompt you:

What are the output column headings?
Type a '*' if they are to be the same:

It will then list each of the external outputs to the circuit and wait while the user either types a '*' or the new name of the output column. If the user hits the <RETURN> key without typing anything that output column heading will be blank.

The program then asks:

Do you wish to load contents of memory(s) Y/N ?

If 'Y' is typed the program will respond with:

Enter the name of the memory input file.

The user should respond with the full file-name of the file containing the data which is to set up the memory(s).
(see: MEMORY DATA FILE, page 56).

The program next prompts:

Do you wish a print out of the output? (Y or N):

If the response is < Y > then an output file will be created and can be printed out.

Next the program will prompt the user with:

    Input manually or from a file? (M or F)

If an input data file name was entered previously, type an <F>, otherwise type an <M> to indicate that data will be entered by hand. If the data is to be entered via a data file the program will then complete without any more prompts.

If the input is to be entered manually, the program will prompt the user with the following:

    How many cycles do you want to simulate?

Respond with an integer greater than zero.

    What are the initial values for the inputs? (0 or 1)

Each external input pin to the circuit will then be printed on the screen and the user will be able to assign a value of <0> or <1> to it.

The program then asks:

    Do you want to change the input data for any cycle? (Y or N)

If the user responds with a <Y>, the next prompt will be:

    Which cycle do you want to update?

Enter an integer between 2 and the total number of cycles to be simulated, as entered previously. The first cycle will use the initial data already entered.

    What are the input values? (0 or 1 or 9)
        (9 indicates no change in the value)

Each external input pin name to the circuit will once again be printed on the screen and the user will be able to assign a value of <0> or <1> to it. If the user does not want to change the value of the input pin from the last value entered, he should type a <9>.

Again, the user will be prompted as to whether or not he wants to change the input pin values for any cycle. If yes, he must choose a cycle between the last cycle chosen and the final cycle. All other choices will be declared inappropriate.

When there are no more cycles whose inputs values may be changed, or when the user does not want to change the input values on any more cycles the program will then run to termination.
If a print out was desired then the output of the simulation will be printed on the screen. This output will also be placed in a file named OUTPUT.DAT for future reference.
The faulted data (ie whether a fault was detected, and in what machine and what cycle) is not printed out on the terminal, but is stored in a file named DETECTED.DAT.

3.0 INPUT FILES

3.1 LIBRARY.DAT

The Bliss-coded Library contains Bliss macros for the following devices:

Device	Inputs
Buffer	1
Inverter	1
AND Gate	2,3,4
NAND Gate	2,3,4
OR Gate	2,3
NOR Gate	2,3,4,5
EXCLUSIVE OR Gate	2
D-Flip Flop	4
RAM	User-Specified
ROM	User-Specified

The Bliss Library also contains the fault model for each of the above devices. The primitive devices are shown in Figures B-1 through B-9.

The library contains a description of the gates used in the circuits to be simulated with GLOSS. For each gate, the library contains the name of the gate, the logical name associated with it, those pins which will be identified as inputs to the gate, and those pins which will be identified as outputs to the gate(pin names in the partslist must be identical to the names in the Library). The library has the following format:

```
NAME:P000M;
LOGIC: BUF(%Y,%A);
INPUTS: .A;
OUTPUTS: .Y;

NAME:T000M;
LOGIC: NAND2(%Y,%A,%B);
INPUTS: .A,.B;
OUTPUTS:.Y;

    .
    .
    .
```

```
NAME:T260M;
LOGIC: NOR5(%Y,%A,%B,%C,%D,%E);
INPUTS: .A,.B,.C,.D,.E;
OUTPUTS: .Y;

NAME:ALUMAC;
LOGIC: ALU(%XX,%YY,%AA,%BB,%CC,%DD,%EE,%FF,%GG);
INPUTS: .AA,.BB,.CC,.DD,.EE,.FF,.GG;
OUTPUTS: .XX,.YY;

NAME:CLK;
LOGIC: CLK(%Y,%A,.K+$);
INPUTS: .A;
OUTPUTS: .Y;
```

The "%" sign precedes the pin names into which GLOSS will be substitute the appropriate net.

GLOSS will substitute an integer where a "$" is found, to be used as an index variable in array within the BLISS modules.

The contents of the library as of now are as follows:

```
NAME:P000M;
LOGIC: BUF(%Y,%A);              { BUF is a non-inverting
INPUTS: .A;                       buffer. }
OUTPUTS: .Y;

NAME:T000M;
LOGIC: NAND2(%Y,%A,%B);         { NAND2 is a two-input
INPUTS: .A,.B;                    nand. }
OUTPUTS: .Y;

NAME:P002M;
LOGIC: NINV(%Y,%A);             { NINV is a non-inverting
INPUTS: .A;                       buffer. }
OUTPUTS: .Y;

NAME:T002M;
LOGIC: NOR2(%Y,%A,%B);
INPUTS: .A,.B;
OUTPUTS: .Y;

NAME:T004M;
LOGIC: NOTT(%Y,%A);             { NOTT is an inverter. }
INPUTS: .A;
OUTPUTS: .Y;
```

```
NAME:T008M;
LOGIC: AND2(%Y,%A,%B);
INPUTS: .A,.B;
OUTPUTS: .Y;

NAME:T010M;
LOGIC: NAND3(%Y,%A,%B,%C);      { NAND3 is a three input nand.}
INPUTS: .A,.B,.C;
OUTPUTS: .Y;

NAME:T011M;
LOGIC: AND3(%Y,%A,%B,%C);       { AND3 is a three input AND. }
INPUTS: .A,.B,.C;
OUTPUTS: .Y;

NAME:T020M;
LOGIC: NAND4(%Y,%A,%B,%C,%D);
INPUTS: .A,.B,.C,.D;
OUTPUTS: .Y;

NAME:T021M;
LOGIC: AND4(%Y,%A,%B,%C,%D);            { AND4 is a four input
INPUTS: .A,.B,.C,.D;                      AND. }
OUTPUTS: .Y;

NAME:T025M;
LOGIC: NOR4(%Y,%A,%B,%C,%D);            { NOR4 is a four input
INPUTS: .A,.B,.C,.D;                      NOR. }
OUTPUTS: .Y;

NAME:T027M;
LOGIC: NOR3(%Y,%A,%B,%C);
INPUTS: .A,.B,.C;
OUTPUTS: .Y;

NAME:T032M;
LOGIC: OR2(%Y,%A,%B);
INPUTS: .A,.B;
OUTPUTS: .Y;

NAME:C050F;
LOGIC: FNIB(%Y,%A);             { FNIB is a non-inverting
INPUTS: .A;                       buffer. }
OUTPUTS: .Y;

NAME:C050L;
LOGIC: LNIB(%Y,%A);             { LNIB is a non-inverting
INPUTS: .A;                       buffer. }
OUTPUTS: .Y;
```

```
NAME:T074M;              { DFF is a D-flip flop. }
LOGIC: DFF(%Q,%QB,%CK,%D,%PR,%CLR,.K+$);
INPUTS: .CK,.D,.PR,.CLR;
OUTPUTS: .Q,.QB;

NAME:C075M;
LOGIC: OR3(%Y,%A,%B,%C);
INPUTS: .A,.B,.C;
OUTPUTS: .Y;

NAME:T086M;
LOGIC: XOR2(%Y,%A,%B);           { XOR2 is a two input
INPUTS: .A,.B;                     exclusive OR. }
OUTPUTS: .Y;

NAME:T260M;
LOGIC: NOR5(%Y,%A,%B,%C,%D,%E); { NOR5 is a five input NOR. }
INPUTS: .A,.B,.C,.D,.E;
OUTPUTS: .Y;

NAME:FT002M;                     { NAND2F is a faulted two input
LOGIC: NAND2F(%Y,%A,%B,%FA,%FB,%FO,%FP);        - nand. }
INPUTS: .A,.B,.FA,.FB,.FO,.FP;
OUTPUTS: .Y;

NAME:FT002M;
LOGIC: NOR2F(%Y,%A,%B,%FA,%FB,%FO,%FP);
INPUTS: .A,.B,.FA,.FB,.FO,.FP;  { NOR2F is a faulted two input
OUTPUTS: .Y;                      NOR. }

NAME:FT008M;
LOGIC: AND2F(%Y,%A,%B,%FA,%FB,%FO,%FP);
INPUTS: .A,.B,.FA,.FB,.FO,.FP;
OUTPUTS: .Y;

NAME:FT032M;
LOGIC: OR2F(%Y,%A,%B,%FA,%FB,%FO,%FP);
INPUTS: .A,.B,.FA,.FB,.FO,.FP;
OUTPUTS: .Y;

NAME:FT027M;
LOGIC: NOR3F(%Y,%A,%B,%C,%FA,%FB,%FC,%FO,%FP);
INPUTS: .A,.B,.C,.FA,.FB,.FC,.FO,.FP;
OUTPUTS: .Y;

NAME:FT025M;
LOGIC: NOR4F(%Y,%A,%B,5C,%D%FA,%FB,%FC,%FD,%FO,%FP);
INPUTS: .A,.B,.C,.D,.FA,.FB,.FC,.FD,.FO,.FP;
OUTPUTS: .Y;
```

```
NAME:FT260M;
LOGIC: NOR5F(%Y,%A,%B,%C,%D,%E,%FA,%FB,%FC,%FD,%FE,%FO,%FP);
INPUTS: .A,.B,.C,.D,.E,.FA,.FB,.FC,.FD,.FE,.FO,.FP;
OUTPUTS: .Y;

NAME:FT011M;
LOGIC: AND3F(%Y,%A,%B,%C,%FA,%FB,%FC,%FO,%FP);
INPUTS: .A,.B,.C,.FA,.FB,.FC,.FO,.FP;
OUTPUTS: .Y;

NAME:FT021M;
LOGIC: AND4F(%Y,%A,%B,%C,%D,%FA,%FB,%FC,%FD,%FO,%FP);
INPUTS: .A,.B,.C,.D,.FA,.FB,.FC,.FD,.FO,.FP;
OUTPUTS: .Y;

NAME:FT010M;
LOGIC: NAND3F(%Y,%A,%B,%C,%FA,%FB,%FC,%FO,%FP);
INPUTS: .A,.B,.C,.FA,.FB,.FC,.FO,.FP;
OUTPUTS: .Y;

NAME:FT020M;
LOGIC: NAND4F(%Y,%A,%B,%C,%D,%FA,%FB,%FC,%FD,%FO,%FP);
INPUTS: .A,.B,.C,.D,.FA,.FB,.FC,.FD,.FO,.FP;
OUTPUTS: .Y;

NAME:FC075M;
LOGIC: OR3F(%Y,%A,%B,%C,%FA,%FB,%FC,%FO,%FP);
INPUTS: .A,.B,.C,.FA,.FB,.FC,.FO,.FP;
OUTPUTS: .Y;

NAME:FT086M;
LOGIC: XOR2F(%Y,%A,%B,%FA,%FB,%FO,%FP);
INPUTS: .A,.B,.FA,.FB,.FO,.FP;
OUTPUTS: .Y;

NAME:FP000M;
LOGIC: FBUFF(%Y,%A,%FO,%FP);
INPUTS: .A,.FO,.FP;
OUTPUTS: .Y;

NAME:FP002M;
LOGIC: NINVF(%Y,%A,%FO,%FP);
INPUTS .A,.FO,.FP;
OUTPUTS: .Y;

NAME:FT004M;
LOGIC: NOTTF(%Y,%A,%FO,%FP);
INPUTS: .A,.FO,.FP;
OUTPUTS: .Y;
```

```
NAME:FC050F;
LOGIC: FNIBF(%Y,%A,%FO,%FP);
INPUTS: .A,.FO,.FP;
OUTPUTS: .Y;

NAME:FC050L;
LOGIC: LNIBF(%Y,%A,%FO,%FP);
INPUTS: .A,.FO,.FP;
OUTPUTS: .Y;
```

3.2 RTNES.R32

RTNES.R32 contains the macro for each gate defined in the library. The macro expansion occurs during the compilation of the BLISS coded programs created by the GLOSS.FOR (The variables in this file do not need to exactly match those in the Library, but the partslist and Library names must match exactly. The order of the variables is pertinent).
The macros are formatted as follows:

```
MACRO

    BUF(R1,I1)=
        R1= .I1 %,

    NAND2(R1,I1,I2)=
        R1= NOT (.I1 AND .I2 ) %,

                .
                .
                .

    NOR5(R1,I1,I2,I3,I4,I5)=
        R1= NOT ((((.I1 OR .I2) OR .I3) OR .I4) OR .I5) %,

    ALU(X,Y,A,B,C,D,E,F,G)=              { ALU is a macro to
        BEGIN                              simulate the logic of
          LOCAL OR1Y,OR2Y,OR3Y,OR4Y;       an ALU device. }
          OR3(OR1Y,A,B,C);
          OR3(OR2Y,D,B,E);
          OR2(OR3Y,D,F);
          OR2(OR4Y,A,G);
          NAND2(X,OR1Y,OR2Y);
          NAND3(Y,OR3Y,B,OR4Y);
        END%,

    CLK(R1,I1,L)=
        ST[L]= .I1 %;
```

These are the Bliss macros that are used by the Bliss routines.

```
MACRO
    BUF(R1,I1)=
        R1= .I1 %,

    NAND2(R1,I1,I2)=
        R1= NOT (.I1 AND .I2 ) %,

    NINV(R1,I1)=
        R1= .I1 %,

    NOR2(R1,I1,I2)=
        R1= NOT (.I1 OR .I2) %,

    NOTT(R1,I1)=
        R1= NOT .I1 %,

    AND2(R1,I1,I2)=
        R1= .I1 AND .I2 %,

    NAND3(R1,I1,I2,I3)=
        R1= NOT ((.I1 AND .I2) AND .I3) %,

    NAND4(R1,I1,I2,I3,I4)=
        R1= NOT ((.I1 AND .I2) AND .I3) AND .I4 %,

    AND3(R1,I1,I2,I3)=
        R1= (.I1 AND .I2) AND .I3 %,

    AND4(R1,I1,I2,I3,I4)=
        R1= ((.I1 AND .I2) AND .I3) AND .I4 %,

    NOR4(R1,I1,I2,I3,I4)=
        R1= NOT (((.I1 OR .I2) OR .I3) OR .I4) %,

    NOR3(R1,I1,I2,I3)=
        R1= NOT ((.I1 OR .I2) OR .I3) %,

    OR2(R1,I1,I2)=
        R1= .I1 OR .I2 %,

    FNIB(R1,I1)=
        R1= .I1 %,

    LNIB(R1,I1)=
        R1= .I1 %,
```

```
DFF(R1,R2,I1,I2,I3,I4,L)=
    R1= (((.ST[L] AND (NOT .I1) OR .I2) AND .I1) AND .I4) OR (NOT .I3) ;
    R2= (NOT .R1) OR (NOT .I4) ;
    ST[L]= .R1 ;
    ST[L+1]= .R2 %,

OR3(R1,I1,I2,I3)=
    R1= .I1 OR .I2 OR .I3 %,


XOR2(R1,I1,I2)=
    R1= (.I1 AND (NOT .I2)) OR (.I2 AND (NOT .I1)) %,

NOR5(R1,I1,I2,I3,I4,I5)=
    R1= NOT ((((.I1 OR .I2) OR .I3) OR .I4) OR .I5) %,

AND2F(R1,I1,I2,FA,FB,FO,FP)=
    R1= (((.I1 OR .FA) AND (.I2 OR .FB)) AND NOT .FO) OR .FP %,

OR2F(R1,I1,I2,FA,FB,FO,FP)=
     R1= (((.I1 AND NOT .FA) OR (.I2 AND NOT .FB)) AND NOT .FO) OR .FP %,

NAND2F(R1,I1,I2,FA,FB,FO,FP)=
      R1=((NOT((.I1 OR .FA) AND (.I2 OR .FB))) AND NOT .FO) OR .FP %,

NOR2F(R1,I1,I2,FA,FB,FO,FP)=
     R1=((NOT (( .I1 AND NOT .FA) OR (.I2 AND NOT .FB))) AND NOT .FO)
            OR .FP %,

NOR3F(R1,I1,I2,I3,FA,FB,FC,FO,FP)=
     R1=((NOT (( .I1 AND NOT .FA) OR ( .I2 AND NOT .FB) OR
    (.I3 AND NOT .FC))) AND NOT .FO) OR .FP %,

NOR4F(R1,I1,I2,I3,I4,FA,FB,FC,FD,FO,FP)=
     R1= ((NOT (( .I1 AND NOT .FA) OR ( .I2 AND NOT .FB)
     OR (.I3 AND NOT .FC) OR ( .I4 AND NOT .FD))) AND NOT .FO) OR .FP%,

NOR5F(R1,I1,I2,I3,I4,I5,FA,FB,FC,FD,FE,FO,FP)=
     R1= ((NOT (( .I1 AND NOT .FA) OR (.I2 AND NOT .FB)
     OR (.I3 AND NOT .FC) OR ( .I4 AND NOT .FD) OR (.I5 AND NOT .FE)))
     AND NOT .FO) OR .FP%,

AND3F(R1,I1,I2,I3,FA,FB,FC,FO,FP)=
     R1= ((( .I1 OR .FA) AND ( .I2 OR .FB) AND ( .I3 OR .FC)) AND NOT .FO)
            OR .FP%,
```

```
AND4F(R1,I1,I2,I3,I4,FA,FB,FC,FD,FO,FP)=
     R1= (( .I1 OR .FA) AND ( .I2 OR .FB) AND ( .I3 OR .FC) AND
          ( .I4 OR .FD)) AND NOT .FO OR .FP %,

NAND3F(R1,I1,I2,I3,FA,FB,FC,FO,FP)=
     R1= ((NOT (( .I1 OR .FA) AND (.I2 OR .FB) AND (.I3 OR .FC)))
           AND NOT .FO) OR .FP%,

NAND4F(R1,I1,I2,I3,I4,FA,FB,FC,FD,FO,FP)=
     R1= ((NOT (( .I1 OR .FA) AND (.I2 OR .FB) AND (.I3 OR .FC)
           AND ( .I4 OR .FD))) AND NOT .FO) OR .FP%,


OR3F(R1,I1,I2,I3,FA,FB,FC,FO,FP)=
     R1= (( .I1 AND NOT .FA) OR ( .I2 AND NOT .FB)
         OR (.I3 AND NOT .FC)) AND NOT .FO OR .FP%,

XOR2F(R1,I1,I2,FA,FB,FO,FP)=
     R1= ((( .I1 AND NOT .FA) AND NOT ( .I2 AND NOT .FB)) OR
          (( .I2 AND NOT .FB) AND NOT ( .I1 AND NOT .FA)))
           AND NOT .FO OR .FP%,


FBUFF(R1,I1,FO,FP)=
     R1= (.I1 AND NOT .FO) OR .FP%,

NINVF(R1,I1,FO,FP)=
     R1= (.I1 AND NOT .FO) OR .FP%,


NOTTF(R1,I1,FO,FP)=
     R1= NOT ((.I1 AND NOT .FP) OR .FO) %,

FNIBF(R1,I1,FO,FP)=
     R1= (.I1  AND NOT .FO) OR .FP%,

LNIBF(R1,I1,FO,FP)=
     R1= (.I1 AND NOT .FO) OR .FP%,

ALU(X,Y,A,B,C,D,E,F,G)=
    BEGIN
      LOCAL OR1Y,OR2Y,OR3Y,OR4Y;
      OR3(OR1Y,A,B,C);
      OR3(OR2Y,D,B,E);
      OR2(OR3Y,D,F);
      OR2(OR4Y,A,G);
      NAND2(X,OR1Y,OR2Y);
      NAND3(Y,OR3Y,B,OR4Y);
    END%,

CLK(R1,I1,L)=
    ST[L]= .I1 %;
```

```
-- This is a keyword macro to simulate memory devices. It accepts from 1 to 16
-- address parameters, from 1 to 16 output parameters and from 1 to 16 data
-- parameters. Unused address,data,or output parameters are set to 'NINE'
-- (which equals 99) and are later set to 0 for the computations involving
-- full word values.

KEYWORDMACRO
MEMR(Y0=NINE,Y1=NINE,Y2=NINE,Y3=NINE,Y4=NINE,Y5=NINE,Y6=NINE,Y7=NINE,Y8=NINE,
        Y9=NINE,Y10=NINE,Y11=NINE,Y12=NINE,Y13=NINE,Y14=NINE,Y15=NINE,Y16=NINE,
        A0=NINE,A1=NINE,A2=NINE,A3=NINE,A4=NINE,A5=NINE,A6=NINE,A7=NINE,
        A8=NINE,A9=NINE,A10=NINE,A11=NINE,A12=NINE,A13=NINE,A14=NINE,A15=NINE,
        A16=NINE,EN=NINE,RW=NINE,D0=NINE,D1=NINE,D2=NINE,D3=NINE,D4=NINE,
        D5=NINE,D6=NINE,D7=NINE,D8=NINE,D9=NINE,D10=NINE,D11=NINE,D12=NINE,
        D13=NINE,D14=NINE,D15=NINE,D16=NINE) =
        BEGIN

         LOCAL JJ,KK;
         KK = 0;
         IF (.A0 NEQ 99) THEN
           BEGIN
                ADDARY[.KK] = .A0;
                KK = .KK + 1;
           END;
         IF (.A1 NEQ 99) THEN
           BEGIN
                ADDARY[.KK] = .A1;
                KK = .KK + 1
         END;
         IF (.A2 NEQ 99) THEN
           BEGIN
                ADDARY[.KK] = .A2;
                KK = .KK + 1
         END;
         IF (.A3 NEQ 99) THEN
           BEGIN
                ADDARY[.KK] = .A3;
                KK = .KK + 1
         END;
         IF (.A4 NEQ 99) THEN
           BEGIN
                ADDARY[.KK] = .A4;
                KK = .KK + 1
         END;
         IF (.A5 NEQ 99) THEN
           BEGIN
                ADDARY[.KK] = .A5;
                KK = .KK + 1
         END;
         IF (.A6 NEQ 99) THEN
           BEGIN
                ADDARY[.KK] = .A6;
                KK = .KK + 1
         END;
```

```
IF (.A7 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A7;
        KK = .KK + 1
END;
IF (.A8 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A8;
        KK = .KK + 1
END;
IF (.A9 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A9;
        KK = .KK + 1
END;
IF (.A10 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A10;
        KK = .KK + 1
END;
IF (.A11 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A11;
        KK = .KK + 1
END;
IF (.A12 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A12;
        KK = .KK + 1
END;
IF (.A13 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A13;
        KK = .KK + 1
END;
IF (.A14 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A14;
        KK = .KK + 1
END;
IF (.A15 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A15;
        KK = .KK + 1
END;
IF (.A16 NEQ 99) THEN
  BEGIN
        ADDARY[.KK] = .A16;
        KK = .KK + 1;
END;
```

```
 NAB = .KK - 1;
 RWB = .RW;
 ENB = .EN;
 KK = 0;

IF (.D0 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D0;
        KK = .KK + 1;
  END;
IF (.D1 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D1;
        KK = .KK + 1;
  END;
IF (.D2 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D2;
        KK = .KK + 1;
  END;
IF (.D3 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D3;
        KK = .KK + 1;
  END;
IF (.D4 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D4;
        KK = .KK + 1;
  END;


IF (.D5 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D5;
        KK = .KK + 1;
  END;
IF (.D6 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D6;
        KK = .KK + 1;
  END;
IF (.D7 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D7;
        KK = .KK + 1;
  END;
IF (.D8 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D8;
        KK = .KK + 1;
  END;
```

```
IF (.D9 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D9;
        KK = .KK + 1;
  END;
IF (.D10 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D10;
        KK = .KK + 1;
  END;
IF (.D11 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D11;
        KK = .KK + 1;
  END;
IF (.D12 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D12;
        KK = .KK + 1;
  END;
IF (.D13 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D13;
        KK = .KK + 1;
  END;
IF (.D14 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D14;
        KK = .KK + 1;
  END;
IF (.D15 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D15;
        KK = .KK + 1;
  END;
IF (.D16 NEQ 99) THEN
  BEGIN
        DATARY[.KK] = .D16;
        KK = .KK + 1;
  END;
IF .KK EQL 0 THEN
   NDB = .KK
ELSE
   NDB = .KK - 1;

 IF .RWB NEQ 99 THEN
   BEGIN
        RWMEM();
   END
 ELSE
   BEGIN
        PRMS();
 END;

 KK = 0;
```

```
IF (.Y0 NEQ 99) THEN
  BEGIN
       Y0 = .OUTARY[.KK];
       KK = .KK + 1;
  END;
IF (.Y1 NEQ 99) THEN
  BEGIN
       Y1 = .OUTARY[.KK];
       KK = .KK + 1;
  END;
IF (.Y2 NEQ 99) THEN
  BEGIN
       Y2 = .OUTARY[.KK];
       KK = .KK + 1;
  END;
IF (.Y3 NEQ 99) THEN
  BEGIN
       Y3 = .OUTARY[.KK];
       KK = .KK + 1;
  END;
IF (.Y4 NEQ 99) THEN
  BEGIN
       Y4 = .OUTARY[.KK];
       KK = .KK + 1;
  END;
IF (.Y5 NEQ 99) THEN
  BEGIN
       Y5 = .OUTARY[.KK];
       KK = .KK + 1;
  END;

IF (.Y6 NEQ 99) THEN
  BEGIN
       Y6 = .OUTARY[.KK];
       KK = .KK + 1;
  END;
IF (.Y7 NEQ 99) THEN
  BEGIN
       Y7 = .OUTARY[.KK];
       KK = .KK + 1;
  END;
IF (.Y8 NEQ 99) THEN
  BEGIN
       Y8 = .OUTARY[.KK];
       KK = .KK + 1;
  END;
IF (.Y9 NEQ 99) THEN
  BEGIN
       Y9 = .OUTARY[.KK];
       KK = .KK + 1;
  END;
```

```
 IF (.Y10 NEQ 99) THEN
   BEGIN
        Y10 = .OUTARY[.KK];
        KK = .KK + 1;
   END;
 IF (.Y11 NEQ 99) THEN
   BEGIN
        Y11 = .OUTARY[.KK];
        KK = .KK + 1;
   END;
 IF (.Y12 NEQ 99) THEN
   BEGIN
        Y12 = .OUTARY[.KK];
        KK = .KK + 1;
   END;
 IF (.Y13 NEQ 99) THEN
   BEGIN
        Y13 = .OUTARY[.KK];
        KK = .KK + 1;
   END;
 IF (.Y14 NEQ 99) THEN
   BEGIN
        Y14 = .OUTARY[.KK];
        KK = .KK + 1;
   END;
 IF (.Y15 NEQ 99) THEN
   BEGIN
        Y15 = .OUTARY[.KK];
        KK = .KK + 1;
   END;
 IF (.Y16 NEQ 99) THEN
   BEGIN
        Y16 = .OUTARY[.KK]
   END
END%;
```

3.3 PARTSLIST

The partslist describing the circuit must be of the following format:

```
(* preliminary declarations *)
(The declarations must be given in the following order)
USER:       "NEMEROFF";
(Every partslist is required to have a user name)
NAME:       DWNCNTG;
(A name is required for every circuit)
PURPOSE:    TEGATE;
(The purpose declaration is optional)
LEVEL:      CHIP;
(The level declaration is optional)
(* declare logical types *)
TYPES:      NAND, INV;
(Every component is required to have a logical type)
(* declare the external connectors *)
EXT::       X, Y, D3, D2, D1, D0;
INPUTS:     .X, .Y;
OUTPUTS:    .D0, .D1, .D2, .D3;
(* declare components *)
NAND:       NAND1, NAND2, NAND3, NAND4;
INV:        NOT1, NOT2;
(Every component must be identified as a logical type)
END;
(* End of preliminary declaration section *)
COMPSEGMENT;
(This segment defines the circuit connectivity)
(* declare main components with input and output buffers *)
      = X*N1, Y*N2, D0*ND0
        D1*ND1;
(* declare inner components *)
NAND1 = A*N2, B*N4, Y*ND0;
NAND2 = A*N2, B*N3, Y*ND1;
NAND3 = A*N4, B*N1, Y*ND2;      (* order is not important *)
NAND4 = A*N1, B*N3, Y*ND3;
NOT1  = A*N1, Y*N2;
NOT2  = A*N3, Y*N4;
ENDCOMPS;
(* End of COMPSEGMENT *)
(* Enter fault detection pins or memory locations *)
DETECTSEGMENT;
PINS;
(These two declarations are not optional even if no
 faults are to be simulated. If there are no faults to
 be entered in the fault list, then type:)
ENDC;
END_OF_FILE;
(Otherwise see detect Point List, page 52)
```

Comments:

The order in which the circuit components are listed in the COMPSEGMENT is arbitrary.

Net names must be legal Bliss variables.

Each component of the compsegment is defined as follows:

component = pin*net <, pin*net, ...> .

Names of input and output pins of a component must be identical to the corresponding names, as defined in LIBRARY.DAT, pages 8-13. The pins are identified as either input pins or output pins by referring to the specific part in the library.

The partslist does not distinguish between upper and lower case characters.

The external connectors have input or output buffers, the nets of which connect to the other parts of the circuit. Every input and output pin is required to be associated with a buffer. If real buffers do not exist, then fictitious buffers must be inserted.

3.4 MEMORY PARTS

If you have memory devices to be included into the partslist, they are to be entered in this format:

1. Under the 'TYPES' declaration "MEMR" is the memory type.

2. In the 'COMPSEGMENT', declaration of the part number (for this example let us choose 'U3') is declared as follows:

```
U3      = A0*U7Y,  A1*U43Y,  A'n'*U15Y,
          EN*U9Y,  RW*U77Y,  D0*U64Y,
          D1*U90Y, D'n'*U2Y, Y0*U3Y0,
          Y1*U3Y1, Y'n'*U3Y'n';

where   A0...A'n' = Address bits.
        EN        = Enable bit.
        RW        = Read/Write bit(For RAM, only).
        D0...D'n' = Data bits.
        Y0...Y'n' = Output bits.

The output nets of the part (U3) become:
        U3Y0...U3Y'n'.
```

3.5 DETECT POINT LIST

The user has the ability to declare certain pins or memory addresses as 'detect points'. These detect points are locations which are watched by IGLOSS and tested at the end of each cycle to determine if a fault was detected. A detected fault is defined as a discrepancy between the 0 bit (the unfaulted machine) and any of the other 31 bits (possibly faulted machines).

The declaration of 'detect points' is done in the partslist.

Following the 'ENDCOMPS;' the detect list is described using this format:

```
DETECTSEGMENT;
PINS;              (* declare the pins to be watched *)
        U1Y;
        U20Y;
        U33Y;
        U34Y;
ADDRESSES;         (* declare hex memory addresses to be
                      watched *)
        00002;
        00010;
        00016;
        00018;
ENDDET;
```

The pins that are declared, are the output nets of the part number. The addresses declared are any location in user defined memories.

The file is then closed in this manner :

```
ENDC;
END_OF_FILE;
```

3.6 FAULT INPUT LIST

If the user wishes to choose the gates to be faulted, he must enter them
in a file which he names. When IGGLOSS is run it will ask for the filename and copy it into a file called <FLTVAL.DAT>.

To declare the faultlist use the following format:

First delcare the parts to be faulted:

```
(* declare faults *)
PINFLTS;
U32:        Y*   0;
U14:        A*   1;
NOR54:      B*   0;
NOT1:       Y*   0;
NOT4:       Y*   1;
U22:        A*   1;
ENDPINS;
```

Then if a fault in ROM is desired, declare the words in memory and the bits to be faulted like this:

```
MEMFLTS;
000001*    4;
000235*    25;
000236*    16;
ENDMEM;
```

(The memory addresses are in hex; bit positions, in decimal)

Then the file is ended in this manner:

```
ENDC;
END_OF_FILE;
```

In this form 'U32' is the part name, 'Y' is the pin name, and 0 or 1 is the stuck at zero or stuck at one fault.

In this form '000001' is the word in ROM which will be faulted. 4 is the bit of this word which will be flipped from 0 to 1 or from 1 to 0 in order to create the memory fault. Since '000001' appears as the first word in this memory fault list, it becomes the faulted value of that word in the first faulted machine. The faulted value of '000235' becomes the fault in the second faulted machine and the rest follow consecutively.

3.7 DETECTED FAULTS OUTPUT LIST

Once the user has chosen test points, faults and input sequence, IGGLOSS will, at each cycle, identify the first detect point in the list at which the fault is detected.
Only the first detect point at which the fault is discovered is printed out by IGGLOSS.

example:
ALU -- page 75

FAULT NAME	TEST PIN #	CYCLE #
INV3	INV3Y	1
INV35	NOR51Y	1
INV4	INV4Y	1
INV5	XOR49	9

(* Fault INV3 is detected at detect point INV3Y on the first cycle. *)

Further modification of IGGLOSS to identify all detect points corresponding to a given fault is a relatively easy task.

3.8 FICTITIOUS CLOCKS

In the event that the circuit to be simulated has a feedback loop, the partslist will be unorderable. In this case, the user must choose which nets in the circuit to "break" in order to eliminate all the loops. Fictitious clocks must be inserted on each net which is to be broken. The following procedure should be followed to add the fictitious gates to the partslist.(It is noted that fictitious clocks are treated like devices in their interconnections in the partslist).

(1) In the section of the partslist where the logical types are declared, add type CLK to the list.
ex: TYPES: NAND,INV,CLK;

(2) In the section in which the components are declared, add type CLK and enough components to correspond to the number of "ficticious" gates needed.
ex:

```
.
.
.

INV:    NOT1, NOT2;
CLK:    C1, C2, ... Cn;
```

(3) In the section of the partslist between the words "COMPSEGMENT;" and "ENDCOMPS;", for each "ficticious gate", choose an "output net" to correspond to the "broken net" and list:
component = A*"broken net", Y*"output net";

ex:

```
COMPSEGMENT;
        .
        .

C1      =  A*U35Y,      Y*C1Y;
C2      =  A*U36Y,      Y*C2Y;
        .
        .

Cn      =  A*U45Y,      Y*CnY;
        .
        .

ENDCOMPS;
```

For an actual example of "fictitious gates", see the example of the Downcounter.

3.9 MEMORY DATA FILES

The data which controls the contents,size and range of the memory devices in the partslist is stored in a file and entered in by the user in response to the prompt:

Enter name of Memory Input File.

The memory input file is to be formatted in the following manner:

```
0,15,0,15,100,000;                                  (* 1 *)
( * "100,000" not used but must be present * )
RADIX,DECIMAL;                                      (* 2 *)
000,000003,000004,000005,000006,000007,000008;      (* 3 *)
006,000009,000010,000011,000012,000013,000014;      (* 4 *)
012,000015,000000,000001,000002;                    (* 5 *)
END
```

The data on line (* 1 *) represents:
- Low data address (i.e., starting decimal location for ROM)
- High data address (i.e., ending decimal location for ROM)
- Low writeable address (i.e., starting decimal location for RAM)
- High writeable address (ie. ending decimal location for RAM)
- "100,000" are not used but must be present

Line (* 2 *) sets the radix to decimal.
Lines (* 3 *)- (* 5 *) consist of the data to be stored in memory with the first number in the line being the address of the piece of data which directly follows it(the line header), and the rest of the data on that line is stored in consecutive addresses (Memory addresses and contents are in decimal format).
When memory is read, the routines will rely on the line headers as the displacement (to be compensated for),when searching for non-consecutive locations in memory.

ex. 000 is the address for the data 000003. then
001 will be the address for 000004

therefore since:

012 is the address for 000015 then
013 is the address for 000000 and
014 is the address for 000001 and
015 is the address for 000002.

The file must close with the word 'END'.

3.10 P-ORDERING

VARS:
- LIST1 = unordered list
- LIST2 = ordered list

INPUT:
- MAIN = position of externals in partslist
- NUM = number of components in partslist

(1) Equate input nets to external input pins and add them to LIST2.

(2) Set FIN = true.

(3) Find the next statement in LIST1 which has not yet been placed in LIST2 (not DONE). If all statements are DONE then go to (10)

(4) Set FIN = false.

(5) Check if the values of all inputs to the statement are known. If not, go to (8).

(6) Check if statement is a ficticious clock. If so, add it to the list of ficticious clocks (LISFC) and go to (8).

(7) Add statement to LIST2.
Set statement = DONE.
Add statement output to list of knowns (KNOWN).

(8) Increment NUM.

(9) If loop was executed more than NUM times, then print an error message and go to (11).

(10) If FIN = true then go to (2).

(11) Add list of ficticious clocks (LISFC) to LIST2.

(12) Equate external output pins to output nets.

4.0 ARITHMETIC LOGIC UNIT

USER INPUT

4.1 PARTSLIST -- ALU(Fig.2)
(ALU.PRT)

```
USER:     "DEUTSCH";
NAME:     ALU;
PURPOSE:  TESTING;
LEVEL:    SUBCHIP;
TYPES:    T004M,ALUMAC,P000M,P002M,T086M,T032M,T002M,T010M,
          T260M,T025M,T020M,T000M,C075M;
EXT: :    SEL1,SEL2,SEL3,SEL4,A1,A2,A3,A4,B1,B2,B3,B4,
          CONTROL,CARIN,
          OUT1,OUT2,OUT3,OUT4,COMPARE,CAROUT,LACO1,LACO2;
INPUTS:   .SEL1,.SEL2,.SEL3,.SEL4,.A1,.A2,.A3,.A4,.B1,.B2,.B3,
          .B4,.CONTROL,.CARIN;
OUTPUTS:  .OUT1,.OUT2,.OUT3,.OUT4,.COMPARE,.LACO1,.LACO2,.CAROUT;
T004M:    INV1,INV2,INV3,INV4,INV5,INV6,INV7,INV8,INV9,INV10,INV11,INV12,
          INV13,INV14,INV15,INV16,INV17,INV18,INV35,INV83,INV82,INV84,INV85,
          INV74,INV75,INV76,INV77,INV78,INV73,INV79,INV80,INV81;
ALUMAC:   ALUMAC1,ALUMAC2,ALUMAC3,ALUMAC4;
P000M:    SEL1,SEL2,SEL3,SEL4,A1,A2,A3,A4,B1,B2,B3,B4,CONTROL,CARIN;
P002M:    OUT1,OUT2,OUT3,OUT4,COMPARE,CAROUT,LACO1,LACO2;
T086M:    XOR49,XOR54,XOR58,XOR61,XOR68,XOR69,XOR70,XOR71;
T032M:    OR46,OR53,OR57,OR60;
T002M:    NOR62,NOR67;
T010M:    NAND65;
T020M:    NAND63,NAND64;
T000M:    NAND66;
C075M:    OR45,OR52,OR56,OR59;
T025M:    NOR44,NOR48,NOR51,NOR55,NOR72;
T260M:    NOR47,NOR50;
END;
COMPSEGMENT;
          = SEL1*ISEL1,SEL2*ISEL2,SEL3*ISEL3,SEL4*ISEL4,A1*IA1,A2*IA2,
            A3*IA3,A4*IA4,B1*IB1,B2*IB2,B3*IB3,B4*IB4,CONTROL*ICONTROL,
            CARIN*ICARIN,OUT1*IOUT1,OUT2*IOUT2,OUT3*IOUT3,OUT4*IOUT4,
            COMPARE*ICOMPARE,LACO1*ILACO1,LACO2*ILACO2,CAROUT*ICAROUT;
INV1      = A*ISEL4,Y*INV1Y;
INV2      = A*ISEL3,Y*INV2Y;
INV3      = A*ISEL2,Y*INV3Y;
INV4      = A*ISEL1,Y*INV4Y;
INV5      = A*IB4,Y*INV5Y;
INV6      = A*IA4,Y*INV6Y;
INV7      = A*IB3,Y*INV7Y;
INV8      = A*IA3,Y*INV8Y;
INV9      = A*IB2,Y*INV9Y;
INV10     = A*IA2,Y*INV10Y;
INV11     = A*IB1,Y*INV11Y;
INV12     = A*IA1,Y*INV12Y;
INV13     = A*INV5Y,Y*INV13Y;
```

```
INV14    = A*INV7Y,Y*INV14Y;
INV15    = A*INV9Y,Y*INV15Y;
INV16    = A*INV11Y,Y*INV16Y;
INV17    = A*ICONTROL,Y*INV17Y;
INV18    = A*ICARIN,Y*INV18Y;
INV35    = A*INV17Y,Y*INV35Y;
ALUMAC1  = AA*INV13Y,BB*INV6Y,CC*INV2Y,DD*INV5Y,EE*INV1Y,FF*INV4Y,
           GG*INV3Y,XX*ALUMAC1X,YY*ALUMAC1Y;
ALUMAC2  = AA*INV14Y,BB*INV8Y,CC*INV2Y,DD*INV7Y,EE*INV1Y,FF*INV4Y,
           GG*INV3Y,XX*ALUMAC2X,YY*ALUMAC2Y;
ALUMAC3  = AA*INV15Y,BB*INV10Y,CC*INV2Y,DD*INV9Y,EE*INV1Y,FF*INV4Y,
           GG*INV3Y,XX*ALUMAC3X,YY*ALUMAC3Y;
ALUMAC4  = AA*INV16Y,BB*INV12Y,CC*INV2Y,DD*INV11Y,EE*INV1Y,FF*INV4Y,
           GG*INV3Y,XX*ALUMAC4X,YY*ALUMAC4Y;
NOR44    =  Y*NOR44Y,A*ALUMAC1X,B*ALUMAC2X,C*ALUMAC3X,D*ALUMAC4Y;
INV83    =  Y*INV83Y,A*NOR44Y;
OR45     =  Y*OR45Y,A*ALUMAC1X,B*ALUMAC2X,C*ALUMAC3Y;
OR46     =  Y*OR46Y,A*ALUMAC1X,B*ALUMAC2Y;
NOR47    =  Y*NOR47Y,A*ALUMAC1X,B*ALUMAC2X,C*ALUMAC3X,D*ALUMAC4X,E*INV18Y;
NOR48    =  Y*NOR48Y,A*ALUMAC1X,B*ALUMAC2X,C*ALUMAC3X,D*ALUMAC4X;
XOR49    =  Y*XOR49Y,A*ALUMAC1X,B*ALUMAC1Y;
NOR50    =  Y*NOR50Y,A*ALUMAC2X,B*ALUMAC3X,C*ALUMAC4X,D*INV18Y,E*INV35Y;
INV82    =  Y*INV82Y,A*NOR50Y;
NOR51    =  Y*NOR51Y,A*ALUMAC2X,B*ALUMAC3X,C*ALUMAC4Y,D*INV35Y;
INV84    =  Y*INV84Y,A*NOR51Y;
OR52     =  Y*OR52Y,A*ALUMAC2X,B*ALUMAC3Y,C*INV35Y;
OR53     =  Y*OR53Y,A*ALUMAC2Y,B*INV35Y;
XOR54    =  Y*XOR54Y,A*ALUMAC2X,B*ALUMAC2Y;
NOR55    =  Y*NOR55Y,A*ALUMAC3X,B*ALUMAC4X,C*INV18Y,D*INV35Y;
INV85    =  Y*INV85Y,A*NOR55Y;
OR56     =  Y*OR56Y,A*ALUMAC3X,B*ALUMAC4Y,C*INV35Y;
OR57     =  Y*OR57Y,A*ALUMAC3Y,B*INV35Y;
XOR58    =  Y*XOR58Y,A*ALUMAC3X,B*ALUMAC3Y;
OR59     =  Y*OR59Y,A*ALUMAC4Y,B*INV18Y,C*INV35Y;
OR60     =  Y*OR60Y,A*INV35Y,B*ALUMAC4Y;
XOR61    =  Y*XOR61Y,A*ALUMAC4X,B*ALUMAC4Y;
NOR62    =  Y*NOR62Y,A*INV18Y,B*INV35Y;
NAND63   =  Y*NAND63Y,A*INV83Y,B*OR45Y,C*OR46Y,D*ALUMAC1Y;
NAND64   =  Y*NAND64Y,A*INV82Y,B*INV84Y,C*OR52Y,D*OR53Y;
NAND65   =  Y*NAND65Y,A*INV85Y,B*OR56Y,C*OR57Y;
NAND66   =  Y*NAND66Y,A*OR59Y,B*OR60Y;
NOR67    =  Y*NOR67Y,A*NAND63Y,B*NOR47Y;
XOR68    =  Y*XOR68Y,A*XOR49Y,B*NAND64Y;
XOR69    =  Y*XOR69Y,A*XOR54Y,B*NAND65Y;
XOR70    =  Y*XOR70Y,A*XOR58Y,B*NAND66Y;
XOR71    =  Y*XOR71Y,A*XOR61Y,B*NOR62Y;
NOR72    =  Y*NOR72Y,A*XOR68Y,B*XOR69Y,C*XOR70Y,D*XOR71Y;
INV73    =  Y*INV73Y,A*NOR72Y;
INV74    =  Y*ILACO1,A*NAND63Y;
INV75    =  Y*ICAROUT,A*NOR67Y;
INV76    =  Y*ILACO2,A*NOR48Y;
INV77    =  Y*IOUT4,A*XOR68Y;
INV78    =  Y*IOUT3,A*XOR69Y;
INV79    =  Y*ICOMPARE,A*INV73Y;
```

```
INV80     =  Y*IOUT2,A*XOR70Y;
INV81     =  Y*IOUT1,A*XOR71Y;
ENDCOMPS;
DETECTSEGMENT;
   PINS;
        INV2Y;
        INV3Y;
        INV4Y;
        XOR49Y;
        NOR50Y;
        INV10Y;
        INV82Y;
        NOR47Y;
        NOR48Y;
        NOR51Y;
        OR52Y;
        OR53Y;
        NAND63Y;
        NAND64Y;
        NAND65Y;
        NAND66Y;
        XOR68Y;
        XOR69Y;
        XOR70Y;
ENDDET;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT

4.2 P-ORDERING -- ALU

(DWN.B32)

```
  ISEL1                          =                          ..SEL1;
ISEL2                            =                          ..SEL2;
ISEL3                            =                          ..SEL3;
ISEL4                            =                          ..SEL4;
IA1                              =                            ..A1;
IA2                              =                            ..A2;
IA3                              =                            ..A3;
IA4                              =                            ..A4;
IB1                              =                            ..B1;
IB2                              =                            ..B2;
IB3                              =                            ..B3;
IB4                              =                            ..B4;
ICONTROL                         =                       ..CONTROL;
ICARIN                           =                         ..CARIN;

 NOTT(INV1Y,ISEL4);
 NOTT(INV2Y,ISEL3);
 NOTTF(INV3Y,ISEL2,ZAND[0000000001],ZERO);
 NOTTF(INV4Y,ISEL1,ZAND[0000000003],ZERO);
 NOTTF(INV5Y,IB4,ZAND[0000000004],ZERO);
 NOTTF(INV6Y,IA4,ZAND[0000000005],ZERO);
 NOTTF(INV7Y,IB3,ZAND[0000000006],ZERO);
 NOTTF(INV8Y,IA3,ZAND[0000000007],ZERO);
 NOTTF(INV9Y,IB2,ZAND[0000000008],ZERO);
 NOTTF(INV10Y,IA2,ZAND[0000000009],ZERO);
 NOTTF(INV11Y,IB1,ZAND[0000000010],ZERO);
 NOTTF(INV12Y,IA1,ZAND[0000000011],ZERO);
 NOTTF(INV13Y,INV5Y,ZAND[0000000012],ZERO);
 NOTTF(INV14Y,INV7Y,ZAND[0000000013],ZERO);
 NOTT(INV15Y,INV9Y);
 NOTT(INV16Y,INV11Y);
 NOTT(INV17Y,ICONTROL);
 NOTT(INV18Y,ICARIN);
 NOTTF(INV35Y,INV17Y,ZAND[0000000002],ZERO);
 ALU(ALUMAC1X,ALUMAC1Y,INV13Y,INV6Y,INV2Y,INV5Y,INV1Y,INV4Y,INV3Y);
```

```
ALU(ALUMAC2X,ALUMAC2Y,INV14Y,INV8Y,INV2Y,INV7Y,INV1Y,INV4Y,INV3Y);
ALU(ALUMAC3X,ALUMAC3Y,INV15Y,INV10Y,INV2Y,INV9Y,INV1Y,INV4Y,INV3Y);
ALU(ALUMAC4X,ALUMAC4Y,INV16Y,INV12Y,INV2Y,INV11Y,INV1Y,INV4Y,INV3Y);
NOR4F(NOR44Y,ALUMAC1X,ALUMAC2X,ALUMAC3X,ALUMAC4Y,ZERO,ZERO,ZERO,ZERO,
ZAND[0000000014],ZERO);
NOTT(INV83Y,NOR44Y);
OR3F(OR45Y,ALUMAC1X,ALUMAC2X,ALUMAC3Y,ZERO,ZERO,ZERO,ZAND[0000000015],
ZERO);
OR2F(OR46Y,ALUMAC1X,ALUMAC2Y,ZERO,ZERO,ZAND[0000000016],ZERO);
NOR5F(NOR47Y,ALUMAC1X,ALUMAC2X,ALUMAC3X,ALUMAC4X,INV18Y,ZERO,ZERO,
ZERO,ZERO,ZERO,ZAND[0000000017],ZERO);
NOR4F(NOR48Y,ALUMAC1X,ALUMAC2X,ALUMAC3X,ALUMAC4X,ZERO,ZERO,ZERO,ZERO,
ZAND[0000000018],ZERO);
XOR2F(XOR49Y,ALUMAC1X,ALUMAC1Y,ZERO,ZERO,ZAND[0000000019],ZERO);
NOR5F(NOR50Y,ALUMAC2X,ALUMAC3X,ALUMAC4X,INV18Y,INV35Y,ZERO,ZERO,ZERO,
ZERO,ZERO,ZAND[0000000020],ZERO);
NOTT(INV82Y,NOR50Y);
NOR4F(NOR51Y,ALUMAC2X,ALUMAC3X,ALUMAC4Y,INV35Y,ZERO,ZERO,ZERO,ZERO,
ZAND[0000000021],ZERO);
NOTT(INV84Y,NOR51Y);
OR3F(OR52Y,ALUMAC2X,ALUMAC3Y,INV35Y,ZERO,ZERO,ZERO,ZAND[0000000022],
ZERO);
OR2F(OR53Y,ALUMAC2Y,INV35Y,ZERO,ZERO,ZAND[0000000023],ZERO);
XOR2F(XOR54Y,ALUMAC2X,ALUMAC2Y,ZERO,ZERO,ZAND[0000000024],ZERO);
NOR4F(NOR55Y,ALUMAC3X,ALUMAC4X,INV18Y,INV35Y,ZERO,ZERO,ZERO,ZERO,
ZAND[0000000025],ZERO);
NOTTF(INV85Y,NOR55Y,ZAND[0000000031],ZERO);
OR3F(OR56Y,ALUMAC3X,ALUMAC4Y,INV35Y,ZERO,ZERO,ZERO,ZAND[0000000026],
ZERO);
OR2F(OR57Y,ALUMAC3Y,INV35Y,ZERO,ZERO,ZAND[0000000027],ZERO);
XOR2F(XOR58Y,ALUMAC3X,ALUMAC3Y,ZERO,ZERO,ZAND[0000000028],ZERO);
OR3F(OR59Y,ALUMAC4Y,INV18Y,INV35Y,ZERO,ZERO,ZERO,ZAND[0000000029],
ZERO);
OR2F(OR60Y,INV35Y,ALUMAC4Y,ZERO,ZERO,ZAND[0000000030],ZERO);
XOR2(XOR61Y,ALUMAC4X,ALUMAC4Y);
NOR2(NOR62Y,INV18Y,INV35Y);
NAND4(NAND63Y,INV83Y,OR45Y,OR46Y,ALUMAC1Y);
NAND4(NAND64Y,INV82Y,INV84Y,OR52Y,OR53Y);
NAND3(NAND65Y,INV85Y,OR56Y,OR57Y);
NAND2(NAND66Y,OR59Y,OR60Y);
NOR2(NOR67Y,NAND63Y,NOR47Y);
XOR2(XOR68Y,XOR49Y,NAND64Y);
XOR2(XOR69Y,XOR54Y,NAND65Y);
XOR2(XOR70Y,XOR58Y,NAND66Y);
XOR2(XOR71Y,XOR61Y,NOR62Y);
NOR4(NOR72Y,XOR68Y,XOR69Y,XOR70Y,XOR71Y);
NOTT(INV73Y,NOR72Y);
NOTT(ILACO1,NAND63Y);
NOTT(ICAROUT,NOR67Y);
NOTT(ILACO2,NOR48Y);
NOTT(IOUT4,XOR68Y);
NOTT(IOUT3,XOR69Y);
NOTT(ICOMPARE,INV73Y);
NOTT(IOUT2,XOR70Y);
NOTT(IOUT1,XOR71Y);
```

```
.OUT1       =    .IOUT1;
.OUT2       =    .IOUT2;
.OUT3       =    .IOUT3;
.OUT4       =    .IOUT4;
.COMPARE    =  .ICOMPARE;
.LACO1      =    .ILACO1;
.LACO2      =    .ILACO2;
.CAROUT     =    .ICAROUT
```

PROGRAM OUTPUT

4.3 OUTPUT (NON-FAULTED) -- ALU

OUT1	OUT2	OUT3	OUT4	CMP	L1	L2	C
1	0	0	0	0	0	0	1
1	1	1	1	1	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	1	1	1	1	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
1	0	1	1	0	1	1	0
1	1	0	1	0	1	1	0
0	0	1	1	0	1	1	0
0	0	1	1	0	1	1	0
1	0	1	1	0	1	1	0
1	1	0	1	0	1	1	0
0	0	1	1	0	1	1	0
0	0	1	1	0	1	1	0
1	1	0	0	0	1	1	0
1	0	0	0	0	1	1	0
0	1	0	0	0	1	1	0
0	1	0	0	0	1	1	0
1	1	0	0	0	1	1	0
1	0	0	0	0	1	1	0
0	1	0	0	0	1	1	0

0	1	0	0	0	1	1	0
1	1	1	1	1	1	1	0
1	0	1	1	0	1	1	0
0	1	1	1	0	1	1	0
0	1	1	1	0	1	1	0
1	1	1	1	1	1	1	0
1	0	1	1	0	1	1	0
0	1	1	1	0	1	1	0
0	1	1	1	0	1	1	0

NOTE: "0" = "00000000"
"1" = "FFFFFFFF"

USER INPUT

4.4 INPUT -- ALU

(ALU.DAT)

```
32              { number of desired cycles. }
0,0,0,0,0,1,1,1,1,1,0,0,1,0
9,9,9,9,9,9,9,9,9,9,9,9,0,9
1,0,0,0,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
0,1,0,0,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
1,1,0,0,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
0,0,1,0,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
1,0,1,0,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
0,1,1,0,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
1,1,1,0,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
0,0,0,1,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
1,0,0,1,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
0,1,0,1,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
1,1,0,1,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
0,0,1,1,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
1,0,1,1,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
0,1,1,1,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
1,1,1,1,9,9,9,9,9,9,9,9,1,9
9,9,9,9,9,9,9,9,9,9,9,9,0,9
```

Column	Variable
1	SEL1
2	SEL2
3	SEL3
4	SEL4
5	A1
6	A2
7	A3
8	A4
9	B1
10	B2
11	B3
12	B4
13	CONTROL
14	CARIN

4.5 EXAMPLES

USER INPUT
EXAMPLE 1

FAULTLIST ALU

(ALUFLT.DAT)

(output stuck at 0)

```
PINFLTS;
INV3:       Y*  0;
INV35:      Y*  0;
INV4:       Y*  0;
INV5:       Y*  0;
INV6:       Y*  0;
INV7:       Y*  0;
INV8:       Y*  0;
INV9:       Y*  0;
INV10:      Y*  0;
INV11:      Y*  0;
INV12:      Y*  0;
INV13:      Y*  0;
INV14:      Y*  0;
NOR44:      Y*  0;
OR45:       Y*  0;
OR46:       Y*  0;
NOR47:      Y*  0;
NOR48:      Y*  0;
XOR49:      Y*  0;
NOR50:      Y*  0;
NOR51:      Y*  0;
OR52:       Y*  0;
OR53:       Y*  0;
XOR54:      Y*  0;
NOR55:      Y*  0;
OR56:       Y*  0;
OR57:       Y*  0;
XOR58:      Y*  0;
OR59:       Y*  0;
OR60:       Y*  0;
INV85:      Y*  0;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 1

ALU

(DETECTED.DAT)

(output stuck at 0)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
INV3	INV3Y	1
INV35	NOR51Y	1
INV4	INV4Y	1
INV5	XOR49Y	9
INV7	XOR69Y	9
INV12	NAND63Y	1
NOR44	NAND63Y	1
OR45	NAND63Y	3
OR46	NAND63Y	3
NOR48	NOR48Y	1
XOR49	XOR49Y	1
NOR51	NOR51Y	2
OR52	OR52Y	1
OR53	OR53Y	1
XOR54	XOR69Y	1
OR56	NAND65Y	1
OR57	NAND65Y	1
XOR58	XOR70Y	1
OR59	NAND66Y	1
OR60	NAND66Y	1
INV85	NAND65Y	1

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 2

FAULTLIST ALU

(ALUFLT.DAT)

```
(output stuck at 1)

PINFLTS;
INV3:       Y*  1;
INV35:      Y*  1;
INV4:       Y*  1;
INV5:       Y*  1;
INV6:       Y*  1;
INV7:       Y*  1;
INV8:       Y*  1;
INV9:       Y*  1;
INV10:      Y*  1;
INV11:      Y*  1;
INV12:      Y*  1;
INV13:      Y*  1;
INV14:      Y*  1;
NOR44:      Y*  1;
OR45:       Y*  1;
OR46:       Y*  1;
NOR47:      Y*  1;
NOR48:      Y*  1;
XOR49:      Y*  1;
NOR50:      Y*  1;
NOR51:      Y*  1;
OR52:       Y*  1;
OR53:       Y*  1;
XOR54:      Y*  1;
NOR55:      Y*  1;
OR56:       Y*  1;
OR57:       Y*  1;
XOR58:      Y*  1;
OR59:       Y*  1;
OR60:       Y*  1;
INV85:      Y*  1;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 2

ALU

(DETECTED.DAT)

(output stuck at 1)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
INV3	INV3Y	5
INV35	NOR51Y	2
INV4	INV4Y	3
INV6	XOR49Y	1
INV8	XOR69Y	1
INV9	XOR70Y	9
INV10	INV10Y	1
INV11	NAND63Y	3
INV13	XOR49Y	9
INV14	XOR69Y	9
NOR44	NAND63Y	3
NOR47	NOR47Y	1
NOR48	NOR48Y	9
XOR49	XOR49Y	9
NOR50	NOR50Y	1
NOR51	NOR51Y	1
XOR54	XOR69Y	9
NOR55	NAND65Y	1
OR56	NAND65Y	2
XOR58	XOR70Y	17
OR60	NAND66Y	2

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 3

FAULTLIST ALU

(ALUFLT.DAT)

(input pin A stuck at 0)

```
PINFLTS;
INV3:       A*  0;
INV35:      A*  0;
INV4:       A*  0;
INV5:       A*  0;
INV6:       A*  0;
INV7:       A*  0;
INV8:       A*  0;
INV9:       A*  0;
INV10:      A*  0;
INV11:      A*  0;
INV12:      A*  0;
INV13:      A*  0;
INV14:      A*  0;
NOR44:      A*  0;
OR45:       A*  0;
OR46:       A*  0;
NOR47:      A*  0;
NOR48:      A*  0;
XOR49:      A*  0;
NOR50:      A*  0;
NOR51:      A*  0;
OR52:       A*  0;
OR53:       A*  0;
XOR54:      A*  0;
NOR55:      A*  0;
OR56:       A*  0;
OR57:       A*  0;
XOR58:      A*  0;
OR59:       A*  0;
OR60:       A*  0;
INV85:      A*  0;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 3

ALU

(DETECTED.DAT)

(input pin A stuck at 0)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
INV5	XOR49Y	9
INV7	XOR69Y	9
OR45	NAND63Y	3
OR46	NAND63Y	3
XOR49	XOR49Y	9
NOR51	NOR51Y	10
OR53	OR53Y	2
XOR54	XOR69Y	9
OR56	NAND65Y	18
OR57	NAND65Y	4
XOR58	XOR70Y	17
OR60	NAND66Y	1

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 4

FAULTLIST ALU

(ALUFLT.DAT)

(input pin A stuck at 1)

```
PINFLTS;
INV3:       A*  1;
INV35:      A*  1;
INV4:       A*  1;
INV5:       A*  1;
INV6:       A*  1;
INV7:       A*  1;
INV8:       A*  1;
INV9:       A*  1;
INV10:      A*  1;
INV11:      A*  1;
INV12:      A*  1;
INV13:      A*  1;
INV14:      A*  1;
NOR44:      A*  1;
OR45:       A*  1;
OR46:       A*  1;
NOR47:      A*  1;
NOR48:      A*  1;
XOR49:      A*  1;
NOR50:      A*  1;
NOR51:      A*  1;
OR52:       A*  1;
OR53:       A*  1;
XOR54:      A*  1;
NOR55:      A*  1;
OR56:       A*  1;
OR57:       A*  1;
XOR58:      A*  1;
OR59:       A*  1;
OR60:       A*  1;
INV85:      A*  1;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 4

ALU

(DETECTED.DAT)

(input pin A stuck at 1)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
INV3	INV3Y	1
INV35	NOR51Y	2
INV6	XOR49Y	1
INV8	XOR69Y	1
NOR44	NAND63Y	1
NOR47	NOR47Y	1
NOR48	NOR48Y	9
XOR49	XOR49Y	9
NOR51	NOR51Y	10
OR53	OR53Y	2
XOR54	XOR69Y	9
OR56	NAND65Y	18
OR57	NAND65Y	4
XOR58	XOR70Y	17
OR60	NAND66Y	1

ADDRESS LOCATION	MACHINE #	CYCLE #

5.0 BCD ADDER

USER INPUT

5.1 PARTSLIST -- BCD ADDER (FIG. 4)

(BCD.PRT)

```
USER:"DEUTSCH";
NAME:BCDADD;
PURPOSE:TESTING;
LEVEL:CHIP;
TYPES:T000M,T002M,T004M,T008M,T010M,T011M,T021M,T025M,T027M,T086M,P000M,
     P002M;
EXT:: A1,A2,A3,A4,B1,B2,B3,B4,CARIN,CAROUT,SUM1,SUM2,SUM3,SUM4;
INPUTS: .A1,.A2,.A3,.A4,.B1,.B2,.B3,.B4,.CARIN;
OUTPUTS: .CAROUT,.SUM1,.SUM2,.SUM3,.SUM4;
T000M: NAND1,NAND2,NAND3,NAND4,NAND30;
T002M: NOR15,NOR16,NOR17,NOR18,NOR31,NOR41,NOR45,NOR46,NOR62,NOR63;
T004M: INV7,INV10,INV19,INV20,INV21,INV42,INV47,INV48,INV64;
T008M: AND5,AND6,AND8,AND9,AND11,AND12,AND13,AND14,AND22,AND25,AND29,AND34,
     AND35,AND40,AND44,AND50,AND51,AND52,AND53,AND54,AND55,AND57,AND59;
T010M: NAND38;
T011M: AND24,AND28,AND39,AND43,AND49,AND56,AND58;
T021M: AND27;
T025M: NOR60;
T027M: NOR32,NOR33,NOR61;
T086M: XOR36,XOR37;
P000M: PA1,PA2,PA3,PA4,PB1,PB2,PB3,PB4,PCARIN,AND23,AND26;
P002M: PSUM1,PSUM2,PSUM3,PSUM4,PCAROUT;
END;
COMPSEGMENT;
= A1*IA1,A2*IA2,A3*IA3,A4*IA4,B1*IB1,B2*IB2,B3*IB3,B4*IB4,CARIN*ICARIN,
     CAROUT*PCAROUTY,SUM1*PSUM1Y,SUM2*PSUM2Y,SUM3*PSUM3Y,SUM4*PSUM4Y;
NAND1= A*PA1Y,B*PB1Y,Y*NAND1Y;
NAND2= A*PA2Y,B*PB2Y,Y*NAND2Y;
NAND3= A*PA3Y,B*PB3Y,Y*NAND3Y;
NAND4= A*PA4Y,B*PB4Y,Y*NAND4Y;
NAND30= A*PCARINY,B*NOR15Y,Y*NAND30Y;
NOR15= A*AND5Y,B*AND6Y,Y*NOR15Y;
NOR16= A*AND8Y,B*AND9Y,Y*NOR16Y;
NOR17= A*AND11Y,B*AND12Y,Y*NOR17Y;
NOR18= A*AND13Y,B*AND14Y,Y*NOR18Y;
NOR31= A*AND22Y,B*AND23Y,Y*NOR31Y;
NOR41= A*AND34Y,B*AND35Y,Y*NOR41Y;
NOR45= A*AND39Y,B*AND40Y,Y*NOR45Y;
NOR46= A*AND43Y,B*AND44Y,Y*NOR46Y;
NOR62= A*AND56Y,B*AND57Y,Y*NOR62Y;
NOR63= A*AND58Y,B*AND59Y,Y*NOR63Y;
INV7= A*NAND1Y,Y*INV7Y;
INV10= A*NAND2Y,Y*INV10Y;
INV19= A*NOR15Y,Y*INV19Y;
INV20= A*NOR16Y,Y*INV20Y;
```

```
INV21= A*NOR17Y,Y*INV21Y;
AND23= A*INV7Y,Y*AND23Y;
AND26= A*INV10Y,Y*AND26Y;
INV42= A*XOR36Y,Y*INV42Y;
INV47= A*NOR46Y,Y*INV47Y;
INV48= A*NOR45Y,Y*INV48Y;
INV64= A*NOR62Y,Y*INV64Y;
AND5= A*PA1Y,B*NAND1Y,Y*AND5Y;
AND6= A*NAND1Y,B*PB1Y,Y*AND6Y;
AND8= A*PA2Y,B*NAND2Y,Y*AND8Y;
AND9= A*NAND2Y,B*PB2Y,Y*AND9Y;
AND11= A*PA3Y,B*NAND3Y,Y*AND11Y;
AND12= A*NAND3Y,B*PB3Y,Y*AND12Y;
AND13= A*PA4Y,B*NAND4Y,Y*AND13Y;
AND14= A*NAND4Y,B*PB4Y,Y*AND14Y;
AND22= A*PCARINY,B*INV19Y,Y*AND22Y;
AND25= A*INV7Y,B*INV20Y,Y*AND25Y;
AND29= A*INV10Y,B*INV21Y,Y*AND29Y;
AND34= A*PCARINY,B*NAND30Y,Y*AND34Y;
AND35= A*NAND30Y,B*NOR15Y,Y*AND35Y;
AND40= A*NOR18Y,B*NAND4Y,Y*AND40Y;
AND44= A*NAND38Y,B*NOR18Y,Y*AND44Y;
AND50= A*INV42Y,B*INV47Y,Y*AND50Y;
AND51= A*XOR37Y,B*INV47Y,Y*AND51Y;
AND52= A*INV42Y,B*NOR45Y,Y*AND52Y;
AND53= A*XOR37Y,B*INV48Y,Y*AND53Y;
AND54= A*XOR36Y,B*INV47Y,Y*AND54Y;
AND55= A*INV42Y,B*NOR45Y,Y*AND55Y;
AND57= A*INV42Y,B*NOR45Y,Y*AND57Y;
AND59= A*INV48Y,B*NOR46Y,Y*AND59Y;
NAND38= A*NOR33Y,B*NAND3Y,C*NOR18Y,Y*NAND38Y;
AND24= A*PCARINY,B*INV19Y,C*INV20Y,Y*AND24Y;
AND28= A*INV7Y,B*INV20Y,C*INV21Y,Y*AND28Y;
AND39= A*NOR33Y,B*NAND3Y,C*NAND4Y,Y*AND39Y;
AND43= A*NAND3Y,B*NOR33Y,C*NAND38Y,Y*AND43Y;
AND49= A*XOR36Y,B*NOR46Y,C*INV48Y,Y*AND49Y;
AND56= A*XOR36Y,B*XOR37Y,C*INV47Y,Y*AND56Y;
AND58= A*XOR36Y,B*XOR37Y,C*INV48Y,Y*AND58Y;
AND27= A*PCARINY,B*INV19Y,C*INV20Y,D*INV21Y,Y*AND27Y;
NOR60= A*AND49Y,B*AND50Y,C*AND51Y,D*AND52Y,Y*NOR60Y;
NOR32= A*AND24Y,B*AND25Y,C*AND26Y,Y*NOR32Y;
NOR33= A*AND27Y,B*AND28Y,C*AND29Y,Y*NOR33Y;
NOR61= A*AND53Y,B*AND54Y,C*AND55Y,Y*NOR61Y;
PA1= A*IA1,Y*PA1Y;
PA2= A*IA2,Y*PA2Y;
PA3= A*IA3,Y*PA3Y;
PA4= A*IA4,Y*PA4Y;
PB1= A*IB1,Y*PB1Y;
PB2= A*IB2,Y*PB2Y;
PB3= A*IB3,Y*PB3Y;
PB4= A*IB4,Y*PB4Y;
PCARIN= A*ICARIN,Y*PCARINY;
PSUM1= A*NOR41Y,Y*PSUM1Y;
PSUM2= A*NOR60Y,Y*PSUM2Y;
```

```
PSUM3= A*NOR61Y,Y*PSUM3Y;
PSUM4= A*INV64Y,Y*PSUM4Y;
PCAROUT= A*NOR63Y,Y*PCAROUTY;
XOR36= A*NOR31Y,B*INV20Y,Y*XOR36Y;
XOR37= A*NOR32Y,B*INV21Y,Y*XOR37Y;
ENDCOMPS;
DETECTSEGMENT;
   PINS;
        AND5Y;
        AND6Y;
        AND8Y;
        AND9Y;
        NOR31Y;
        NOR41Y;
        NOR45Y;
        NOR62Y;
        INV42Y;
        INV64Y;
        AND50Y;
        AND51Y;
        AND52Y;
        AND53Y;
        AND55Y;
        AND57Y;
ENDDET;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT

5.2 P-ORDERING -- BCD ADDER

(DWN.B32)

```
 IA1                                =                                ..A1;

IA2                                 =                                ..A2;

IA3                                 =                                ..A3;

IA4                                 =                                ..A4;

IB1                                 =                                ..B1;

IB2                                 =                                ..B2;

IB3                                 =                                ..B3;

IB4                                 =                                ..B4;

ICARIN                              =                              ..CARIN;

 BUF(PA1Y,IA1);
 BUF(PA2Y,IA2);
 BUF(PA3Y,IA3);
 BUF(PA4Y,IA4);
 BUF(PB1Y,IB1);
 BUF(PB2Y,IB2);
 BUF(PB3Y,IB3);
 BUF(PB4Y,IB4);
 BUF(PCARINY,ICARIN);
 NAND2F(NAND1Y,PA1Y,PB1Y,ZERO,ZERO,ZAND[0000000001],ZERO);
 NAND2F(NAND2Y,PA2Y,PB2Y,ZERO,ZERO,ZAND[0000000002],ZERO);
 NAND2F(NAND3Y,PA3Y,PB3Y,ZERO,ZERO,ZAND[0000000003],ZERO);
 NAND2F(NAND4Y,PA4Y,PB4Y,ZERO,ZERO,ZAND[0000000004],ZERO);
 NOTTF(INV7Y,NAND1Y,ZAND[0000000007],ZERO);
 NOTTF(INV10Y,NAND2Y,ZAND[0000000009],ZERO);
 FBUFF(AND23Y,INV7Y,ZAND[0000000013],ZERO);
 FBUFF(AND26Y,INV10Y,ZAND[0000000016],ZERO);
 AND2F(AND5Y,PA1Y,NAND1Y,ZERO,ZERO,ZAND[0000000005],ZERO);
 AND2F(AND6Y,NAND1Y,PB1Y,ZERO,ZERO,ZAND[0000000006],ZERO);
 AND2(AND8Y,PA2Y,NAND2Y);
 AND2F(AND9Y,NAND2Y,PB2Y,ZERO,ZERO,ZAND[0000000008],ZERO);
 AND2F(AND11Y,PA3Y,NAND3Y,ZERO,ZERO,ZAND[0000000010],ZERO);
 AND2(AND12Y,NAND3Y,PB3Y);
 AND2F(AND13Y,PA4Y,NAND4Y,ZERO,ZERO,ZAND[0000000011],ZERO);
 AND2F(AND14Y,NAND4Y,PB4Y,ZERO,ZERO,ZAND[0000000012],ZERO);
 NOR2(NOR15Y,AND5Y,AND6Y);
 NOR2(NOR16Y,AND8Y,AND9Y);
 NOR2(NOR17Y,AND11Y,AND12Y);
 NOR2(NOR18Y,AND13Y,AND14Y);
 NOTT(INV19Y,NOR15Y);
```

```
NOTT(INV20Y,NOR16Y);
NOTT(INV21Y,NOR17Y);
AND2(AND22Y,PCARINY,INV19Y);
AND2F(AND25Y,INV7Y,INV20Y,ZERO,ZERO,ZAND[0000000015],ZERO);
AND2F(AND29Y,INV10Y,INV21Y,ZERO,ZERO,ZAND[0000000017],ZERO);
AND2F(AND40Y,NOR18Y,NAND4Y,ZERO,ZERO,ZAND[0000000023],ZERO);
AND3F(AND24Y,PCARINY,INV19Y,INV20Y,ZERO,ZERO,ZERO,ZAND[0000000014],
ZERO);
AND3(AND28Y,INV7Y,INV20Y,INV21Y);
AND4(AND27Y,PCARINY,INV19Y,INV20Y,INV21Y);
NOR3(NOR32Y,AND24Y,AND25Y,AND26Y);
NOR3F(NOR33Y,AND27Y,AND28Y,AND29Y,ZERO,ZERO,ZERO,ZAND[0000000019],
ZERO);
XOR2(XOR37Y,NOR32Y,INV21Y);
NAND2F(NAND30Y,PCARINY,NOR15Y,ZERO,ZERO,ZAND[0000000018],ZERO);
NOR2(NOR31Y,AND22Y,AND23Y);
AND2F(AND34Y,PCARINY,NAND30Y,ZERO,ZERO,ZAND[0000000020],ZERO);
AND2F(AND35Y,NAND30Y,NOR15Y,ZERO,ZERO,ZAND[0000000021],ZERO);
NAND3(NAND38Y,NOR33Y,NAND3Y,NOR18Y);
AND3F(AND39Y,NOR33Y,NAND3Y,NAND4Y,ZERO,ZERO,ZERO,ZAND[0000000022],
ZERO);
AND3(AND43Y,NAND3Y,NOR33Y,NAND38Y);
XOR2(XOR36Y,NOR31Y,INV20Y);
NOR2(NOR41Y,AND34Y,AND35Y);
NOR2F(NOR45Y,AND39Y,AND40Y,ZERO,ZERO,ZAND[0000000024],ZERO);
NOTT(INV42Y,XOR36Y);
NOTT(INV48Y,NOR45Y);
AND2(AND44Y,NAND38Y,NOR18Y);
AND2F(AND52Y,INV42Y,NOR45Y,ZERO,ZERO,ZAND[0000000026],ZERO);
AND2F(AND53Y,XOR37Y,INV48Y,ZERO,ZERO,ZAND[0000000027],ZERO);
AND2(AND55Y,INV42Y,NOR45Y);
AND2(AND57Y,INV42Y,NOR45Y);
AND3F(AND58Y,XOR36Y,XOR37Y,INV48Y,ZERO,ZERO,ZERO,ZAND[0000000029],
ZERO);
NINV(PSUM1Y,NOR41Y);
NOR2(NOR46Y,AND43Y,AND44Y);
NOTT(INV47Y,NOR46Y);
AND2(AND50Y,INV42Y,INV47Y);
AND2F(AND51Y,XOR37Y,INV47Y,ZERO,ZERO,ZAND[0000000025],ZERO);
AND2(AND54Y,XOR36Y,INV47Y);
AND2(AND59Y,INV48Y,NOR46Y);
AND3(AND49Y,XOR36Y,NOR46Y,INV48Y);
AND3F(AND56Y,XOR36Y,XOR37Y,INV47Y,ZERO,ZERO,ZERO,ZAND[0000000028],
ZERO);

NOR4F(NOR60Y,AND49Y,AND50Y,AND51Y,AND52Y,ZERO,ZERO,ZERO,ZERO,ZAND[0000000030],
ZERO);
NOR3F(NOR61Y,AND53Y,AND54Y,AND55Y,ZERO,ZERO,ZERO,ZAND[0000000031],
ZERO);
NINV(PSUM2Y,NOR60Y);
NINV(PSUM3Y,NOR61Y);
NOR2(NOR62Y,AND56Y,AND57Y);
NOR2(NOR63Y,AND58Y,AND59Y);
NOTT(INV64Y,NOR62Y);
```

```
 NINV(PSUM4Y,INV64Y);
 NINV(PCAROUTY,NOR63Y);
.CAROUT = .PCAROUTY;
.SUM1 = .PSUM1Y;
.SUM2 = .PSUM2Y;
.SUM3 = .PSUM3Y;
.SUM4 = .PSUM4Y
```

PROGRAM OUTPUT

5.3 OUTPUT (NON-FAULTED) -- BCD ADDER

CAROUT	SUM1	SUM2	SUM3	SUM4
0	0	0	0	0
0	1	0	0	0
0	1	1	1	0
0	0	0	0	1
1	1	0	0	0
1	0	1	0	0
1	1	1	0	0
1	0	0	1	0
1	1	0	0	1

NOTE: "0" = "00000000"
"1" = "FFFFFFFF"

PROGRAM INPUT

5.4 INPUT-- BCD ADDER

(BCD.DAT)

```
9                       { number of desired cycles. }
0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,1
0,0,1,0,1,1,0,0,0
9,9,9,9,9,9,9,9,1
1,1,1,0,0,0,1,0,0
9,9,9,9,9,9,9,9,1
0,0,0,1,1,0,1,0,0
0,1,1,0,0,0,0,1,0
1,0,0,1,1,0,0,1,1
```

Column	Variables
1	A1
2	A2
3	A3
4	A4
5	B1
6	B2
7	B3
8	B4
9	CARIN

5.5 EXAMPLES

USER INPUT
EXAMPLE 1

FAULTLIST BCD ADDER

(BCDFLT.DAT)

```
(output stuck at 0)

 PINFLTS:
 NAND1:      Y*  0;
 NAND2:      Y*  0;
 NAND3:      Y*  0;
 NAND4:      Y*  0;
 AND5:       Y*  0;
 AND6:       Y*  0;
 INV7:       Y*  0;
 AND9:       Y*  0;
 INV10:      Y*  0;
 AND11:      Y*  0;
 AND13:      Y*  0;
 AND14:      Y*  0;
 AND23:      Y*  0;
 AND24:      Y*  0;
 AND25:      Y*  0;
 AND26:      Y*  0;
 AND29:      Y*  0;
 NAND30:     Y*  0;
 NOR33:      Y*  0;
 AND34:      Y*  0;
 AND35:      Y*  0;
 AND39:      Y*  0;
 AND40:      Y*  0;
 NOR45:      Y*  0;
 AND51:      Y*  0;
 AND52:      Y*  0;
 AND53:      Y*  0;
 AND56:      Y*  0;
 AND58:      Y*  0;
 NOR60:      Y*  0;
 NOR61:      Y*  0;
 ENDPINS;
 ENDC;
 END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 1

BCD

(DETECTED.DAT)

(output stuck at 0)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
NAND1	NOR31Y	1
NAND2	AND53Y	1
NAND3	NOR62Y	1
NAND4	NOR45Y	1
AND5	AND5Y	5
AND6	AND6Y	3
INV7	NOR31Y	9
AND9	AND9Y	3
AND11	AND53Y	3
AND14	AND50Y	8
AND23	NOR31Y	9
AND24	NOR62Y	4
NAND30	NOR41Y	1
NOR33	NOR62Y	1
AND34	NOR41Y	4
AND35	NOR41Y	1
AND39	NOR45Y	7
AND40	NOR45Y	4
NOR45	NOR45Y	9
AND51	AND51Y	4
AND52	AND52Y	9
AND53	AND53Y	1
AND56	NOR62Y	4

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 2

FAULTLIST BCD ADDER

(BCDFLT.DAT)

```
(output stuck at 1)

PINFLTS:
NAND1:      Y*  1;
NAND2:      Y*  1;
NAND3:      Y*  1;
NAND4:      Y*  1;
AND5:       Y*  1;
AND6:       Y*  1;
INV7:       Y*  1;
AND9:       Y*  1;
INV10:      Y*  1;
AND11:      Y*  1;
AND13:      Y*  1;
AND14:      Y*  1;
AND23:      Y*  1;
AND24:      Y*  1;
AND25:      Y*  1;
AND26:      Y*  1;
AND29:      Y*  1;
NAND30:     Y*  1;
NOR33:      Y*  1;
AND34:      Y*  1;
AND35:      Y*  1;
AND39:      Y*  1;
AND40:      Y*  1;
NOR45:      Y*  1;
AND51:      Y*  1;
AND52:      Y*  1;
AND53:      Y*  1;
AND56:      Y*  1;
AND58:      Y*  1;
NOR60:      Y*  1;
NOR61:      Y*  1;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 2

BCD

(DETECTED.DAT)

(output stuck at 1)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
NAND1	AND5Y	9
NAND3	AND50Y	5
NAND4	NOR45Y	9
AND5	AND5Y	1
AND6	AND6Y	1
INV7	NOR31Y	1
AND9	AND9Y	1
INV10	AND53Y	1
AND11	AND53Y	1
AND13	NOR62Y	1
AND14	NOR62Y	1
AND23	NOR31Y	1
AND24	AND53Y	1
AND25	AND53Y	1
AND26	AND53Y	1
AND29	NOR62Y	1
NAND30	NOR41Y	2
NOR33	NOR62Y	4
AND34	NOR41Y	2
AND35	NOR41Y	2
AND39	NOR45Y	9
AND40	NOR45Y	9
NOR45	NOR45Y	1
AND51	AND51Y	1
AND52	AND52Y	1
AND53	AND53Y	3
AND56	NOR62Y	1

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 3

FAULTLIST BCD ADDER

(BCDFLT.DAT)

```
(input stuck at 0)

PINFLTS:
NAND1:      A*  0;
NAND2:      A*  0;
NAND3:      A*  0;
NAND4:      A*  0;
AND5:       A*  0;
AND6:       A*  0;
INV7:       A*  0;
AND9:       A*  0;
INV10:      A*  0;
AND11:      A*  0;
AND13:      A*  0;
AND14:      A*  0;
AND23:      A*  0;
AND24:      A*  0;
AND25:      A*  0;
AND26:      A*  0;
AND29:      A*  0;
NAND30:     A*  0;
NOR33:      A*  0;
AND34:      A*  0;
AND35:      A*  0;
AND39:      A*  0;
AND40:      A*  0;
NOR45:      A*  0;
AND51:      A*  0;
AND52:      A*  0;
AND53:      A*  0;
AND56:      A*  0;
AND58:      A*  0;
NOR60:      A*  0;
NOR61:      A*  0;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 3

BCD

(DETECTED.DAT)

(input stuck at 0)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
NAND1	AND6Y	3
NAND2	AND9Y	3
NAND3	NOR45Y	7
NAND4	NOR45Y	8
AND5	AND5Y	5
AND6	AND6Y	9
AND11	AND53Y	1
AND13	NOR62Y	1
AND14	AND50Y	9
AND23	NOR31Y	9
AND24	AND53Y	3
AND25	AND53Y	3
AND29	AND50Y	3
NAND30	NOR41Y	1
NOR33	NOR62Y	4
AND34	NOR41Y	3
AND35	NOR41Y	2
NOR45	NOR45Y	7
AND51	AND51Y	6
AND53	AND53Y	3
AND56	NOR62Y	5

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 4

FAULTLIST BCD ADDER

(BCDFLT.DAT)

```
(input stuck at 1)

PINFLTS:
NAND1:      A*  1;
NAND2:      A*  1;
NAND3:      A*  1;
NAND4:      A*  1;
AND5:       A*  1;
AND6:       A*  1;
INV7:       A*  1;
AND9:       A*  1;
INV10:      A*  1;
AND11:      A*  1;
AND13:      A*  1;
AND14:      A*  1;
AND23:      A*  1;
AND24:      A*  1;
AND25:      A*  1;
AND26:      A*  1;
AND29:      A*  1;
NAND30:     A*  1;
NOR33:      A*  1;
AND34:      A*  1;
AND35:      A*  1;
AND39:      A*  1;
AND40:      A*  1;
NOR45:      A*  1;
AND51:      A*  1;
AND52:      A*  1;
AND53:      A*  1;
AND56:      A*  1;
AND58:      A*  1;
NOR60:      A*  1;
NOR61:      A*  1;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 4

BCD

(DETECTED.DAT)

(input stuck at 1)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
NAND1	AND5Y	9
NAND2	AND9Y	3
NAND3	NOR62Y	1
NAND4	NOR45Y	8
AND5	AND5Y	1
AND6	AND6Y	1
INV7	NOR31Y	1
AND9	AND9Y	1
AND11	AND53Y	1
AND13	NOR62Y	1
AND14	AND50Y	9
AND24	AND53Y	3
AND25	AND53Y	3
AND26	AND53Y	1
AND29	AND50Y	3
NAND30	NOR41Y	1
NOR33	NOR62Y	4
AND34	NOR41Y	3
AND35	NOR41Y	2
NOR45	NOR45Y	1
AND51	AND51Y	6
AND53	AND53Y	3
AND56	NOR62Y	5

ADDRESS LOCATION	MACHINE #	CYCLE #

6.0 DOWNCOUNTER

USER INPUT

6.1 PARTSLIST - DOWNCOUNTER (FIG. 3)

(DWN.PRT)

```
USER:       "SWD";
NAME:       DWNCNTG;
PURPOSE:    TEGATE;
LEVEL:      CHIP;
TYPES:      C050F,      C050L,      P000M,      P002M,      T002M,
            T004M,      T008M,      T027M,      T032M,      T074M,
            CLK;
EXT::       CARRYIN,      CLEAR,        CLOCK,        P0,
            P1,           P2,           P3,           PRELOAD,
            SSET,         CARRYOUT,     Q0,           Q1,
            Q2,           Q3;
INPUTS:     .CARRYIN,     .CLEAR,       .CLOCK,       .P0,
            .P1,          .P2,          .P3,          .PRELOAD,
            .SSET;
OUTPUTS:    .CARRYOUT,    .Q0,          .Q1,          .Q2,
            .Q3;
C050F:      U48,        U46;
C050L:      U47;
P000M:      PI2,        PI4,        PI3,        PI1,        PI8,
            PI9,        PI6,        PI7,        PI5;
P002M:      PO1,        PO3,        PO2,        PO5,        PO4;
T002M:      U19,        U44,        U20,        U3,         U2,
            U1,         U21,        U22;
T004M:      U8,         U26,        U7,         U25,        U24,
            U6,         U23,        U9;
T008M:      U29,        U11,        U10,        U15,        U33,
            U34,        U16,        U17,        U28,        U18,
            U27,        U5,         U14,        U32,        U31,
            U30,        U13,        U45;
T027M:      U4,         U43;
T032M:      U35,        U36,        U37,        U38;
T074M:      U42,        U41,        U40,        U39;
CLK:        C1,         C2,         C3,         C4;
END;
COMPSEGMENT;
PO5       = A*U45Y,               Y*XCARRYOUT;
PO4       = A*U42Q,               Y*XQ3;
PO3       = A*U41Q,               Y*XQ2;
PO2       = A*U40Q,               Y*XQ1;
PO1       = A*U39Q,               Y*XQ0;
U48       = A*PI5Y,               Y*LD;
U47       = A*U46Y,               Y*CK;
U46       = A*PI3Y,               Y*U46Y;
U45       = A*U43Y,               B*U44Y,               Y*U45Y;
U44       = A*U39Q,               B*CI,                 Y*U44Y;
```

```
U43     = A*U42Q,       B*U41Q,         C*U40Q,
          Y*U43Y;
U42     = D*U38Y,       CK*CK,          CLR*CLR,
          Q*U42Q,       QB*U42QB,       PR*STT;
U41     = D*U37Y,       CK*CK,          CLR*CLR,
          Q*U41Q,       QB*U41QB,       PR*STT;
U40     = D*U36Y,       CK*CK,          CLR*CLR,
          Q*U40Q,       QB*U40QB,       PR*STT;
U39     = D*U35Y,       CK*CK,          CLR*CLR,
          Q*U39Q,       QB*U39QB,       PR*STT;
U38     = A*U33Y,       B*U34Y,         Y*U38Y;
U37     = A*U31Y,       B*U32Y,         Y*U37Y;
U36     = A*U29Y,       B*U30Y,         Y*U36Y;
U35     = A*U27Y,       B*U28Y,         Y*U35Y;
U34     = A*U26Y,       B*LD3,          Y*U34Y;
U33     = A*U22Y,       B*LD,           Y*U33Y;
U32     = A*U25Y,       B*LD2,          Y*U32Y;
U31     = A*U21Y,       B*LD,           Y*U31Y;
U30     = A*U24Y,       B*LD1,          Y*U30Y;
U29     = A*U20Y,       B*LD,           Y*U29Y;
U28     = A*U23Y,       B*LD0,          Y*U28Y;
U27     = A*U19Y,       B*LD,           Y*U27Y;
U26     = A*LD,         Y*U26Y;
U25     = A*LD,         Y*U25Y;
U24     = A*LD,         Y*U24Y;
U23     = A*LD,         Y*U23Y;
U22     = A*U17Y,       B*U18Y,         Y*U22Y;
U21     = A*U15Y,       B*U16Y,         Y*U21Y;
U20     = A*U13Y,       B*U14Y,         Y*U20Y;
U19     = A*U10Y,       B*U11Y,         Y*U19Y;
U18     = A*U9Y,        B*U42QB,        Y*U18Y;
U17     = A*U42Q,       B*U5Y,          Y*U17Y;
U16     = A*U8Y,        B*U41QB,        Y*U16Y;
U15     = A*U41Q,       B*U4Y,          Y*U15Y;
U14     = A*U7Y,        B*U40QB,        Y*U14Y;
U13     = A*U40Q,       B*U3Y,          Y*U13Y;
U11     = A*CI,         B*U39QB,        Y*U11Y;
U10     = A*U39Q,       B*U6Y,          Y*U10Y;
U9      = A*U5Y,        Y*U9Y;
U8      = A*U4Y,        Y*U8Y;
U7      = A*U3Y,        Y*U7Y;
U6      = A*CI,         Y*U6Y;
U5      = A*U1Y,        B*U2Y,          Y*U5Y;
U4      = A*CI,         B*U39Q,         C*U40Q,
          Y*U4Y;
U3      = A*U39Q,       B*CI,           Y*U3Y;
U2      = A*U40Q,       B*U41Q,         Y*U2Y;
```

```
U1        = A*CI,                B*U39Q,              Y*U1Y;
          =          CARRYIN*XCARRYIN,              CLEAR*XCLEAR,
                     CLOCK*XCLOCK,                  P0*XP0,
                     P1*XP1,                        P2*XP2,
                     P3*XP3,                        PRELOAD*XPRELOAD,
                     SSET*XSSET,                    CARRYOUT*XCARRYOUT,
                     Q0*XQ0,                        Q1*XQ1,
                     Q2*XQ2,                        Q3*XQ3;
PI9       = A*XP3,               Y*LD3;
PI8       = A*XP2,               Y*LD2;
PI7       = A*XP1,               Y*LD1;
PI6       = A*XP0,               Y*LD0;
PI5       = A*XPRELOAD,          Y*PI5Y;
PI4       = A*XCARRYIN,          Y*CI;
PI3       = A*XCLOCK,            Y*PI3Y;
PI2       = A*XSSET,             Y*STT;
PI1       = A*XCLEAR,            Y*CLR;
C1        = A*U35Y,              Y*T1;
C2        = A*U36Y,              Y*T2;
C3        = A*U37Y,              Y*T3;
C4        = A*U38Y,              Y*T4;
ENDCOMPS;
DETECTSEGMENT;
   PINS;
        U1Y;
        U2Y;
        U3Y;
        U15Y;
        U16Y;
        U17Y;
        U20Y;
        U30Y;
        U31Y;
        U32Y;
        U33Y;
        U34Y;
ENDDET;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT

6.2 P-ORDERING -- DOWNCOUNTER

(DWN.B32)

```
XCARRYIN                    =                    ..CARRYIN;

XCLEAR                      =                      ..CLEAR;

XCLOCK                      =                      ..CLOCK;

XP0                         =                         ..P0;

XP1                         =                         ..P1;

XP2                         =                         ..P2;

XP3                         =                         ..P3;

XPRELOAD                    =                    ..PRELOAD;

XSSET                       =                       ..SSET;

 BUF(LD3,XP3);
 BUF(LD2,XP2);
 BUF(LD1,XP1);
 BUF(LD0,XP0);
 BUF(PI5Y,XPRELOAD);
 BUF(CI,XCARRYIN);
 BUF(PI3Y,XCLOCK);
 BUF(STT,XSSET);
 BUF(CLR,XCLEAR);
 FNIB(LD,PI5Y);
 FNIB(U46Y,PI3Y);
 NOTTF(U26Y,LD,ZAND[0000000014],ZERO);
 NOTTF(U25Y,LD,ZAND[0000000030],ZERO);
 NOTT(U24Y,LD);
 NOTTF(U23Y,LD,ZAND[0000000015],ZERO);
 NOTTF(U6Y,CI,ZAND[0000000005],ZERO);
 LNIB(CK,U46Y);
 DFF(U42Q,U42QB,CK,T4,STT,CLR,.K+1);
 DFF(U41Q,U41QB,CK,T3,STT,CLR,.K+3);
 DFF(U40Q,U40QB,CK,T2,STT,CLR,.K+5);
 DFF(U39Q,U39QB,CK,T1,STT,CLR,.K+7);
 AND2F(U34Y,U26Y,LD3,ZERO,ZERO,ZAND[0000000007],ZERO);
 AND2(U32Y,U25Y,LD2);
 AND2F(U30Y,U24Y,LD1,ZERO,ZERO,ZAND[0000000009],ZERO);
 AND2F(U28Y,U23Y,LD0,ZERO,ZERO,ZAND[0000000031],ZERO);
 AND2F(U11Y,CI,U39QB,ZERO,ZERO,ZAND[0000000016],ZERO);
 AND2F(U10Y,U39Q,U6Y,ZERO,ZERO,ZAND[0000000026],ZERO);
 NOR3F(U4Y,CI,U39Q,U40Q,ZERO,ZERO,ZERO,ZAND[0000000010],ZERO);
 NOR2(U3Y,U39Q,CI);
```

```
 NOR2(U2Y,U40Q,U41Q);
 NOR2F(U1Y,CI,U39Q,ZERO,ZERO,ZAND[0000000003],ZERO);
 NINV(XQ3,U42Q);
 NINV(XQ2,U41Q);
 NINV(XQ1,U40Q);
 NINV(XQ0,U39Q);
 NOR2F(U44Y,U39Q,CI,ZERO,ZERO,ZAND[0000000012],ZERO);
 NOR3F(U43Y,U42Q,U41Q,U40Q,ZERO,ZERO,ZERO,ZAND[0000000020],ZERO);
 NOR2F(U19Y,U10Y,U11Y,ZERO,ZERO,ZAND[0000000001],ZERO);
 AND2F(U15Y,U41Q,U4Y,ZERO,ZERO,ZAND[0000000002],ZERO);
 AND2F(U13Y,U40Q,U3Y,ZERO,ZERO,ZAND[0000000019],ZERO);
 NOTTF(U8Y,U4Y,ZAND[0000000004],ZERO);
 NOTTF(U7Y,U3Y,ZAND[0000000024],ZERO);
 AND2F(U5Y,U1Y,U2Y,ZERO,ZERO,ZAND[0000000018],ZERO);
 AND2F(U45Y,U43Y,U44Y,ZERO,ZERO,ZAND[0000000029],ZERO);
 AND2F(U27Y,U19Y,LD,ZERO,ZERO,ZAND[0000000008],ZERO);
 AND2F(U17Y,U42Q,U5Y,ZERO,ZERO,ZAND[0000000027],ZERO);
 AND2F(U16Y,U8Y,U41QB,ZERO,ZERO,ZAND[0000000017],ZERO);
 AND2F(U14Y,U7Y,U40QB,ZERO,ZERO,ZAND[0000000028],ZERO);
 NOTTF(U9Y,U5Y,ZAND[0000000025],ZERO);
 NINV(XCARRYOUT,U45Y);
 OR2F(U35Y,U27Y,U28Y,ZERO,ZERO,ZAND[0000000011],ZERO);
 NOR2F(U21Y,U15Y,U16Y,ZERO,ZERO,ZAND[0000000013],ZERO);
 NOR2F(U20Y,U13Y,U14Y,ZERO,ZERO,ZAND[0000000022],ZERO);
 AND2(U18Y,U9Y,U42QB);
 AND2(U31Y,U21Y,LD);
 AND2F(U29Y,U20Y,LD,ZERO,ZERO,ZAND[0000000006],ZERO);
 NOR2F(U22Y,U17Y,U18Y,ZERO,ZERO,ZAND[0000000023],ZERO);
 OR2(U37Y,U31Y,U32Y);
 OR2F(U36Y,U29Y,U30Y,ZERO,ZERO,ZAND[0000000021],ZERO);
 AND2(U33Y,U22Y,LD);
 OR2(U38Y,U33Y,U34Y);
CLK(T1,U35Y,.K+9);

CLK(T2,U36Y,.K+10);

CLK(T3,U37Y,.K+11);

CLK(T4,U38Y,.K+12);

.CARRYOUT                    =                    .XCARRYOUT;

.Q0                          =                          .XQ0;

.Q1                          =                          .XQ1;

.Q2                          =                          .XQ2;

.Q3                          =                          .XQ3
```

PROGRAM OUTPUT

6.3 OUTPUT (NON-FAULTED) -- DOWNCOUNTER

CARRYOUT	Q0	Q1	Q2	Q3
1	0	0	0	0
0	1	1	1	1
0	0	1	1	1
0	1	0	1	1
0	0	0	1	1
0	1	1	0	1
0	0	1	0	1
0	1	0	0	1
0	0	0	0	1
0	1	1	1	0
0	0	1	1	0
0	1	0	1	0
0	0	0	1	0
0	1	1	0	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0
0	1	1	1	1
0	0	1	1	1
0	1	0	1	1
0	0	0	1	1
0	1	1	0	1
0	0	1	0	1

0	1	0	0	1
0	0	0	0	1
0	1	1	1	0
0	0	1	1	0
0	1	0	1	0
0	0	0	1	0
0	1	1	0	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0
0	1	1	1	1
0	0	1	1	1
0	1	0	1	1
0	0	0	1	1
0	1	1	0	1
0	0	1	0	1
0	1	0	0	1

NOTE: "0" = "00000000"
"1" = "FFFFFFFF"

6.4 EXAMPLES

USER INPUT
EXAMPLE 1

FAULTLIST DOWNCOUNTER

(output stuck at 0)

```
PINFLTS;
U19:      Y*   0;
U15:      Y*   0;
U1:       Y*   0;
U8:       Y*   0;
U6:       Y*   0;
U29:      Y*   0;
U34:      Y*   0;
U27:      Y*   0;
U30:      Y*   0;
U4:       Y*   0;
U35:      Y*   0;
U44:      Y*   0;
U21:      Y*   0;
U26:      Y*   0;
U23:      Y*   0;
U11:      Y*   0;
U16:      Y*   0;
U5:       Y*   0;
U13:      Y*   0;
U43:      Y*   0;
U36:      Y*   0;
U20:      Y*   0;
U22:      Y*   0;
U7:       Y*   0;
U9:       Y*   0;
U10:      Y*   0;
U17:      Y*   0;
U14:      Y*   0;
U45:      Y*   0;
U25:      Y*   0;
U28:      Y*   0;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
FOR
EXAMPLE 1

DOWNCOUNTER

(DETECTED.DAT)

(output stuck at 0)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
U19	U1Y	2
U15	U15Y	5
U1	U1Y	1
U8	U16Y	6
U6	U1Y	3
U29	U20Y	2
U27	U1Y	2
U4	U16Y	1
U35	U1Y	2
U21	U31Y	1
U16	U16Y	6
U5	U33Y	1
U13	U20Y	3
U36	U20Y	2
U20	U20Y	1
U22	U33Y	1
U7	U20Y	4
U9	U33Y	10
U10	U1Y	3
U17	U17Y	9
U14	U20Y	4

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 2

FAULTLIST DOWNCOUNTER

(output stuck at 1)

```
PINFLTS;
U19:      Y*   1;
U15:      Y*   1;
U1:       Y*   1;
U8:       Y*   1;
U6:       Y*   1;
U29:      Y*   1;
U34:      Y*   1;
U27:      Y*   1;
U30:      Y*   1;
U4:       Y*   1;
U35:      Y*   1;
U44:      Y*   1;
U21:      Y*   1;
U26:      Y*   1;
U23:      Y*   1;
U11:      Y*   1;
U16:      Y*   1;
U5:       Y*   1;
U13:      Y*   1;
U43:      Y*   1;
U36:      Y*   1;
U20:      Y*   1;
U22:      Y*   1;
U7:       Y*   1;
U9:       Y*   1;
U10:      Y*   1;
U17:      Y*   1;
U14:      Y*   1;
U45:      Y*   1;
U25:      Y*   1;
U28:      Y*   1;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 2

DOWNCOUNTER

(output stuck at 1)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
U19	U1Y	3
U15	U15Y	1
U1	U1Y	2
U8	U16Y	1
U29	U20Y	4
U34	U34Y	1
U27	U1Y	3
U30	U30Y	1
U4	U15Y	2
U35	U1Y	3
U21	U31Y	5
U26	U34Y	1
U23	U1Y	3
U11	U1Y	2
U16	U16Y	1
U5	U17Y	2
U13	U20Y	1
U36	U20Y	4
U20	U20Y	3
U22	U33Y	9
U7	U20Y	1
U9	U33Y	1
U10	U1Y	2
U17	U17Y	1
U14	U20Y	1
U25	U32Y	1
U28	U1Y	3

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 3

FAULTLIST DOWNCOUNTER

(input stuck at 0)

```
PINFLTS;
U19:      A*   0;
U15:      A*   0;
U1:       A*   0;
U8:       A*   0;
U6:       A*   0;
U29:      A*   0;
U34:      A*   0;
U27:      A*   0;
U30:      A*   0;
U4:       A*   0;
U35:      A*   0;
U44:      A*   0;
U21:      A*   0;
U26:      A*   0;
U23:      A*   0;
U11:      A*   0;
U16:      A*   0;
U5:       A*   0;
U13:      A*   0;
U43:      A*   0;
U36:      A*   0;
U20:      A*   0;
U22:      A*   0;
U7:       A*   0;
U9:       A*   0;
U10:      A*   0;
U17:      A*   0;
U14:      A*   0;
U45:      A*   0;
U25:      A*   0;
U28:      A*   0;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 3

DOWNCOUNTER

(input stuck at 0)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
U19	U1Y	3
U15	U15Y	1
U29	U20Y	4
U34	U34Y	1
U27	U1Y	2
U30	U30Y	1
U35	U1Y	2
U21	U31Y	5
U11	U1Y	2
U16	U16Y	1
U5	U17Y	8
U13	U20Y	1
U36	U20Y	2
U20	U20Y	3
U22	U33Y	9
U10	U1Y	2
U17	U17Y	1
U14	U20Y	1
U28	U1Y	3

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 4

FAULTLIST DOWNCOUNTER

```
(input stuck at 1)

PINFLTS;
U19:      A*   1;
U15:      A*   1;
U1:       A*   1;
U8:       A*   1;
U6:       A*   1;
U29:      A*   1;
U34:      A*   1;
U27:      A*   1;
U30:      A*   1;
U4:       A*   1;
U35:      A*   1;
U44:      A*   1;
U21:      A*   1;
U26:      A*   1;
U23:      A*   1;
U11:      A*   1;
U16:      A*   1;
U5:       A*   1;
U13:      A*   1;
U43:      A*   1;
U36:      A*   1;
U20:      A*   1;
U22:      A*   1;
U7:       A*   1;
U9:       A*   1;
U10:      A*   1;
U17:      A*   1;
U14:      A*   1;
U45:      A*   1;
U25:      A*   1;
U28:      A*   1;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 4

DOWNCOUNTER

(input stuck at 1)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
U19	U1Y	3
U15	U15Y	1
U29	U20Y	2
U34	U34Y	1
U27	U1Y	3
U35	U1Y	2
U21	U31Y	5
U23	U1Y	3
U11	U1Y	2
U16	U16Y	1
U5	U17Y	8
U13	U20Y	1
U36	U20Y	2
U20	U20Y	3
U22	U33Y	9
U10	U1Y	2
U17	U17Y	1
U14	U20Y	1
U25	U32Y	1

ADDRESS LOCATION	MACHINE #	CYCLE #

7.0 TEST

USER INPUT

7.1 PARTSLIST TEST (FIG. 1)

(TEST.PRT)

```
USER: "MCGOUGH"
NAME: TEST;
PURPOSE: TESTING;
LEVEL: SUBCHIP;
TYPES: T004M,T008M,T002M,T000M,T032M,C050MF;
EXT :: A1,B1,A2,B2,A3,B3,A4,B4,A5,B5,C5,P1,P2,P3,P4,P5;
INPUTS: .A1,.B1,.A2,.B2,.A3,.B3,.A4,.B4,.A5,.B5,.C5;
OUTPUTS: .P1,.P2,.P3,.P4,.P5;
P000M: IA1,IA2,IA3,IA4,IB1,IB2,IB3,IB4,IA5,IB5,IC5;
P002M: PO1,PO2,PO3,PO4,PO5;
T004M: INV1,INV2,INV3,INV4,INV5;
T008M: AND1;
T002M: NOR1;
T027M: NOR2;
T000M: NAND1;
T032M: OR1;
C050F: PA1,PA2,PA3,PA4,PA5,PA6;
END;
COMPSEGMENT;
       = A1*XA1,A2*XA2,A3*XA3,A4*XA4,B1*XB1,B2*XB2,B3*XB3,B4*XB4,
         A5*XA5,B5*XB5,C5*XC5,P1*XP1,P2*XP2,P3*XP3,P4*XP4,P5*XP5;
IA1 =  A*XA1,Y*IA1Y;
IA2 =  A*XA2,Y*IA2Y;
IA3 =  A*XA3,Y*IA3Y;
IA4 =  A*XA4,Y*IA4Y;
IA5 =  A*XA5,Y*IA5Y;
IB1 =  A*XB1,Y*IB1Y;
IB2 =  A*XB2,Y*IB2Y;
IB3 =  A*XB3,Y*IB3Y;
IB4 =  A*XB4,Y*IB4Y;
IB5 =  A*XB5,Y*IB5Y;
IC5 =  A*XC5,Y*IC5Y;
INV1 = A*IA1Y,Y*INV1Y;
PA1 =  A*IB1Y,Y*PA1Y;
AND1 = A*INV1Y,B*PA1Y,Y*AND1Y;
PA2 =  A*IA2Y,Y*PA2Y;
INV2 = A*IB2Y,Y*INV2Y;
NOR1 = A*PA2Y,B*INV2Y,Y*NOR1Y;
INV3 = A*IA3Y,Y*INV3Y;
PA3 =  A*IB3Y,Y*PA3Y;
NAND1 = A*INV3Y,B*PA3Y,Y*NAND1Y;
INV4 = A*IA4Y,Y*INV4Y;
PA4 =  A*IB4Y,Y*PA4Y;
OR1 =  A*INV4Y,B*PA4Y,Y*OR1Y;
INV5 = A*IA5Y,Y*INV5Y;
```

```
PA5 = A*IB5Y,Y*PA5Y;
PA6 = A*IC5Y,Y*PA6Y;
NOR2 = A*INV5Y,B*PA5Y,C*PA6Y,Y*NOR2Y;
PO1 = A*AND1Y,Y*XP1;
PO2 = A*NOR1Y,Y*XP2;
PO3 = A*NAND1Y,Y*XP3;
PO4 = A*OR1Y,Y*XP4;
PO5 = A*NOR2Y,Y*XP5;
ENDCOMPS;
DETECTSEGMENT;
  PINS;
       INV1Y;
       AND1Y;
       PA2Y;
       NOR1Y;
       PA3Y;
       NAND1Y;
       INV4Y;
       OR1Y;
       NOR2Y;
ENDDET;
ENDC;
END_OF_FILE;
```

7.2 P-ORDERING TEST

(DWN.B32)

```
XA1                                =                                ..A1;

XA2                                =                                ..A2;

XA3                                =                                ..A3;

XA4                                =                                ..A4;

XB1                                =                                ..B1;

XB2                                =                                ..B2;

XB3                                =                                ..B3;

XB4                                =                                ..B4;

XA5                                =                                ..A5;

XB5                                =                                ..B5;

XC5                                =                                ..C5;

 BUF(IA1Y,XA1);
 BUF(IA2Y,XA2);
 BUF(IA3Y,XA3);
 BUF(IA4Y,XA4);
 BUF(IA5Y,XA5);
```

```
BUF(IB1Y,XB1);
BUF(IB2Y,XB2);
BUF(IB3Y,XB3);
BUF(IB4Y,XB4);
BUF(IB5Y,XB5);
BUF(IC5Y,XC5);
NOTTF(INV1Y,IA1Y,ZERO,ZAND[0000000001]);
FNIBF(PA1Y,IB1Y,ZERO,ZERO);
AND2F(AND1Y,INV1Y,PA1Y,ZAND[0000000007],ZERO,ZERO,ZERO);
FNIBF(PA2Y,IA2Y,ZERO,ZERO);
NOTTF(INV2Y,IB2Y,ZERO,ZAND[0000000009]);
NOR2F(NOR1Y,PA2Y,INV2Y,ZAND[0000000010],ZERO,ZERO,ZERO);
NOTTF(INV3Y,IA3Y,ZAND[0000000011],ZERO);
FNIBF(PA3Y,IB3Y,ZERO,ZAND[0000000012]);
NAND2F(NAND1Y,INV3Y,PA3Y,ZERO,ZERO,ZAND[0000000013],ZERO);
NOTTF(INV4Y,IA4Y,ZERO,ZERO);
FNIBF(PA4Y,IB4Y,ZERO,ZAND[0000000015]);
OR2F(OR1Y,INV4Y,PA4Y,ZERO,ZERO,ZAND[0000000016],ZERO);
NOTTF(INV5Y,IA5Y,ZAND[0000000002],ZERO);
FNIBF(PA5Y,IB5Y,ZAND[0000000003],ZERO);
FNIBF(PA6Y,IC5Y,ZERO,ZERO);
NOR3F(NOR2Y,INV5Y,PA5Y,PA6Y,ZAND[0000000005],ZERO,ZERO,ZERO,ZERO);
NINV(XP1,AND1Y);
NINV(XP2,NOR1Y);
NINV(XP3,NAND1Y);
NINV(XP4,OR1Y);
NINV(XP5,NOR2Y);

.P1                                    =                                    .XP1;


.P2                                    =                                    .XP2;


.P3                                    =                                    .XP3;


.P4                                    =                                    .XP4;


.P5                                     =                                     .XP5


    END;

END
ELUDOM
```

USER INPUT

7.3 INPUT TEST

(TEST.DAT)

```
4                          { number of desired cycles. }
0,1,1,0,1,1,1,1,0,0,0
0,0,0,1,0,0,0,0,1,1,1
1,0,1,0,0,1,0,1,0,1,0
1,0,1,0,0,1,0,1,1,0,0
```

Column	Variables
1	A1
2	B1
3	A2
4	B2
5	A3
6	B3
7	A4
8	B4
9	A5
10	B5
11	C5

7.4 EXAMPLES

USER INPUT
EXAMPLE 1

FAULTLIST TEST

(TESTFLT.DAT)

```
(output stuck at 0)

PINFLTS;
INV1:    Y*      0;
INV5:    Y*      0;
PA5:     Y*      0;
PA6:     Y*      0;
NOR2:    Y*      0;
PA1:     Y*      0;
AND1:    Y*      0;
PA2:     Y*      0;
INV2:    Y*      0;
NOR1:    Y*      0;
INV3:    Y*      0;
PA3:     Y*      0;
NAND1:   Y*      0;
INV4:    Y*      0;
PA4:     Y*      0;
OR1:     Y*      0;
PA7:     Y*      0;
INV6:    Y*      0;
PA8:     Y*      0;
OR2:     Y*      0;
PA9:     Y*      0;
INV7:    Y*      0;
PA10:    Y*      0;
INV8:    Y*      0;
NOR3:    Y*      0;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 1

TEST

(DETECTED.DAT)

(output stuck at 0)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
INV1	INV1Y	1
INV5	NOR2Y	1
NOR2	NOR2Y	4
PA1	AND1Y	1
AND1	AND1Y	1
PA2	PA2Y	1
NOR1	NOR1Y	2
INV3	NAND1Y	3
PA3	PA3Y	1
NAND1	NAND1Y	1
INV4	INV4Y	2
PA4	OR1Y	1
OR1	OR1Y	1

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 2

FAULTLIST TEST

(TESTFLT.DAT)

```
(output stuck at 1)

PINFLTS;
INV1:    Y*      1;
INV5:    Y*      1;
PA5:     Y*      1;
PA6:     Y*      1;
NOR2:    Y*      1;
PA1:     Y*      1;
AND1:    Y*      1;
PA2:     Y*      1;
INV2:    Y*      1;
NOR1:    Y*      1;
INV3:    Y*      1;
PA3:     Y*      1;
NAND1:   Y*      1;
INV4:    Y*      1;
PA4:     Y*      1;
OR1:     Y*      1;
PA7:     Y*      1;
INV6:    Y*      1;
PA8:     Y*      1;
OR2:     Y*      1;
PA9:     Y*      1;
INV7:    Y*      1;
PA10:    Y*      1;
INV8:    Y*      1;
NOR3:    Y*      1;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 2

TEST

(DETECTED.DAT)

(output stuck at 1)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
INV1	INV1Y	3
INV5	NOR2Y	4
PA5	NOR2Y	4
PA6	NOR2Y	4
NOR2	NOR2Y	1
PA1	AND1Y	2
AND1	AND1Y	2
PA2	PA2Y	2
INV2	NOR1Y	2
NOR1	NOR1Y	1
INV3	NAND1Y	1
PA3	PA3Y	2
NAND1	NAND1Y	3
INV4	INV4Y	1

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 3

FAULTLIST TEST

(TESTFLT.DAT)

```
(input stuck at 0)

PINFLTS;
INV1:   A*      0;
INV5:   A*      0;
PA5:    A*      0;
PA6:    A*      0;
NOR2:   A*      0;
PA1:    A*      0;
AND1:   A*      0;
PA2:    A*      0;
INV2:   A*      0;
NOR1:   A*      0;
INV3:   A*      0;
PA3:    A*      0;
NAND1:  A*      0;
INV4:   A*      0;
PA4:    A*      0;
OR1:    A*      0;
PA7:    A*      0;
INV6:   A*      0;
PA8:    A*      0;
OR2:    A*      0;
PA9:    A*      0;
INV7:   A*      0;
PA10:   A*      0;
INV8:   A*      0;
NOR3:   A*      0;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 3

TEST

(DETECTED.DAT)

(input stuck at 0)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
NOR2	NOR2Y	4
NAND1	NAND1Y	1
INV4	INV4Y	2
OR1	OR1Y	2

ADDRESS LOCATION	MACHINE #	CYCLE #

USER INPUT
EXAMPLE 4

FAULTLIST TEST

(TESTFLT.DAT)

```
(input stuck at 1)

PINFLTS;
INV1:   A*      1;
INV5:   A*      1;
PA5:    A*      1;
PA6:    A*      1;
NOR2:   A*      1;
PA1:    A*      1;
AND1:   A*      1;
PA2:    A*      1;
INV2:   A*      1;
NOR1:   A*      1;
INV3:   A*      1;
PA3:    A*      1;
NAND1:  A*      1;
INV4:   A*      1;
PA4:    A*      1;
OR1:    A*      1;
PA7:    A*      1;
INV6:   A*      1;
PA8:    A*      1;
OR2:    A*      1;
PA9:    A*      1;
INV7:   A*      1;
PA10:   A*      1;
INV8:   A*      1;
NOR3:   A*      1;
ENDPINS;
ENDC;
END_OF_FILE;
```

PROGRAM OUTPUT
EXAMPLE 4

TEST

(DETECTED.DAT)

(input stuck at 1)

PIN FAULTS DETECTED :

FAULT NAME	TEST PIN #	CYCLE #
INV1	INV1Y	3
INV5	NOR2Y	1
NOR2	NOR2Y	1
INV2	NOR1Y	2
INV3	NAND1Y	3
PA3	PA3Y	2
NAND1	NAND1Y	1
OR1	OR1Y	1

ADDRESS LOCATION	MACHINE #	CYCLE #

8.0 MEMORY CIRCUIT

USER INPUT

8.1 PARTSLIST -- MEMORY CIRCUIT (FIG. 5)

```
USER:          "NEMEROFF";
NAME:          MEMRYRW;
PURPOSE:       PRAC;
LEVEL:         CHIP;
TYPES:         MEMR,          C050F,          P000M,
               T004M,         P002M;
EXT::          E0,     E1,    E2,     E3,     E4,
               E5,     P0,    P1,     P2,     P3;
INPUTS:        E0,     E1,    E2,     E3,     E4,     E5;
OUTPUTS:       P0,     P1,    P2,     P3;
P000M:         U0      U1,    U2,     U3,     U4;
T004M:         U10;
C050F:         U6,     U7,    U8,     U9;
MEMR:          U5,     U11;
P002M:         P01,    P02,   P03,    P04;
END;
COMPSEGMENT;
               = E0*XE0,      E1*XE1,         E2*XE2,         E3*XE3,
                 E4*XE4,      E5*XE5,         P0*XP0,         P1*XP1,
                 P2*XP2,      P3*XP3;
P01            = A*U11Y1,     Y*XP0;
P02            = A*U11Y2,     Y*XP1;
P03            = A*U11Y3,     Y*XP2;
P04            = A*U11Y4,     Y*XP3;
U0             = A*XE5,       Y*U0Y;
U1             = A*XE0,       Y*U1Y;
U2             = A*XE1,       Y*U2Y;
U3             = A*XE2,       Y*U3Y;
U4             = A*XE3,       Y*U4Y;
U10            = A*XE4,       Y*U10Y;
U6             = A*U5Y1,      Y*U6Y;
U7             = A*U5Y2,      Y*U7Y;
U8             = A*U5Y3,      Y*U8Y;
U9             = A*U5Y4,      Y*U9Y;
U5             = A0*U1Y,      A1*U2Y,         A2*U3Y,         A3*U4Y,
                 EN*XE4,      Y1*U5Y1,        Y2*U5Y2,        Y3*U5Y3,
                 Y4*U5Y4;
 U11           = A0*U6Y,      A1*U7Y,         A2*U8Y,         A3*U9Y,
                 EN*U10Y,     RW*U0Y,         D0*U1Y,         D1*U2Y,
                 D2*U3Y,      D3*U4Y,         Y1*U11Y1,       Y2*U11Y2,
                 Y3*U11Y3,    Y4*U11Y4;
```

```
ENDCOMPS;
DETECTSEGMENT;
PINS:
 U1Y;
 U2Y;
 U3Y;
 U4Y;
ADDRESSES;
0001;
0003;
0012;
0013;
0014;
0015;
0002;
0010;
0011;
ENDDET;
ENDC;
END_OF_FILE;
```

8.2 P-ORDERING -- MEMORY CIRCUIT

(DWN.B32)

```
  XE0 = ..E0;
XE1 = ..E1;
XE2 = ..E2;
XE3 = ..E3;
XE4 = ..E4;
XE5 = ..E5;
BUF(U0Y,XE5);
BUF(U1Y,XE0);
BUF(U2Y,XE1);
BUF(U3Y,XE2);
BUF(U4Y,XE3);
NOTT(U10Y,XE4);
MEMR(A0=U1Y,A1=U2Y,A2=U3Y,A3=U4Y,EN=XE4,Y1=U5Y1,Y2=U5Y2,Y3=U5Y3,Y4=U5Y4);
FNIB(U6Y,U5Y1);
FNIB(U7Y,U5Y2);
FNIB(U8Y,U5Y3);
FNIB(U9Y,U5Y4);
MEMR(A0=U6Y,A1=U7Y,A2=U8Y,A3=U9Y,EN=U10Y,RW=U0Y,D0=U1Y,D1=U2Y,D2=U3Y,
D3=U4Y,Y1=U11Y1,Y2=U11Y2,Y3=U11Y3,Y4=U11Y4);
NINV(XP0,U11Y1);
NINV(XP1,U11Y2);
NINV(XP2,U11Y3);
NINV(XP3,U11Y4);
.P0 = .XP0;
.P1 = .XP1;
.P2 = .XP2;
.P3 = .XP3
```

USER INPUT

8.3 INPUT -- MEMORY CIRCUIT

```
48
0,0,0,0,1,0
9,9,9,9,0,1
9,9,9,9,0,0
0,0,0,1,1,0
9,9,9,9,0,1
9,9,9,9,0,0
0,0,1,0,1,0
9,9,9,9,0,1
9,9,9,9,0,0
0,0,1,1,1,0
9,9,9,9,0,1
9,9,9,9,0,0
0,1,0,0,1,0
9,9,9,9,0,1
9,9,9,9,0,0
0,1,0,1,1,0
9,9,9,9,0,1
9,9,9,9,0,0
0,1,1,0,1,0
9,9,9,9,0,1
9,9,9,9,0,0
0,1,1,1,1,0
9,9,9,9,0,1
9,9,9,9,0,0
1,0,0,0,1,0
9,9,9,9,0,1
9,9,9,9,0,0
1,0,0,1,1,0
9,9,9,9,0,1
9,9,9,9,0,0
1,0,1,0,1,0
9,9,9,9,0,1
9,9,9,9,0,0
1,0,1,1,1,0
9,9,9,9,0,1
9,9,9,9,0,0
1,1,0,0,1,0
9,9,9,9,0,1
9,9,9,9,0,0
1,1,0,1,1,0
9,9,9,9,0,1
9,9,9,9,0,0
1,1,1,0,1,0
9,9,9,9,0,1
9,9,9,9,0,0
1,1,1,1,1,0
9,9,9,9,0,1
9,9,9,9,0,0
```

-- NUMBER OF DESIRED CYCLES --

COLUMN	VARIABLES
1	E0
2	E1
3	E2
4	E3
5	E4
6	E5

8.4 INITITAL MEMORY DATA

```
0,15,0,15,100,000,000;
RADIX,DECIMAL;
000,000003,000004,000005,000006,000007,000008;
006,000009,000010,000011,000012,000013,000014;
012,000015,000000,000001,000002;
END
```

8.5 FAULTLIST -- MEMORY CIRCUIT

```
PINFLTS;
ENDPINS;
MEMFLTS;
000000*       2;
000002*       1;
000010*       3;
000011*       1;
000012*       2;
000013*       2;
000014*       3;
000015*       1;
ENDMEM;
ENDC;
END_OF_FILE;
```

8.6 OUTPUT -- MEMORY CIRCUIT

P0	P1	P2	P3
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	FFFFFFFF
00000000	00000000	00000000	FFFFFFFF
00000000	00000000	00000000	FFFFFFFF
00000000	00000000	FFFFFFFF	00000000
00000000	00000000	FFFFFFFF	00000000
00000000	00000000	FFFFFFFF	00000000
00000000	00000000	FFFFFFFF	FFFFFFFF
00000000	00000000	FFFFFFFF	FFFFFFFF
00000000	00000000	FFFFFFFF	FFFFFFFF
00000000	FFFFFFFF	00000000	00000000
00000000	FFFFFFFF	00000000	00000000
00000000	FFFFFFFF	00000000	00000000
00000000	FFFFFFFF	00000000	FFFFFFFF
00000000	FFFFFFFF	00000000	FFFFFFFF
00000000	FFFFFFFF	00000000	FFFFFFFF
00000000	FFFFFFFF	FFFFFFFF	00000000
00000000	FFFFFFFF	FFFFFFFF	00000000
00000000	FFFFFFFF	FFFFFFFF	00000000
00000000	FFFFFFFF	FFFFFFFF	FFFFFFFF
00000000	FFFFFFFF	FFFFFFFF	FFFFFFFF

00000000	FFFFFFFF	FFFFFFFF	FFFFFFFF
FFFFFFFF	00000000	00000000	00000000
FFFFFFFF	00000000	00000000	00000000
FFFFFFFF	00000000	00000000	00000000
FFFFFFFF	00000000	00000000	FFFFFFFF
FFFFFFFF	00000000	00000000	FFFFFFFF
FFFFFFFF	00000000	00000000	FFFFFFFF
FFFFFFFF	00000000	FFFFFFFF	00000000
FFFFFFFF	00000000	FFFFFFFF	00000000
FFFFFFFF	00000000	FFFFFFFF	00000000
FFFFFFFF	00000000	FFFFFFFF	FFFFFFFF
FFFFFFFF	00000000	FFFFFFFF	FFFFFFFF
FFFFFFFF	00000000	FFFFFFFF	FFFFFFFF
FFFFFFFF	FFFFFFFF	00000000	00000000
FFFFFFFF	FFFFFFFF	00000000	00000000
FFFFFFFF	FFFFFFFF	00000000	00000000
FFFFFFFF	FFFFFFFF	00000000	FFFFFFFF
FFFFFFFF	FFFFFFFF	00000000	FFFFFFFF
FFFFFFFF	FFFFFFFF	00000000	FFFFFFFF
FFFFFFFF	FFFFFFFF	FFFFFFFF	00000000
FFFFFFFF	FFFFFFFF	FFFFFFFF	00000000
FFFFFFFF	FFFFFFFF	FFFFFFFF	00000000
FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF

8.7 MEMORY DETECTION RESULT

ADDRESS LOCATION	MACHINE #	CYCLE #
000000	0003	2
000010	0013	17
000011	0012	41
000012	0015	11
000014	0001	23
000015	0002	47

9.0 SUBROUTINES, MODULES & VARIABLES

PROGRAM : GLOSS.FOR

MAIN VARIABLES:

INTEGER:

I	= Index variable.
I1	= Index variables used to find
I2	= corresponding types for components.
ICOL	= Column position of parser.
IK	= Index to reference knowns in p-ordered list.
IO	= Call to function INOUT.
IONE	= 1.
ISTAT	= Sets cursor to a specified position.
ITKLN	= Length of variable ITOK.
J	= Index variable.
J1	= Index variable for OUT(NUM,J1,0).
J2	= Index variable for IN (NUM,J2,0).
MAIN	= Position of Main component in partslist.
NC	= Number of characters read into NCARD.
NCLOCKS	= Number of ficticious clocks.
NI	= Number of external inputs in main circuit.
NK	= Number of nets whose values are known.
NO	= Number of external outputs in main circuit.
NTYPES	= Number of types.
NUM	= Number of components on partslist.
ZERO	= Constant set to zero for fault input.
IA	= Index variable.
NF	= Number of faults.
IB	= Index variable.

BOOLEAN:

CK	= True if part is a ficticious clock.

CHARACTER:

EXTNAM	= External pins name.
NAME	= Name of circuit.
NET	= Net being examined.
PIN	= Pin being examined.
T	= Temporary variable for TYPE(I1,I2).
TOKEN	= Token returned by parser (=ITOK).
T1	= Temporary variable for FLT(NF,1).

BYTE:

	ITOK	= Token returned by parser.
	NCARD	= 80 character line read in to be parsed.

INTEGER ARRAY:

(100)	NETIN	= Number of inputs to a particular component.
(100)	NETOUT	= Number of outputs to a particular component.
(100)	NIN	= Position of last input in LIB.
(100)	NOUT	= Position of last output in LIB.
(20)	NPARTS	= Number of parts for each type.

BOOLEAN ARRAY:

(100)	DONE	= False if part is not yet in p-ordered list

CHARACTER ARRAY:

(0:20)	CH_NUM	= Character equivalents of integers.
(20, 2)	CLOCK	= List of input and output nets of the ficticious clocks.
(100, 20, 0:1)	IN	= Array of inputs to a particular component.
(100)	INPIN	= Array of inputs to main circuit.
(500)	KNOWN	= List of known external inputs and clock outputs.
(50, 0:10)	LIB	= Library.
(150)	LIST1	= Unordered list.
(150)	LIST2	= P-ordered list.
(100)	LOG	= Array of logical equations for each component.
(50)	LOGIC	= Array of logical equations for each type.
(100, 20, 0:1)	OUT	= Array of outputs to a particular component.
(100)	OUTPIN	= Array of outputs to main circuit.
(100, 2)	PART	= Array of all components on partslist.
(30, 0:50)	TYPE	= Array of all types associated w/ parts on partslist.
(100,0:5)	FLT	= Array of all faults and their values.

SUBROUTINES CALLED:

CREATE_DWN	= Creates bliss coded P-ordered program.
CREATE_EXEC	= Creates fortran coded executive program.
CREATE_MAIN	= Creates bliss coded control program.
CREATE_PRINT	= Creates fortran coded program to print results.
GETOK	= Parses a given line.
INIT_CH_NUM	= Converts integers to characters.
LIBARY	= Reads in library of cells and their logic.
SUBSTITUTE	= Substitutes parts, their inputs and their outputs into the proper logical macro equation.
SET_FLT	= Reads faults from partslist and stores them in array FLT.

CLPS_FLT	= Collapses faults.
FAULT_SUB	= Substitutes faulted parts,their inputs and their outputs into the proper logical macro equation.
SET_MEM	= reads in memory input data.
CREATE_MEM	= parses memory input data and stores in array FLT.
CR_FLTPRN	= creates subroutine for printing results of fault detection.
CREATE_RWM	= creates bliss module for simulating random access memories.
RWMEM	= creates bliss module for simulating read only memories.
CREATE_PAS	= creates routine for passing results from the bliss module DETFLT (DETECT) to the routine FLTPRN.
CREATE_DET	= creates a bliss module for detecting whether or not a fault has been injected. The module routine called DETECT is performed after every cycle.

FUNCTIONS CALLED:

(INTEGER)	IFORM	= Determines format of a line in the partslist.
(INTEGER)	INOUT	= Determines whether a net is an input or an output net.

LIBARY :

This routine reads in the contents of the standard library of components .

VARIABLES:

INTEGER:

	I	= Index variable.
(main)	ICOL	= Column position of parser.
(main)	ITKLN	= Length of variable ITOK.
	J	= Index variable.
	K	= Index variable for read statement.
(main)	NC	= Number of characters read into NCARD.

CHARACTER:

(main)	TOKEN	= Token returned by parser (=ITOK).

BYTE:

(main)	ITOK	= Token returned by parser.
(main)	NCARD	= 80 character line read in to be parsed.

INTEGER ARRAY:

(main)	NIN	= Position of last input in LIB.
(main)	NOUT	= Position of last output in LIB.

CHARACTER ARRAY:

(main)	LIB	= Library.
(main)	LOGIC	= Array of logical equations for each type.

SUBROUTINES CALLED:

GETOK	= Parses a given line.

GETOK :

This routine parses an 80 column line and returns the first word of characters stripped of all punctuation marks. It's length is stored in ITKLN, and the length of the line to be parsed is stored in NC. Words are separated by any of the following punctuation marks or characters: "S", "SS", ";", ",", ":", "*", "=", ".", where "S" and "SS" denote single and double spaces, respectively.

VARIABLES:

INTEGER:

(main)	ICOL	= Column position of parser.
(main)	ITKLN	= Length of variable ITOK.
	J	= Index variable.

BYTE:

(main)	ITOK	= Token returned by parser.
(main)	NCARD	= 80 character line read in to be parsed.

IFORM :

Used with GETOK to determine format of word currently being parsed.

VARIABLES:

INTEGER:

IPOS = Column position of IFORM parser in string.

BYTE:

(main) NCARD = 80 character line read in to be parsed.

INOUT :

Determines whether a pin is an input or output pin and returns 0 if an output pin and 1 if an input pin.

VARIABLES:

INTEGER:

	I	= Index variable.
	IN1	= Position of a component's first input pin in LIB.
	IN2	= Position of a component's last input pin in LIB.
	IOUT1	= Position of a component's first output pin in LIB.
	IOUT2	= Position of a component's last output pin in LIB.
	J	= Index variable
(main)	NI	= Number of inputs in main circuit.
(main)	NO	= Number of outputs in main circuit.
(main)	NUM	= Number of components on partslist.

CHARACTER:

(main)	EXTNAM	= External pins name.
	L	= Temporary for LIB(I,J).
	PIN	= The pin to be checked.
(main)	TOKEN	= Token returned by parser (=ITOK).
	TYPE	= The type of the part which the pin belongs to.

BYTE:

(main)	ITOK	= Token returned by parser.
(main)	NCARD	= 80 character line read in to be parsed.

INTEGER ARRAY:

(main)	NIN	= Position of last input in LIB.
(main)	NOUT	= Position of last output in LIB.

CHARACTER ARRAY:

(main)	INPIN	= Array of inputs to main circuit.
(main)	LIB	= Library.
(main)	LOG	= Array of logical equations for each component.
(main)	LOGIC	= Array of logical equations for each type.
(main)	OUTPIN	= Array of outputs to main circuit.

INIT_CH_NUM :

This routine finds the ASCII equivalents of integers.

VARIABLES:
CHARACTER ARRAY:

(main) CH_NUM = Character equivalents of integers.

SET_FLT :

Parses the fault data file. Fault data is stored in the array FLT.

VARIABLES:

INTEGER:

	I	= Index variable.
	J	= Index variable.
	K	= Index variable.
	NF	= Number of faults.
(main)	ITKLN	= Length or variable ITOK.
(main)	ICOL	= Column position of parser.
(main)	NC	= Number of characters read into NCARD.

CHARACTER:

	ANS	= Temporary variable.
(main)	CHNM	= Character equivalent of integer.
	T	= Temporary variable.
	TEMP	= Temporary variable.
	TOKN1	= Token returned by parser (= ITOK).

BYTE ARRAYS:

(80)	ITOK1	= Token returned by parser.
(80)	N1CARD	= 80 character line read in to be parsed.

CHARACTER ARRAYS:

(0:20)	(main)	CH_NUM	= Character equivalent of integer.
(100,0:5)	(main)	FLT	= Array of all faults and their values.

SUBROUTINES REFERENCED:

GETOK.
IN_CH_NM.

CLPS_FLT :

Used to find unique faults.

VARIABLES:

INTEGER:

NM	=	Number of faults.
NZ	=	Number of faults.

CHARACTER:

T	=	Temporary variable.
T1	=	Temporary variable.
L	=	Logic of gate.

CHARACTER ARRAYS:

(100,0:5)(main) FLT	=	Array of faults and their values.

IN_CH_NM :

For finding ASCII equivalents of integers. Uses data stored in INIT_CH_NM.

VARIABLES:

INTEGER:

P1	=	Temporary variable.
P2	=	Temporary variable.
P3	=	Temporary variable.
P4	=	Temporary variable.
NF	=	Number of faults.

CHARACTER:

C1	=	Temporary character variable.
C2	=	Temporary character variable.
C3	=	Temporary character variable.
C4	=	Temporary character variable.

CHARACTER ARRAYS:

(0:20) (main) CH_NUM = Character equivalent of integer.

CREATE_ZND :

Sets input fault values into bliss arrays for substitution in place of non-faulted inputs, it creates the bliss module ZND.B32.

VARIABLES:
INTEGER:
NF = Number of faults.

ZND.B32 :

ZAND = vector containing the input vaules for the expanded (faulted) gates.

SUBSTITUTE :

Substitutes inputs and outputs of each part(except for faulted and memory parts) into the bliss logical phrase for its type.

VARIABLES:

INTEGER:

	I	= Index variable.
	IC	= Column position.
	IJ	= Length of line to be printed.
(main)	IK	= Index to reference knowns in p-ordered list.
	IPOS	= Index variable for OOUT and IIN.
	J	= Index variable.
	LN	= Length of net.
	LPIN	= Length of pin to be replaced by a net.
	M	= Index variable.
(main)	NCLOCKS	= Number of ficticious clocks.
(main)	NI	= Number of external inputs in main circuit.
(main)	NUM	= Index number of "part" being substituted

CHARACTER:

	IIN	= Input string being parsed and substituted.
(main)	L	= Logical phrase from library.
	NO	= Output net.
	N1	= Input net.
	OOUT	= Output string being parsed and substituted.
(main)	PIN	= Pin being replaced.
(main)	PRINT	= Variable containing line to be printed.
	T	= Temporary variable.

INTEGER ARRAY:

(main)	NETIN	= Number of inputs to a particular component.
(main)	NETOUT	= Number of outputs to a particular component.
(100)	NINP	= Number of inputs for each part.
(100)	NOUTP	= Number of outputs for each part.

CHARACTER ARRAY:

(main)	CH_NUM	= Character equivalents of integers.
(main)	CLOCK	= List of input and output nets of the ficticious clocks.
(main)	IN	= Array of inputs to a particular component.
(100, 10)	INP	= Array of inputs to a particular component.
(main)	LIST1	= Unordered list.
(main)	OUT	= Array of outputs to a particular component.
(100, 10)	OUTP	= Array of outputs to a particular component.

FAULT_SUB :

Substitutes the inputs and outputs of faulted parts into the faulted logic for its type.

VARIABLES:

INTEGER:

	I	= Index variable.
	IC	= Column position.
	IJ	= Lenth of line to be printed.
(main)	IK	= Index to reference knowns in p-ordered list.
	IPOS	= Index variable for OOUT and IIN.
	J	= Index variable.
	LN	= Length of net.
	LPIN	= Length of pin to be replaced by a net.
	M	= Index variable.
(main)	NCLOCKS	= Number of ficticious clocks.
(main)	NI	= Number of external inputs in main circuit.
(main)	NUM	= Index number of "part" being substituted.
(main)	NF	= Number of faults.
	TK	= Temporary variable.

CHARACTER:

	IIN	= Input string being parsed and substituted.
(main)	L	= Logical phrase from library.
	NO	= Output net.
	N1	= Input net.
	OOUT	= Output string being parsed and substituted.
(main)	PIN	= Pin being replaced.
(main)	PRINT	= Variable containing line to be printed.
	T	= Temporary variable.
	NPT	= Name of faulted part.
	NPN	= Name of faulted pin.
	T1	= Temporary variable.
	N2	= Temporary variable.
	N3	= Temporary variable.

INTEGER ARRAY:

	(main)	NETIN	= Number of inputs to a particular component.
	(main)	NETOUT	= Number of outputs to a particular component.
(100)		NINP	= Number of inputs for each part.
(100)		NOUTP	= Number of outputs for each part.

CHARACTER ARRAY:

(main) CH_NUM = Character equivalents of integers.
(main) CLOCK = List of input and output nets of the ficticious clocks.
(main) IN = Array of inputs to a particular component.
(100, 10) INP = Array of inputs to a particular component.
(main) LIST1 = Unordered list.
(main) OUT = Array of outputs to a particular component.
(100, 10) OUTP = Array of outputs to a particular component.
(100,0:5)(main) FLT = Array of faults and their values.

```
CREATE_DWN :

        This routine creates the bliss emulation module DWN.B32. It
        calls the bliss macros which represent the gates and memories.

  VARIABLES:
        INTEGER:

                I       = Index variable.
                J       = Index variable.
                K       = Number of knowns at beginning of execution.
                MAIN    = Position of Main component in partslist.
        (main)  NI      = Number of inputs in main circuit.
        (main)  NO      = Number of outputs in main circuit.
        (main)  NUM     = Index number of "part" being substituted

        CHARACTER:

        (main)  L       = Logical phrase from library.


        INTEGER ARRAY:

        (main)  NETIN   = Number of inputs to a particular component.
  (substitute)  NOUTP   = Number of outputs for each part.


        CHARACTER ARRAY:

        (main)  IN      = Array of inputs to a particular component.
        (main)  INPIN   = Array of inputs to main circuit.
        (main)  LIST1   = Unordered list.
(150)           LIST3   = List of variables in ST array (in DWN.B32).
        (main)  OUTP    = Array of outputs to a particular component.
        (main)  OUTPIN  = Array of outputs to main circuit.


        SUBROUTINES CALLED:

                P_ORDER         = P_orders LIST1.
```

P_ORDER :

This routine (called from CREATE_DWN) sets up the macro calls so that they may be performed in the order required by the circuit and writes them into DWN.B32.

VARIABLES:

INTEGER:

	I	= Index variable.
	IX1	= Index variable.
	IX2	= Index variable.
	IX3	= Index variable.
	IX4	= Index variable.
	IX5	= Index variable.
	IX6	= Index variable.
	J	= Index variable.
	K	= Index variable.
(main)	MAIN	= Position of Main component in partslist.
	MNO	= Number of passes made while p_ordering.
	NFC	= Number of ficticious clocks.
(main)	NK	= Number of nets whose values are known.
(main)	NUM	= Index number of "part" being substituted
(main)	NUM	= Number of components on partslist.

CHARACTER:

(main)	L	= Logical phrase from library.
	T	= Temporary variable for INP(IX1,IX2).
	T1	= Temporary variable for IN(MAIN,I,1).

BOOLEAN:

	FIN	= True when list is successfully p_ordered.

INTEGER ARRAY:

(80)		LISFC	= List of ficticious clocks.
	(main)	NETIN	= Number of inputs to a particular component.
	(main)	NETOUT	= Number of outputs to a particular component.
(substitute)		NINP	= Number of inputs for each part.
(substitute)		NOUTP	= Number of outputs for each part.

CHARACTER ARRAY:

(main)	IN	= Array of inputs to a particular component.
(substitute)	INP	= Array of inputs to a particular component.
(main)	LIST1	= Unordered list.
(main)	LIST2	= P-ordered list.
(main)	OUT	= Array of outputs to a particular component.
(substitute)	OUTP	= Array of outputs to a particular component.

BOOLEAN ARRAY:

(main)	DONE	= False if part is not yet in p-ordered list

CREATE_EXEC :

Creates the subprogram EXEC.For which sets up the output headings in OUTPUT.DAT and calls the bliss module CTRL which controls the execution of the circuit.

VARIABLES:

INTEGER:

	I	= Index variable.
(main)	NO	= Number of outputs in main circuit.

CHARACTER ARRAY:

(100)	NAME	= Output column headings.
(main)	OUTPIN	= Array of outputs to main circuit.

CREATE_MAIN :

Creates MAIN.B32 ,the main bliss module which contains the routine CTRL . CTRL keeps track of cycles, changes in input ,calls ZND.B32 and MEM.B32 to load their data, calls DWN.B32 to run the actual emulation, calls detect to determaine if any faults were detected, calls PASPIN.FOR and PASADD.FOR to pass in detected results and calls PRINT.FOR and FLTPRN.FOR to print the results of the emulation and the fault detection.

VARIABLES:

INTEGER:

	I	= Index variable.
	IBEG	= Start index for cycle in MAIN.B32.
	IEND	= End index for cycle in MAIN.B32.
(main)	IONE	= 1.
(main)	ISTAT	= Sets cursor to a specified position.
	J	= Index variable.
	K	= Index variable.
	M	= Total number of cycles to be simulated.
	NCYCLE	= Cycle in which input values are to be changed.
(main)	NI	= Number of external inputs in main circuit.
(main)	NO	= Number of external outputs in main circuit.

CHARACTER:

	ANS	= Update input? Y or N
	QU	= Input manually or from a file? M or F

INTEGER ARRAY:

(100)	NUM	= Initial input valuas for circuit.

CHARACTER ARRAY:

(main)	INPIN	= Array of inputs to main circuit.
(main)	OUTPIN	= Array of outputs to main circuit.

MAIN.B32 :

(for explanation see CREATE_MAIN page)

VARIABLES:

AMACH = contains values of machines in which address faults were detected.

ACYCLE = contains values of cycles in which address faults were detected.

PMACH = contains values of machines in which pin faults were detected.

PCYCLE = contains values of cycles in which pin faults were detected.

MFTADD = contains values of addresses of words to be faulted in ROM.

MFTBIT = contains values of the bits of the words in MFTADD which are to be flipped,thus faulting ROM.

ST = contains values of temporary variables used in emulation (ie ficticious clocks).

ROUTINES CALLED:

PRINT = fortran routine to print the results of emulation.

ZND = loads the faulted gate input data into the vector ZAND.

PRMS = read only memory simulation routine.

RWMEM = random access memory simulation routine.

DETFLT = routine to detect injected faults. Stores results in AMACH,ACYCLE,PMACH and PCYCLE.

FLTPRN = prints data in AFINF and PFINF to file named DETECTED.DAT.

PASADD = passes data in AMACH and ACYCLE to the fortran array AFINF used by FLTPRN.

PASPIN = passes data in PMACH and PCYCLE to the fortran array PFINF used by FLTPRN.

INTRAM = makes 32 copies of RAM.

MEM = loads data into memory, calls INTRAM.

CREATE_PRINT :

This routine creates a fortran subroutine PRINT.FOR to print out the results of the emulation.

VARIABLES:

INTEGER:

	I	= Index variable.
(main)	NO	= Number of external outputs in main circuit.

CHARACTER ARRAY:

(main)	INPIN	= Array of inputs to main circuit.
(main)	OUTPIN	= Array of outputs to main circuit.

SET_MEM :

Parses the memory data file (FOR006.DAT).

VARIABLES:

CHARACTER:

(main)	TOKEN	= Token returnes by parser (= ITOK).
	LIB*10	= Temporary variable.
	Q1	= Temporary variable.

CHARACTER ARRAY:

(main)	LIB	= Library.
(main)	LOGIC	= Arrary of logical equations for each type.
	TMP1	= temporary array of memory data.

INTEGER ARRAY:

(main)	NIN	= Position of last input in Lib.
(main)	NOUT	= Position of last output in Lib.

BYTE:

(main)	ITOK	= Token returned by parser.
(main)	NCARD	= 80 character line read in to be parsed.

RWMEM :

Creates PRM.B32, the bliss module that emulates a read-only memory. If ROM is to be faulted it makes 32 copies of ROM in a scratchpad memory. In the fault list (see page) the user specifies the word in memory and the bit to be faulted and faults are injected by flipping the bit of the chosen word of memory to 0 if it was 1 ,or to 1 if it was 0.

VARIABLES:

MASK1 = vector containing the bit mask for the serial to parallel conversion.

ADDARY= vector containing the address in memory to be read from or written to.

into memory.

ENB = enable bit. If 1 then read ROM else end.

OUTARY = vector containing memory output data.

MFTADD = vector containing the addresses of the words in memory to be faulted (if ROM faults are desired).

MFTBIT = vector containing the value of the bits to be flipped (from 0 to 1 or 1 to 0) in ROM thus causing the chosen words of memory to be faulted.

ZMIM = vector containing 32 copies of ROM with the faults injected.

ISTART= if ISTART = 1 then fault ROM else no ROM faults are desired.

MN1 = vector containg the displacement to be compensated for if the given addresses in memory are not consecutive.

MN2 = vector containing the addresses in bliss memory to be used after the displacement is taken into account.

MNN = contains the number of entries in MN1.

MACRO XLATE = bliss macro to convert from serial to parallel and vice versa.

CREATE_RWM :

Create RWM.B32 , the bliss module that reads or writes from RAM. If the read/write bit is 1 (on), the write routine is called else the read routine is done.

VARIABLES:

MASK1 = vector containing the bit mask for the serial to parallel conversion.

ADDARY= vector containing the address in memory to be read from or written to.

DATARY= vector containing the data to be written into memory.

ENB = enable bit. If 1 then proceed else end.

RWB = read/write bit. If 1 then write else read.

WMACMEM = RAM scratch-pad memory. loaded in module MEM.B32, it is copied 32 times so it can be faulted.

MN1 = vector containg the displacement to be compensated for if the given addresses in memory are not consecutive.

MN2 = vector containing the addresses in bliss memory to be used after the displacement is taken into account.

MNN = contains the number of entries in MN1.

OUTARY = vector containing memory output data.

MACRO XLATE = bliss macro to convert from serial to parallel and vice versa.

CREATE_PAS :

This routine creates two fortran modules PASPIN.FOR and PASADD.FOR. These routines pass the detected fault information contained in the bliss vectors PCYCLE ,PMACH ,ACYCLE and AMACH to to the fortran arrays PFINF and AFINF so that it can be formatted and printed out by the routine FLTPRN.FOR

VARIABLES :

PFINF = character array containing detected pin faults and the cycle in which they were discovered.

AFINF = character array containing detected address faults and the cycle in which they were discovered.

FLTPRN.FOR

This routine prints out the fault detection data contained in the arrays PFINF and AFINF. The output file which then contains the information is named DETECTED.DAT. This routine is created by CR_FLTPRN a GLOSS.FOR subroutine.

VARIABLES :

PFINF = character array containing detected pin faults and the cycle in which they were discovered.

AFINF = character array containing detected address faults and the cycle in which they were discovered.

```
CH_IN_NM :

Uses IN_NUM to convert characters to integers.

  VARIABLES:
        CHARACTER:


                CH      = Holds value to be returned.
                TEMP    = Temporary variable.

        FUNCTIONS REFERENCED:


                IN_NUM.
```

FUNCTION IN_NUM :

(returns an integer equivalent of a character)

VARIABLES:
CHARACTER:

C = Character that is input parameter.

CREATE_MEM :

This routine creates a bliss module to store the memory input data that is read in from the user input file. The bliss module created is named MEM.B32

VARIABLES:

INTEGER:

RADIX	=	Value of memory index(octal or decimal).
NUMLOC	=	Place of current storage for memory data.
LRW	=	Integer equivalent of LRWMEM.
HIW	=	Integer equivalent of HIWMEM.
LOD	=	Integer equivalent of LODMEM.
HIADD	=	Integer equivalent of HIDMEM.

CHARACTER:

CH_NUM	=	Character equivalents of integers.
NUMR	=	Temporary variable.
TEMP2	=	Temporary variable.
LODMEM	=	Starting address for read only memory.
HIDMEM	=	Ending address for read only memory.
LRWMEM	=	Starting address for read/write memory.
HIWMEM	=	Ending address for read/write memory.
ENDPC	=	Ending address for program counter.
WMEMBS	=	Base address for read/write memory.
TOKEN	=	Token returned by parser(= ITOK).
CHNM	=	Character equivalents of integers.
CHEM	=	Character equivalents of integers.

SUBROUTINES CALLED:

GETOK	=	Parses a given line.
IN_CH_NM	=	Returns a character equivalent of an integer.
NUMDEC	=	Sets radix for decimal memory data.
NUMOCT	=	Sets radix for octal memory data.
CH_IN_NM	=	Returns an integer equivalent of a character.

MEM.B32 :

(for explanation see CREATE_MEM above)

VARIABLES :

MNN = number of displacement for nonconsecutive addresses in memory.

MN1 = vector containing the address line headers supplied by the user (the first number in each the memory input file line)

MN2 = vector containing the values of the actual position in bliss memory . MN1 - MN2 gives the displacement used to the desired word in memory.

MACMEM = vector containing the memory data.

ROUTINES CALLED :

INTRAM = routine which makes 32 copies of the RAM portion of MACMEM.

MEM_SUB :

Substitutes the proper memory logic for the memory devices.

VARIABLES:

CHARACTER:

	T1	= Temporary variable.
	NO	= Output net.
	N1	= Input net.
	IIN	= Input string being parsed and substituted.
	OOUT	= Output string being parsed and substituted.
(main)	PIN	= Pin being replaced.
	PRINT	= Variable containing line to be printed.
	T	= Temporary variable.

CHARACTER ARRAY:

(main)	LIST1	= Unordered list.
(main)	IN	= Array of inputs to a particular component.
(main)	OUT	= Array of outputs to a particular component.
(main)	CLOCK	= List of input and output nets of ficticious clocks.

INTEGER ARRAY:

(main)	NETOUT	= Number of outputs to a particular component.
(main)	NETIN	= Number of inputs to a particular component.
	INP	= Number of inputs for each part.
	OUTP	= Number of outputs for each part.

FLTDET.B32

This is a bliss module that contains the routine DETECT.
The purpose of DETECT is to determine whether there were any differences between the output of the unfaulted machine and that of the 31 faulted machines. If so, it stores the results of the machine and the cycle in which the fault was detected.
FLTDET.B32 is created by CREATE_DET.

VARIABLES :

AMACH = contains values of the machines in which address faults were detected.

ACYCLE = contains values of the cycles in which address faults were detected.

PMACH = contains values of the machines in which pin faults were detected.

PCYCLE = contains values of the cycles in which pin faults were detected.

ER343A

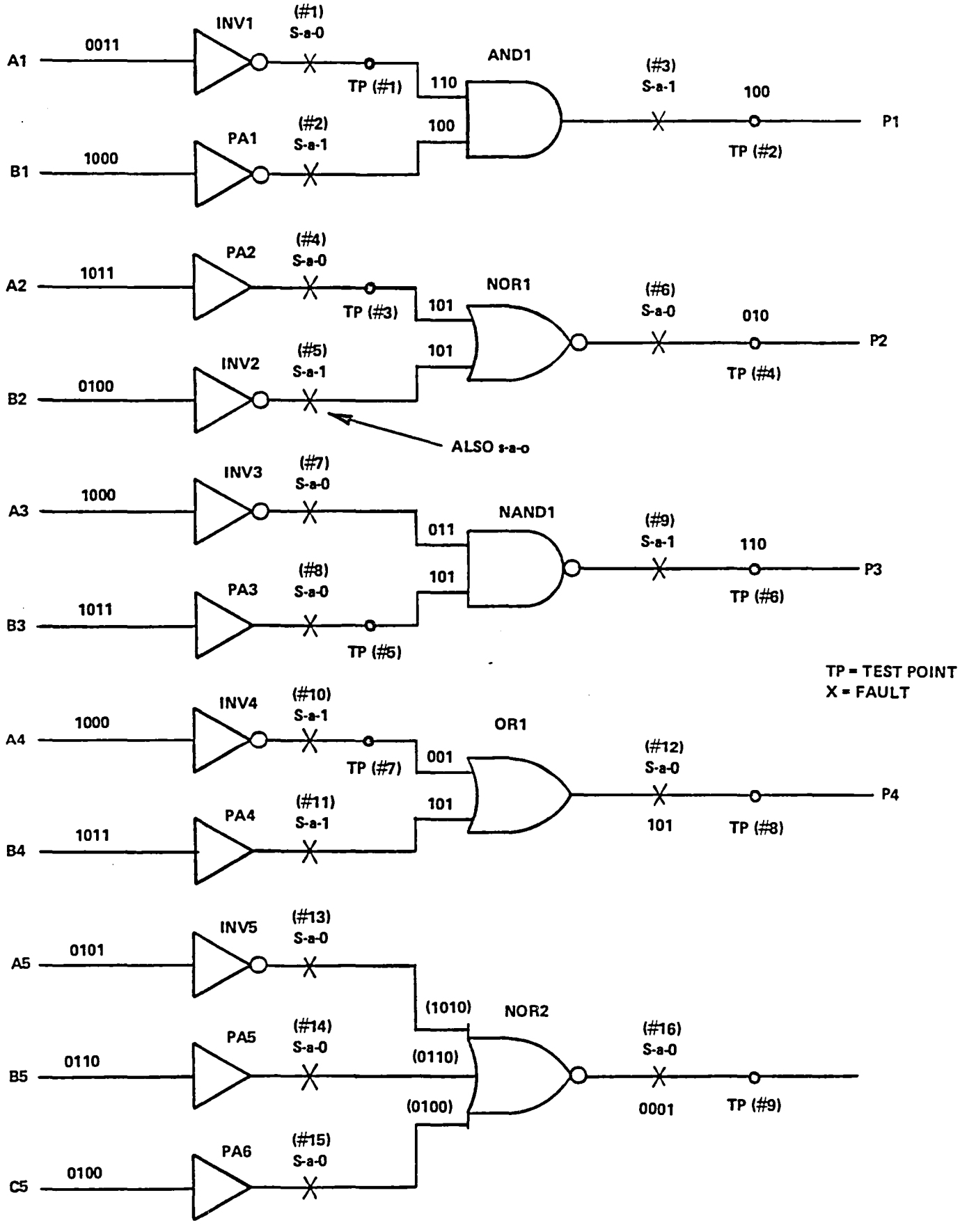


FIGURE 1 TEST CIRCUIT

FIGURE 2
ARITHMETIC LOGIC UNIT (ALU)
FUNCTIONAL EQUIVALENT LOGIC DIAGRAM

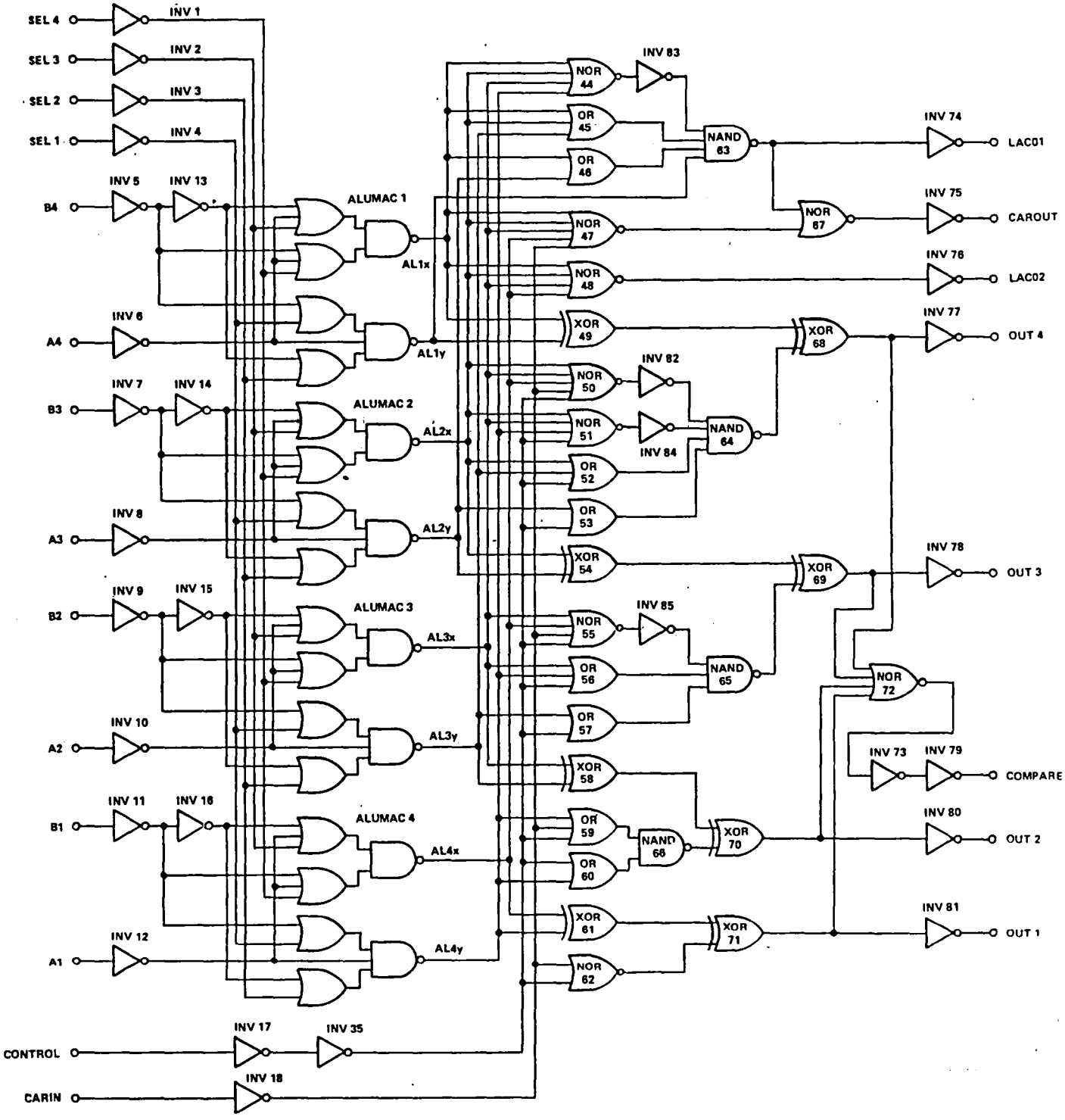
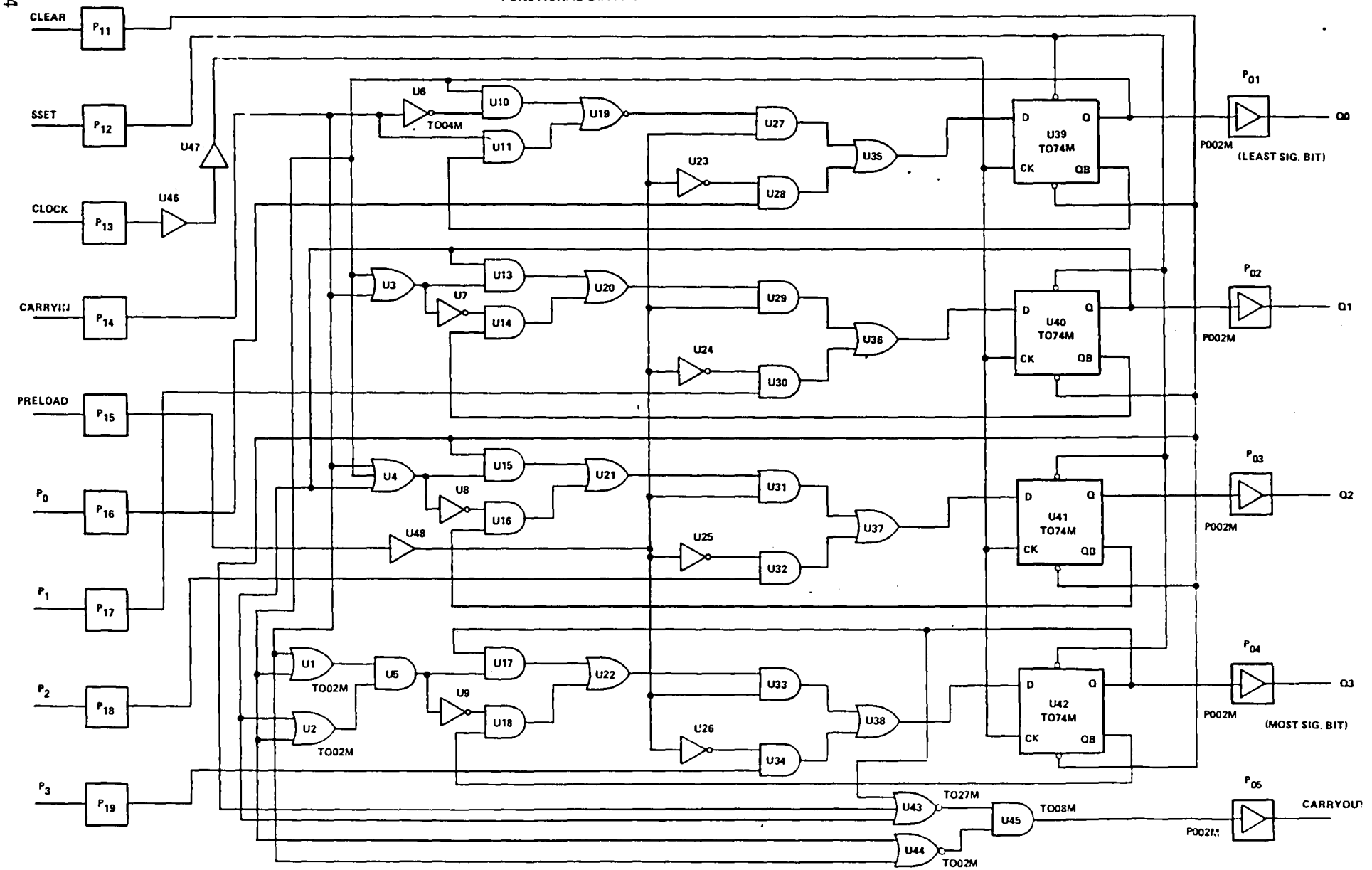


FIGURE 3
4-BIT DOWN COUNTER
FUNCTIONAL EQUIVALENT LOGIC DIAGRAM



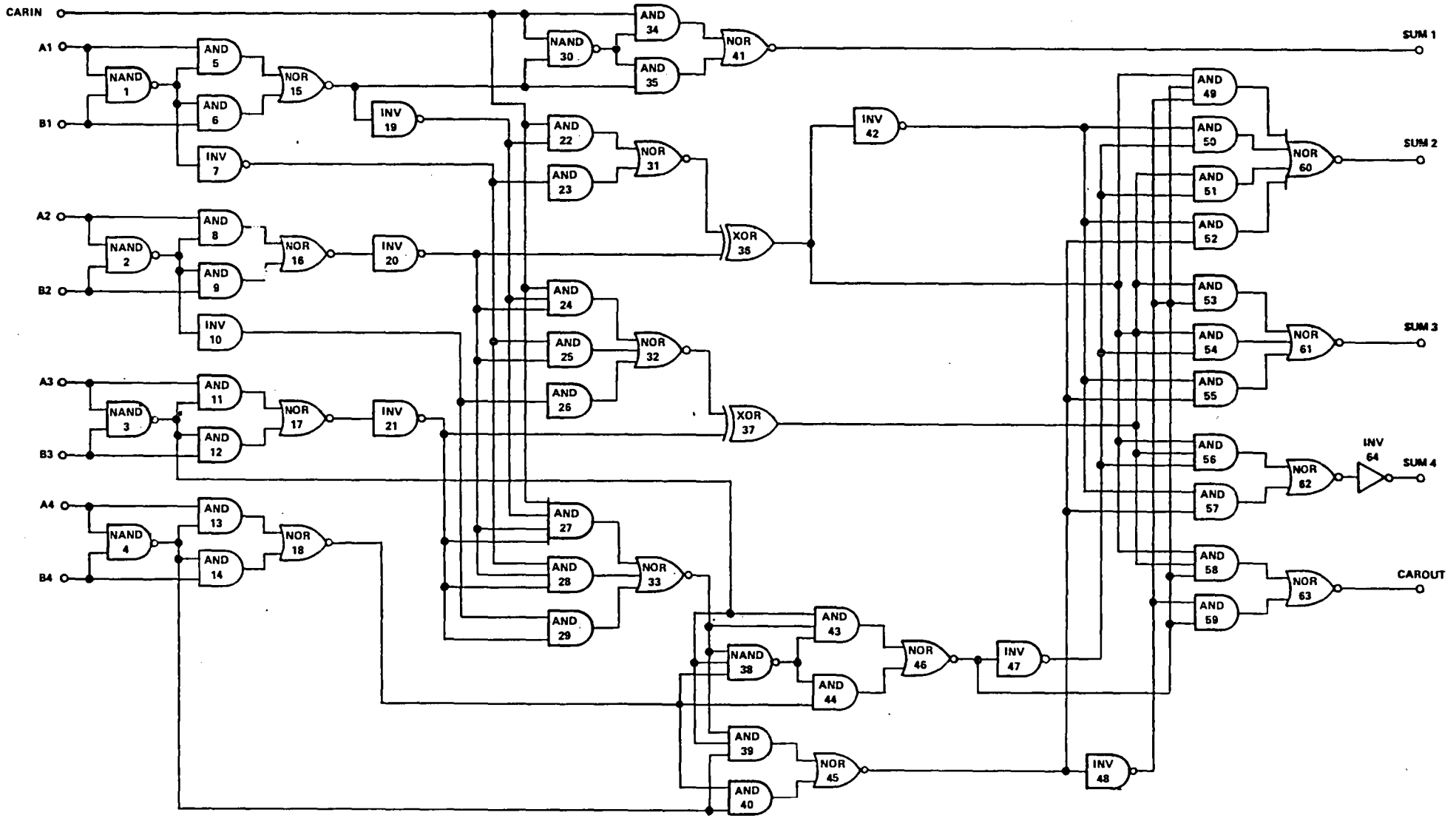


FIGURE 4
BCD ADDER
FUNCTIONAL EQUIVALENT LOGIC DIAGRAM

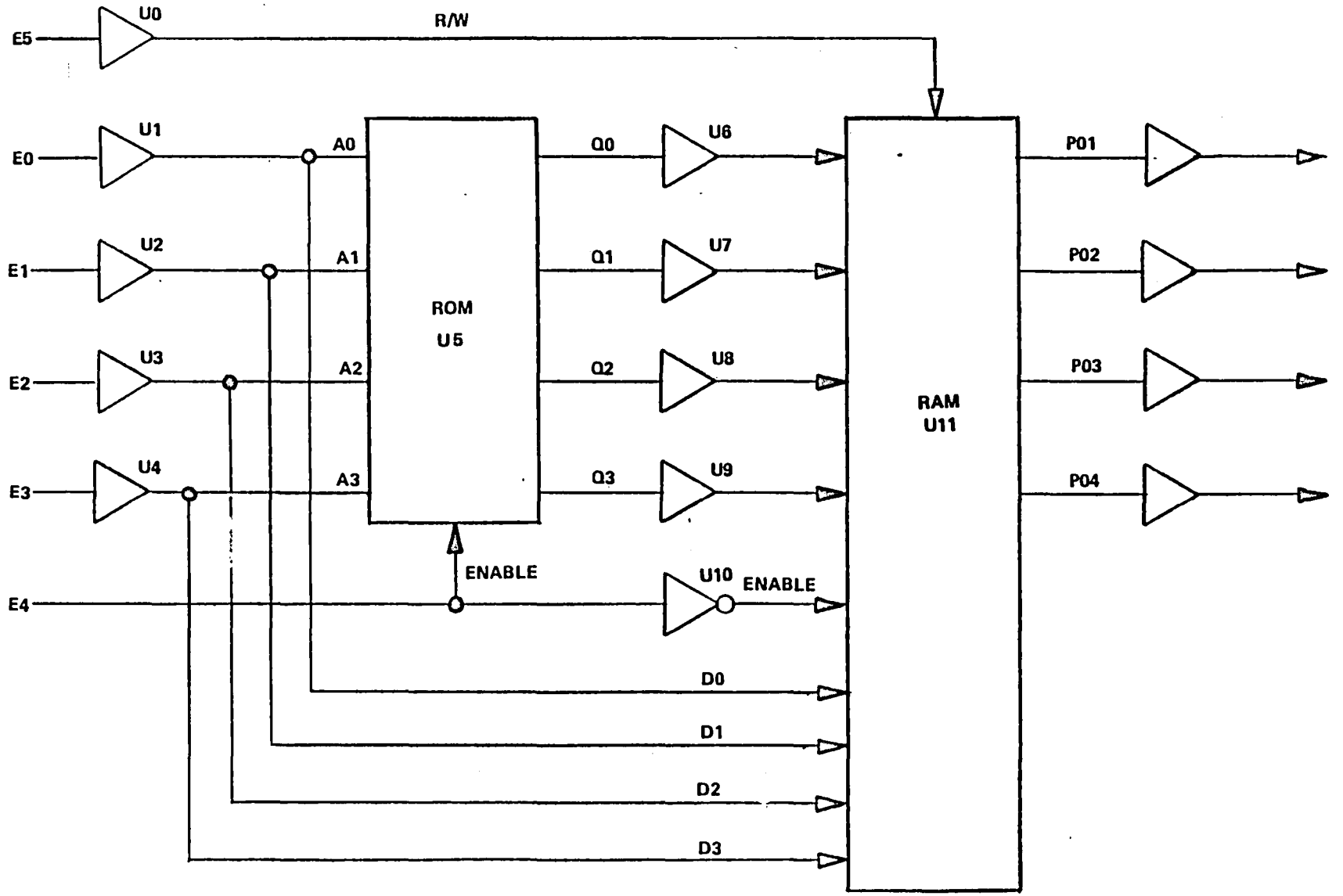


FIGURE 5 MEMORY CIRCUIT

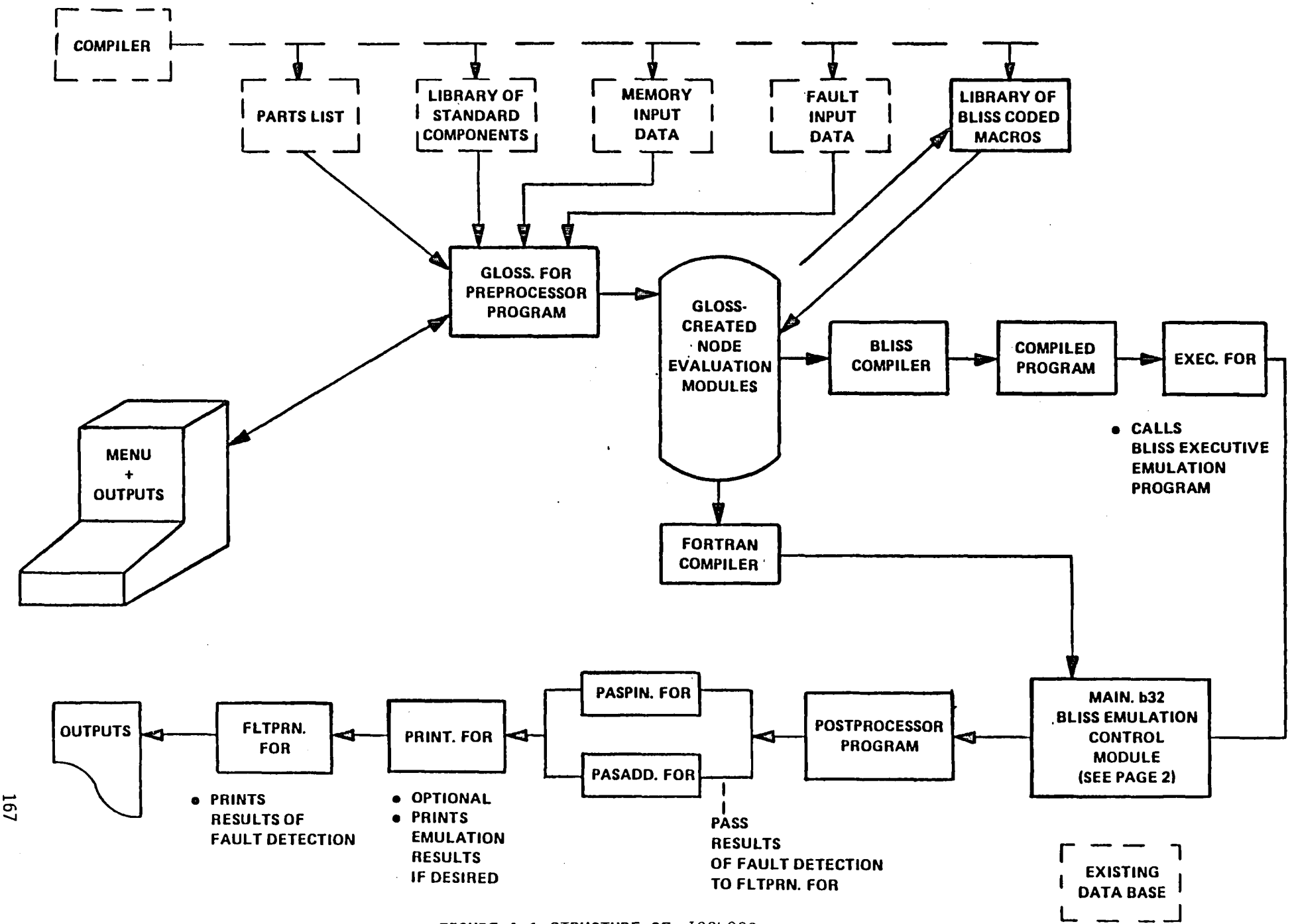
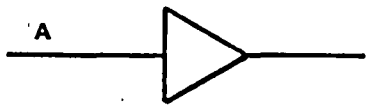


FIGURE A-1 STRUCTURE OF IGGLOSS



NON-FAULTED BUFFER

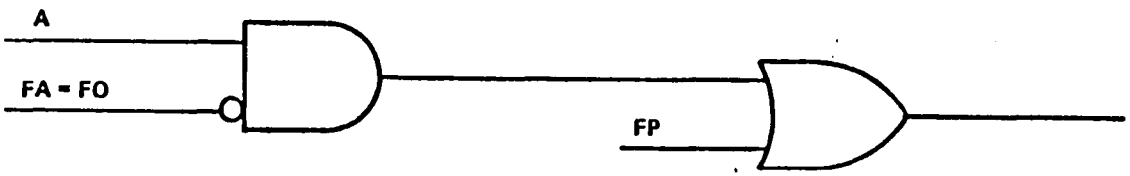
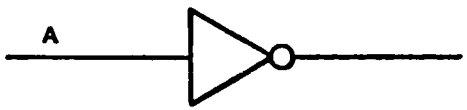


FIGURE B-1 FAULTED BUFFER



NON-FAULTED INVERTER

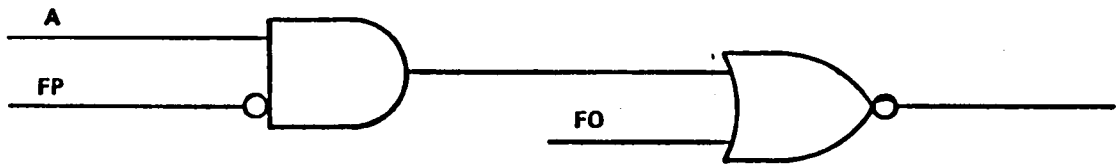


FIGURE B-2 FAULTED INVERTER

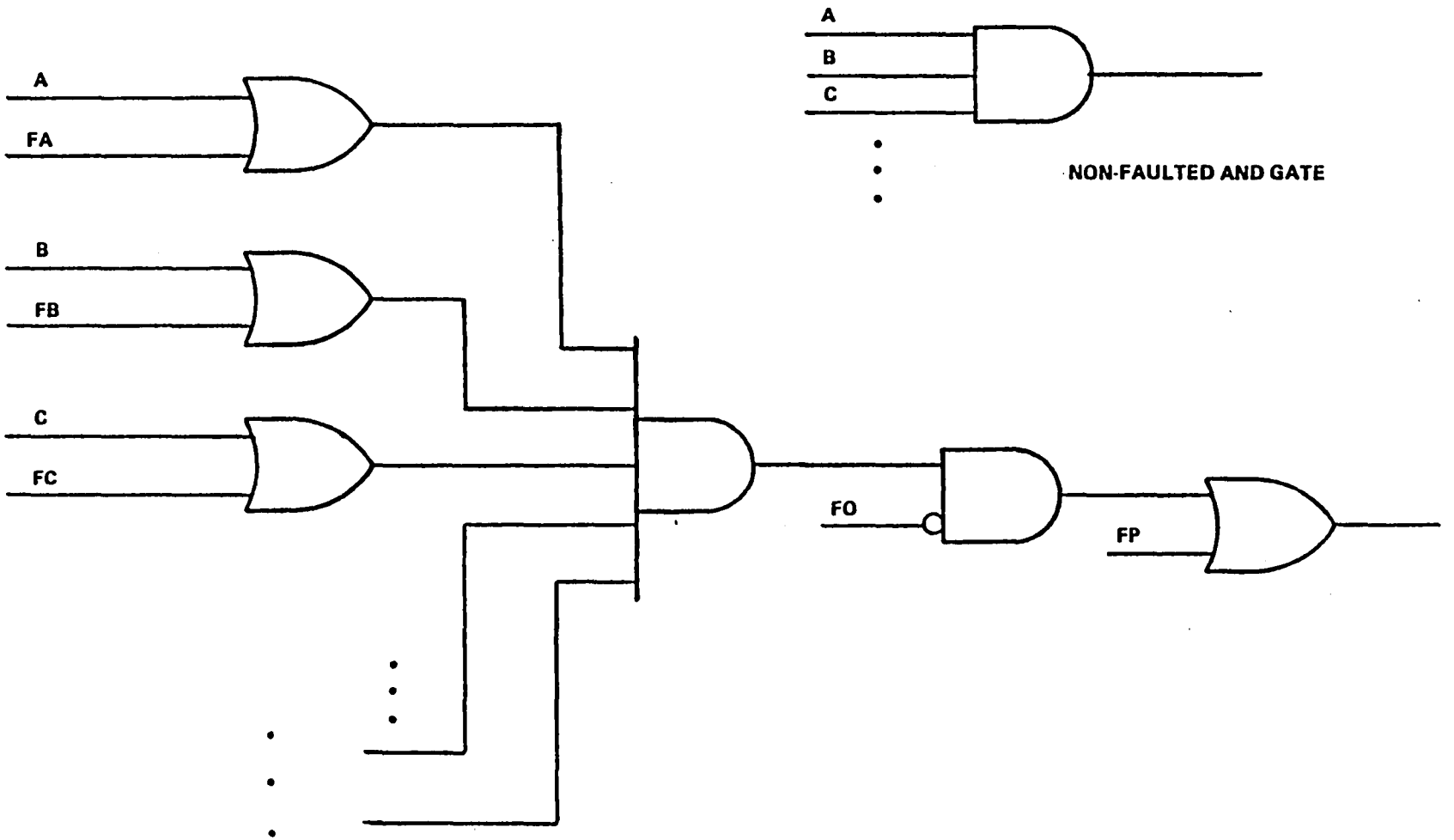


FIGURE B-3 FAULTED AND GATE

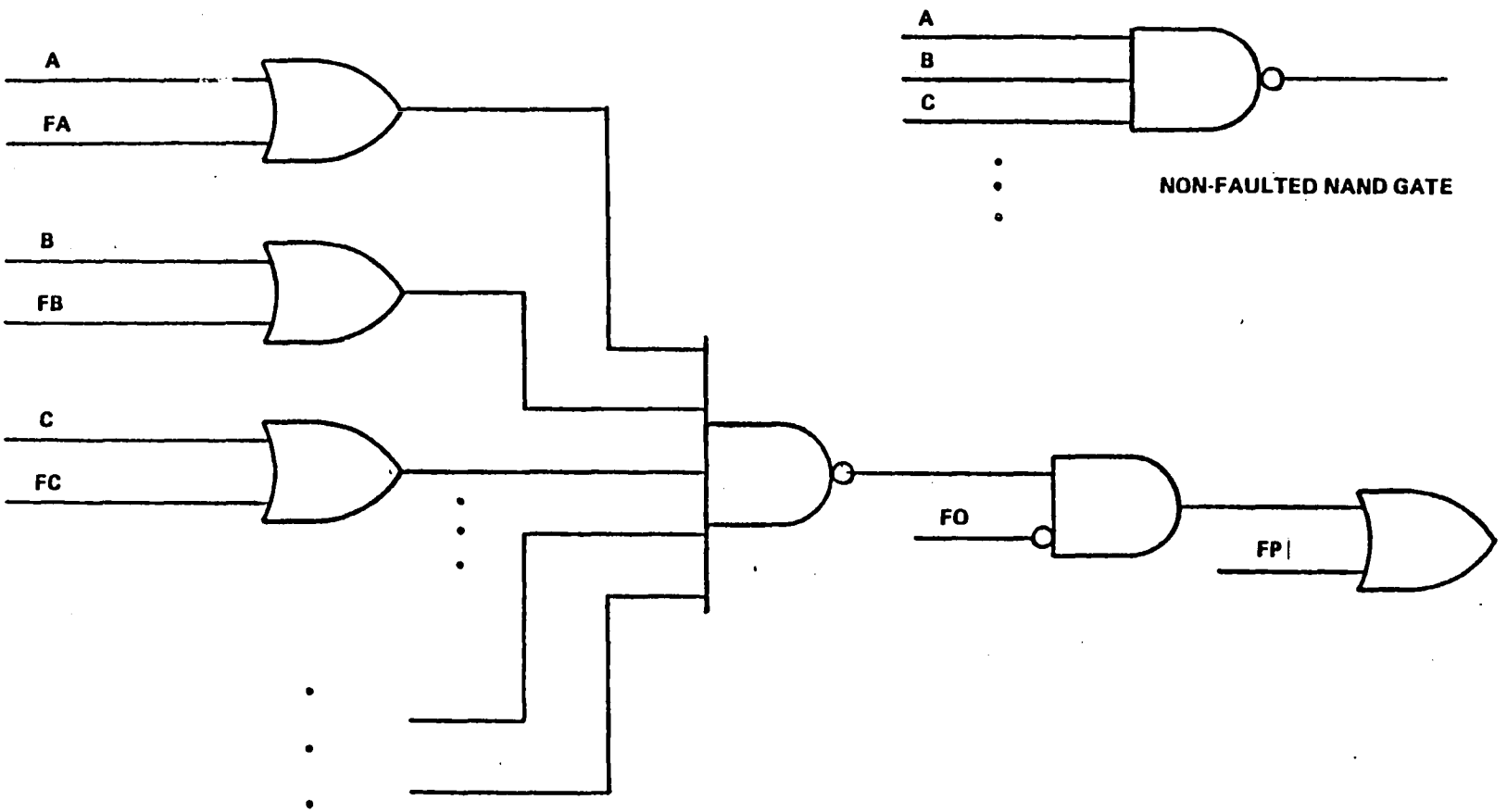


FIGURE B-4 FAULTED NAND GATE

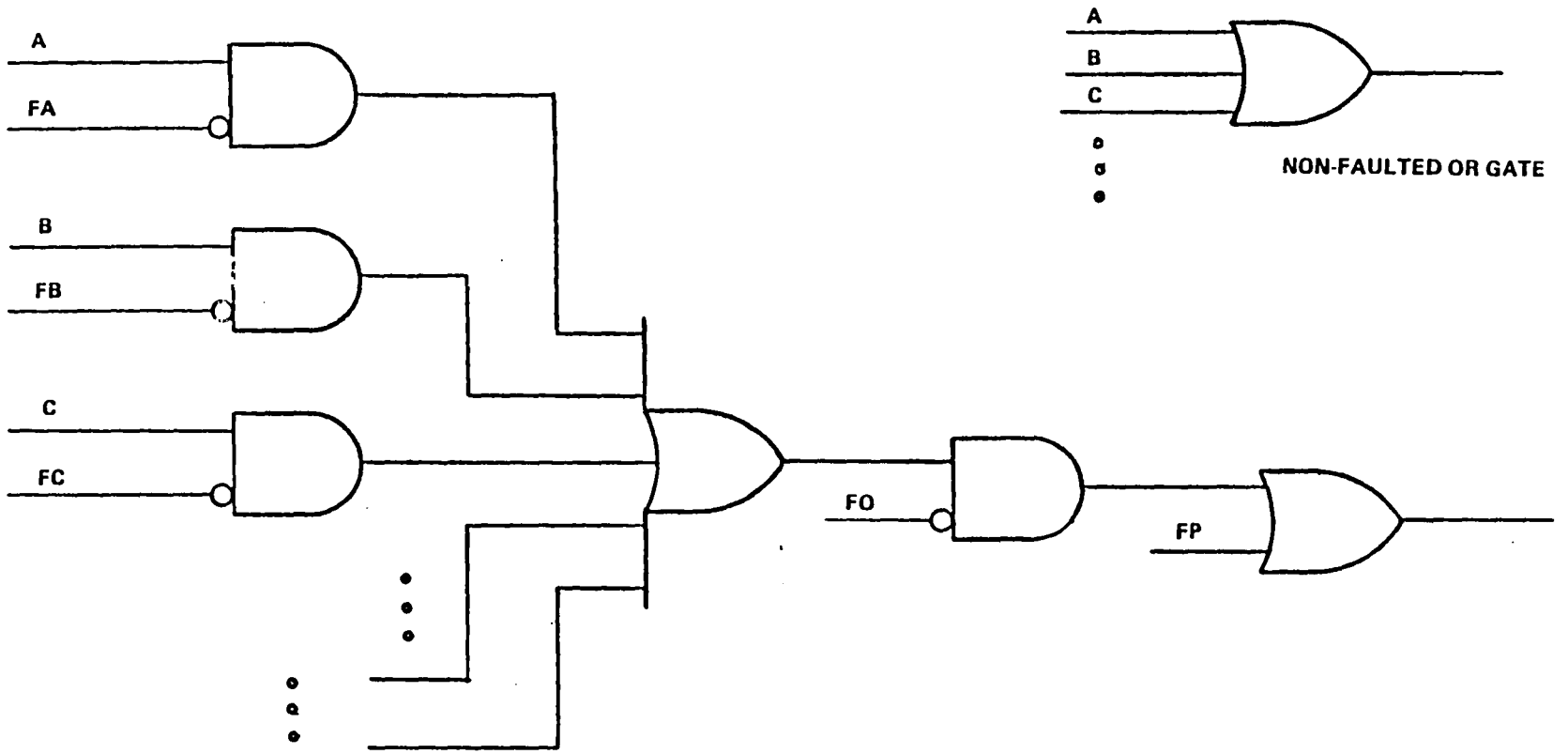


FIGURE B-5 FAULTED OR GATE

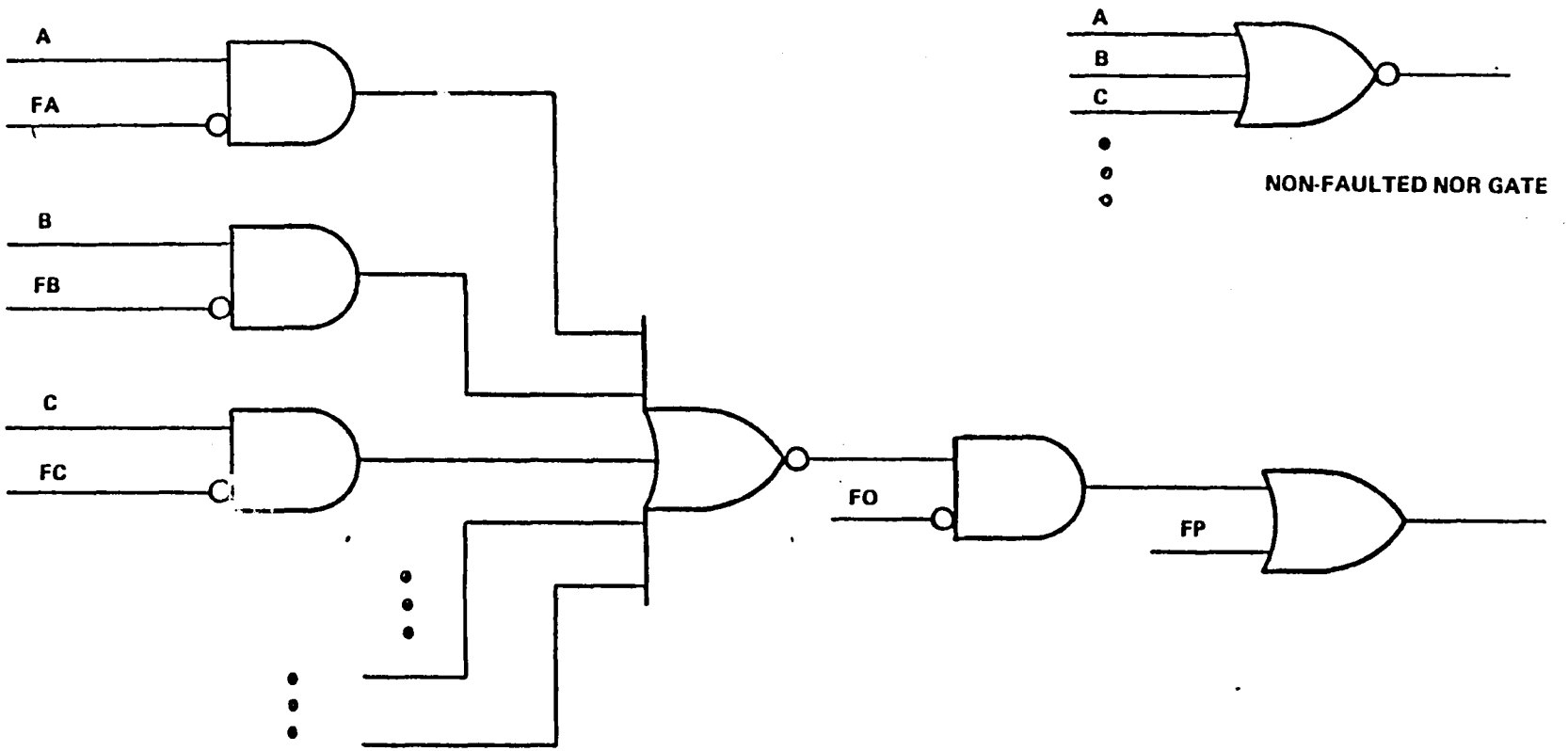


FIGURE B-6 FAULTED NOR GATE

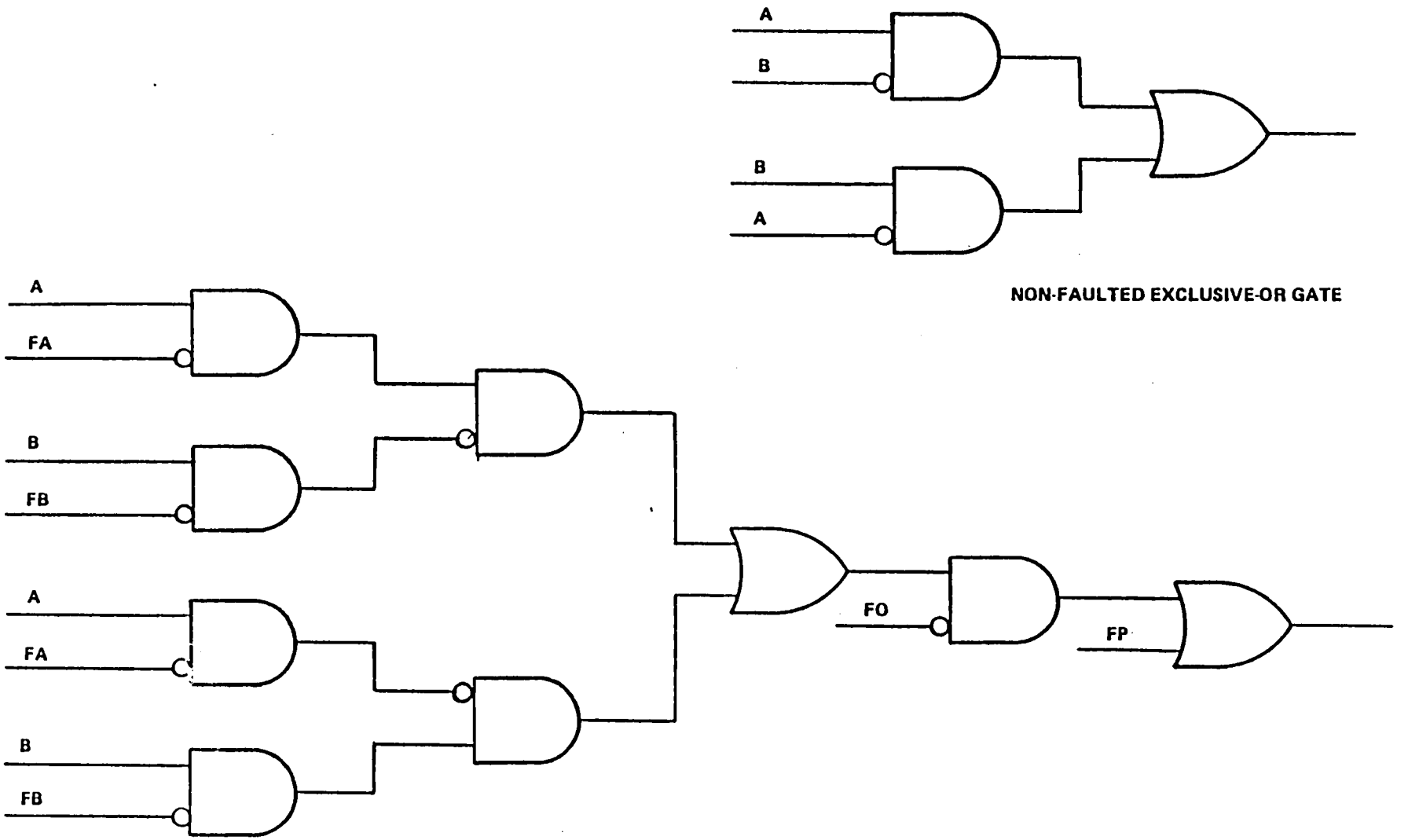


FIGURE B-7 FAULTED EXCLUSIVE OR GATE

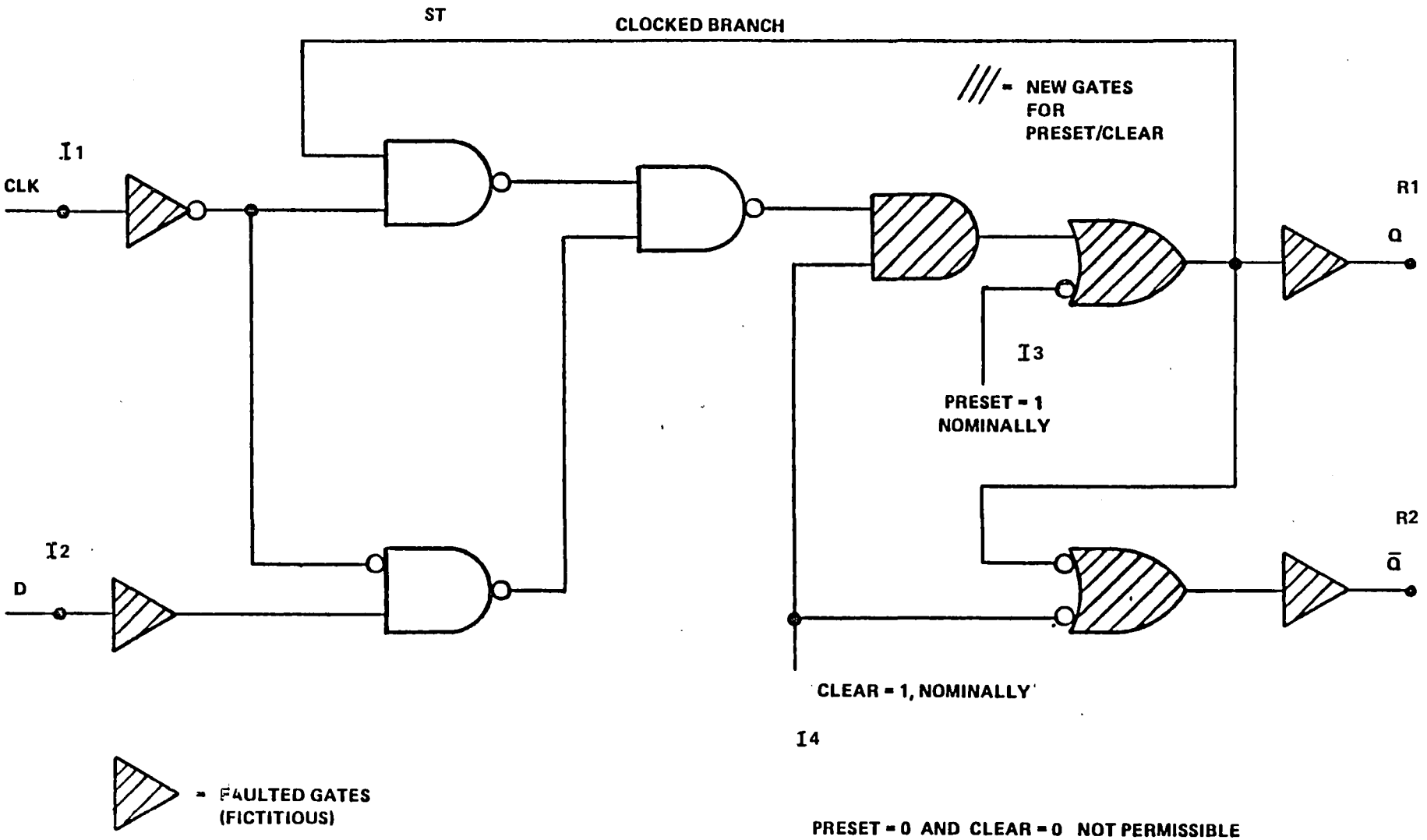


FIGURE B-8 D-FLIP FLOP WITH PRESET AND CLEAR

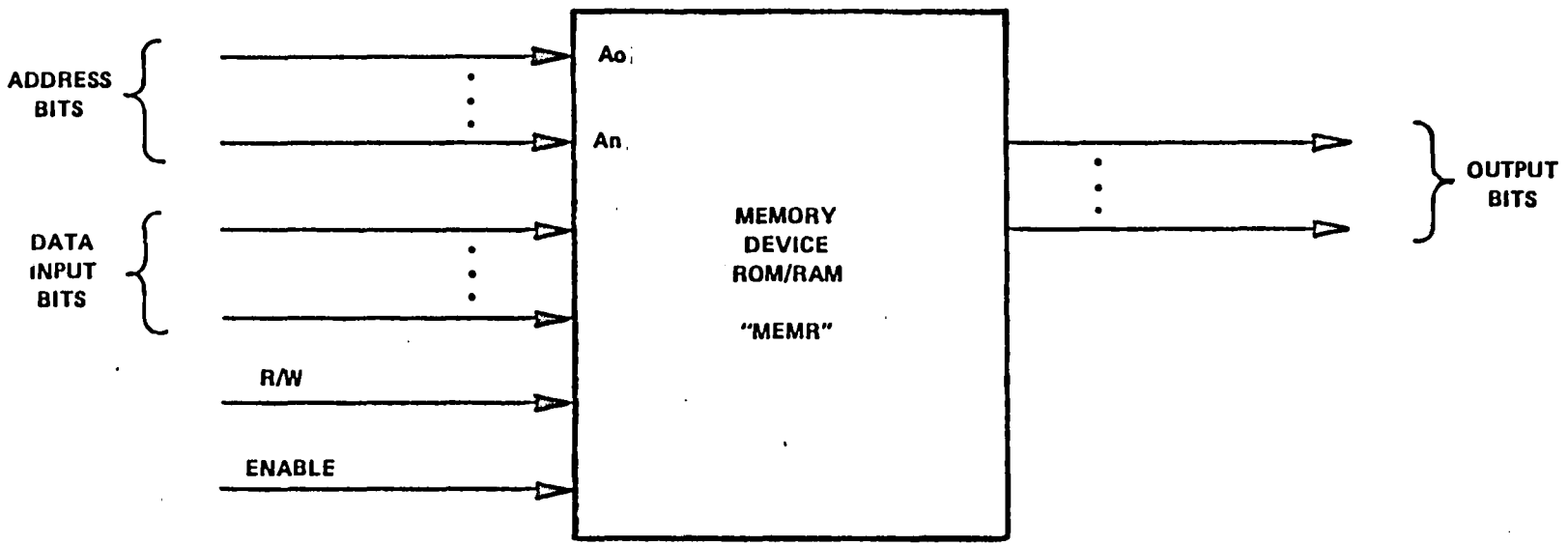


FIGURE B-9 STRUCTURE OF MEMORY DEVICE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-177939		

4. Title and Subtitle	5. Report Date
The Development of an Interim Generalized Gate Logic Software Simulator	December 1985
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
J. G. McGough and S. Nemeroff	
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Allied/Bendix Aerospace Flight Systems Division Teterboro, New Jersey	NAS1-15946
	13. Type of Report and Period Covered Contractor Report

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, DC 20546	505-34-13-30

15. Supplementary Notes

NASA Langley Senior Project Engineer: Salvatore J. Bavuso

16. Abstract

A proof-of-concept computer program called IGGLOSS (Interim Generalized Gate Logic Software Simulator) was developed and is discussed in this report. The simulator engine was designed to perform stochastic estimation of self-test coverage (fault-detection latency times) of digital computers or systems. A major attribute of the IGGLOSS is its high-speed simulation: 9.5×10^6 gates/cpu sec. for nonfaulted circuits and 4.4×10^6 gates/cpu sec. for faulted circuits on a VAX 11/780 host computer. See NASA Contractor Report 172159, April 1983 for design principles.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Emulation Gate-level Fault detection Fault latency Self-test Comparison-monitoring Coverage	Unclassified - Unlimited Subject Category 59

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	178	

N-305 For sale by the National Technical Information Service, Springfield, Virginia 22161

THIS PAGE INTENTIONALLY LEFT BLANK

End of Document