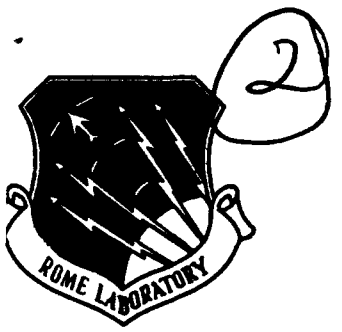
RL-TR-92-199
Final Technical Report
July 1992

AD-A257 712



SCAN-BASED SWITCHING TESTS

DTIC
ELECTE
NOV 0 6 1992
S A D

Harris Corporation

Larry D. Bashaw, Ted H. Vriezen

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

411661
92-28913
6985

92 11 04 067

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-92-199 has been reviewed and is approved for publication.

APPROVED: Mark Gorniak

MARK GORNIAK
Project Engineer

FOR THE COMMANDER: John J. Bart

JOHN J. BART, Chief Scientist
Reliability Sciences
Electromagnetics & Reliability Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(ERDA) Griffiss AFB, NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1992	Final -----

4. TITLE AND SUBTITLE
SCAN-BASED SWITCHING TESTS

5. FUNDING NUMBERS
C - F30602-90-D-0006,
Task 4
PE - 62702F
PR - 2338
TA - QD
WU - 05

6. AUTHOR(S)
Larry D. Bashaw, Ted H. Vriezen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Harris Corporation
Government Information Systems Division
P O Box 98000
Melbourne FL 32902

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (ERDA)
525 Brooks Rd
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-92-199

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Mark Gorniak/ERDA/(315) 330-2047

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report describes an algorithm for generating scan-based switching tests. The only design consideration required by this algorithm, besides the scan design is the capability to execute two functional clocks with a desired interval between logic scans. The algorithm generates a test vector which is scanned in to cause the source register of the delay path to toggle on the first clock, and captures a transition at the output register of the delay path on the second clock. The algorithm also identifies paths that are not testable, and identifies the points of conflict.

14. SUBJECT TERMS
Automated Timing Test Generation
Scan-based Switching Tests

15. NUMBER OF PAGES
64

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

CONTENTS

Figures

Tables

Attachments

EVALUATION

This effort successfully developed an algorithm for generating scan-based switching tests. The algorithm identifies timing paths from incomplete lists of nodes and automatically marks them for timing test generation. Tests are attempted for both rising and falling edge transitions at the source bistables of the path. If a test cannot be generated, the logic conflict which prohibits test generation is identified.

The only design considerations for this algorithm were the scan-design and the ability to execute two functional clocks with desired interval between logic scans. However, it became apparent from the test case that circuits which use many state-machines and counters, require multiple path control bistables to be simultaneously switched in order to achieve path sensitization. As a result, further work is needed to address this issue.

MARK G. GORNIAK
Center Program Manager

DTIC QUALITY INSPECTED

Accesion For
NTIS CRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification .....
By .....
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

I. Introduction

This report describes an algorithm which provides automated timing test generation for selected timing paths in VLSI/VHSIC-scale ASIC designs. Timing tests have many uses besides providing delay measurements of logic paths in ASICs (See Table 1). However, these measurements are usually difficult to make because the signals required to control the measurement of the path delay are not directly addressable from the chip Input/Output pins. Often, multiple levels of bistables exist between the logic path to be measured and the I/O pins.

The manual development of functional tests that can be applied to the I/O pins of an ASIC, and can be guaranteed to test a desired path, is difficult and time consuming. This is true even if the tests can be applied via scan. Automatically generated tests can reduce test generation costs and schedule risk while producing tests for a larger number of paths. This program researches the utility of a delay test generation algorithm which takes advantage of scan design, and does not require additional logic design considerations.

Scan is used to load the test vectors generated by this algorithm. The test is performed by executing two clocks which define a specific time interval. The results are then scanned out for examination to determine if the test passed (the delay path was shorter than the clock interval), or failed (the delay path was longer than the clock interval). With repeated executions of a given test vector with differing clock intervals, the precise timing of the delay path can be determined.

The switching test algorithm was implemented in Ada for demonstration on the Test-bus Interface Unit (TIU) design, an ASIC design used in the Radiation Hard 32 bit Processor (RH32) Program. [F30602-88-C0060]

An example circuit was generated for the purpose of describing the algorithm. This sample circuit (Figure 6 - 'Timing Path Example') is not intended to perform any

particular function but is intended only for descriptive reference. The bistables and gates in the example would normally drive into other logic cones. However, to simplify the schematic, these other logic cones are not shown, nor are scan paths shown.

Test generation algorithm results for 5 delay paths from the TIU test case are presented in.Figure 5 and Figures 7 through 10.

Delay Test	Application
In a VLSI/VHSIC chip, measure delay paths which contain particular combinations of gate delay, media delay, and gate loading	Verify SPICE or other Timing Analysis program calibration.
Measure the N longest delay paths in a fabricated chip.	Verify that required design margins are being met.
Measure selected paths in a design under various combinations of temperature, voltage or radiation	Verify that environmental performance requirements are being met
Use delay path tests in IC fabrication test screen	Part of a Total Quality Management plan.
At the Line Replaceable Module or System level, measure the timing of selected paths in a design.	Verify performance margins over time or after repair.

Table 1 Applications for delay path tests are not restricted to the detection of individual gate or delay path timing faults.

Glossary Of Terms

ASIC - Application Specific Integrated Circuit.

'B' logic state - A state sensitive rising logic transition.

Bistable - An edge sensitive two state storage element. (Elements F1 through F15 in Figure 6). Also referred to as a flip-flop.

'D' logic state - A state-sensitive falling logic transition.

First Level Bistable - Any bistable which drives into the logic cone enclosing a defined timing path. (Bistables F7 through F14 in Figure 6).

Implicate a gate - Simulate the loads of a gate, which has had its output state established, until a quiescent (static) logic state is reached.

Justify a gate - Establish the input states on a gate that will produce a required output state on that gate.

Second Level Bistables - Any bistable which drives the logic cone formed by extending the logic cone enclosing the defined timing path to the next level of bistables. (Bistables F1 through F6 and F14 and F12 in Figure 6). Note: A first level bistable, source bistable, or target bistable may also be a second level bistable.

Sensitize - Establish input states on a logic gate such that a logic transition on a selected input (the sensitized input) will cause an output transition on the gate.

Source Bistable - The bistable at the beginning of the the delay path to be tested, which will be switched by a test to send a logic transition down the path. This is a special case of the first level bistable. (Bistable F10, for the timing path depicted by the bold line in Figure 6).

Target Bistable - The bistable at the load end of the delay path being tested, into which the logic transition is clocked on the second of two clocks. The target bistable receives the pre-transition logic state if the path time is longer than the clock period, and receives the post transition logic state if the path time is shorter than the clock period. (Bistable F15 in Figure 6). Note: the target bistable is also a first level bistable when it drives into the timing path logic cone.

VLSI/VHSIC - Very large scale integrated/Very high speed integrated circuit.

'X' logic state - A logic state that has not been justified to a known state.

II. ALGORITHM OVERVIEW

Delay Test Algorithm Characteristics

The delay test generation algorithm is designed to produce timing tests for logic paths in ASIC designs. Tests are produced for specified logic paths between clocked registers. Specific timing information of the logic being tested is not required by the algorithm to generate the tests. This imposes limitations on the design architectures that are testable (discussed in Section IV). The algorithm has the following characteristics.

- Applicable to designs which utilize scan based Design-For-Test using edge sensitive synchronous clocking.
- The only design considerations required are scan and the capability to cause exactly two functional clocks to be executable between logic scans.
- The algorithm assures that only the timing path of interest is measured and that no alternate path is enabled.
- When an alternate path is unavoidably enabled, no test is produced.
- The algorithm is also applicable to designs which can be scanned at functional clock speed, with clock control switching between the last scan clock and a functional clock.

The algorithm is intended for the generation of timing tests for defined individual logic paths, and not for the generation of a comprehensive set of timing tests for a design. Timing tests produced by the switching test algorithm are applicable to an ASIC at the chip, board, and Line Replaceable Module (LRM) level by a tester. The tests can be applied at the cabinet and system level by an IEEE 1149.1 Test Access Port and Boundary Scan interface or equivalent. Regardless of the test application logistics, the tests are applied by scanning in a single test pattern, applying two clocks at a desired interval, and scanning out the test results for examination. If the interval between the two functional clocks is equal to or greater than the tested path delay, then the expected test result state is clocked into the scan path for scan out and examination. If the interval between the two functional clocks is less than the delay of the timing path, then the state opposite from the expected state is clocked into the target bistable for

scan out and examination. Only one test, of the two possible signal transitions, is intended to be executed at a time. However, if tests are generated for completely disconnected logic areas in an ASIC, then two or more of these tests could be executed at the same time.

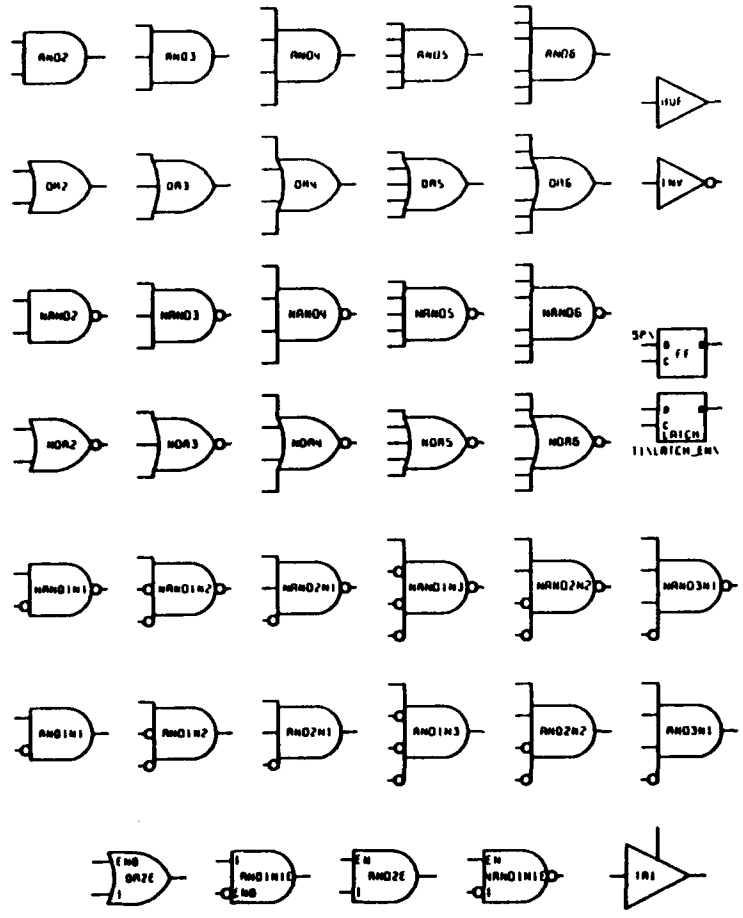
Switching Test Logic Model

A gate-level logic model of the logic to be tested is generated from a netlist by a logic model generation program. The model is composed of primitive gates and macro modules which expand into primitive gates (see Figure 1 - 'Simulation Primitive Cells and Macros'). The primitive gates include a buffer, an inverter, AND, NAND, OR, and NOR gates with two to six inputs. There is also a set of AND and NAND primitives, with two to four inputs, which contain input inversions to model true and complement signals, without requiring inverters. Additionally, there is a tri-state selector gate and a set of tri-state enable gates. The macros consist of a two input multiplexer, a latch and a flip-flop which have asychronous sets and resets, an exclusive OR, and an exclusive NOR. Also, there are macros for modeling ASIC bidirectional and tristate input/output pins.

The Switching Test Algorithm accepts the logic model description of interconnected primitive elements. The model description consists of three machine readable data files (see Figure 2 for the record layout of these files) which define the gate code for each gate in the model and the interconnection between the gates. The first file is the gate identification file which contains a header record for each primitive logic element. The header record contains an integer gate code (Figure 3 - 'Internal Gate Codes'), the number of true inputs on the gate, five unused boolean fields, (which were assigned early on, are not needed, but were not removed), the number of drives into the gate, number of gates which are loads of this gate, an index into an array of the drive index file, and an index into an array of the load index file. The second file is the drive index file which identifies the gates which provide the signal source for each input signal of each primitive logic element. The third file is a load index file which defines the gates being driven by each primitive logic element.

The model description also includes an Instance Index File and an Instance and Net Pathname file (Figure 4 -'Instance and Net Pathname Example'). The Instance and Pathname file defines the net names and instance names of each primitive logic element. The Instance Index file relates each gate in the simulation model to its instance and path name. These data files are generated by a Logic Model Partition Generator which is run prior to running the Switching Test Generator.

After the logic model is generated, a scan check program is run to verify the proper interconnection of the scan control logic and to automatically determine the scan chain bit order.



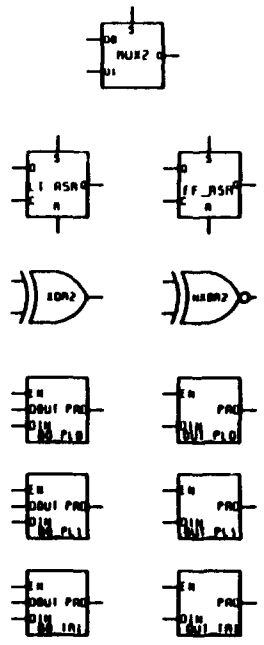


Figure 1

Simulation Primitive Cells and Macros

```
type t_head_record is
     record
          gate_code          :   integer;
          true_inputs        :   integer  := 0;
          logic_trace        :   boolean;
          is_light           :   boolean:
          model_flts         :   boolean;
          reconv_back        :   boolean;
          no_d_push:         :   boolean;
          nmbr_of_drivs      :   integer;
          nmbr_of_loads      :   integer;
          driv_indx          :   integer;
          load_indx          :   integer;
     end record;

type t_driv_record is
     record
          indx   :   integer;
     end record;

type t_load_record is
     record
          indx   :   integer;
     end record;
```

Figure 2
Header, Drive, and Load Records

```
and_g          :   constant     :=  1;
nand_g         :   constant     :=  2;
in_g           :   constant     :=  3;
out_g          :   constant     :=  4;
ff             :   constant     :=  5;
wbus_g         :   constant     :=  8;
enab_g         :   constant     :=  9;
```

Figure 3
Internal Gate Codes

Model Index	Network Index	Gate Type Code	Network Index	Gate Type Code / Gate Type Name / Instance Name, Net Name
3993	3810	503	3810,	503,NOR3,/SCI_A/CNTBLK_A/U124/I$471,/SCI_A/N$457;
3994	5263	450	3811,	503,NOR3,/SCI_A/CNTBLK_A/U75/I$471,/SCI_A/CNTBLK_A/N$480;
3995	5266	302	3812,	503,NOR3,/SCI_A/CNTBLK_A/U113/I$471,/SCI_A/CNTBLK_A/N$259;
3996	5859	301	3813,	503,NOR3,/SCI_A/CNTBLK_A/U91/I$471,/SCI_A/CNTBLK_A/N$255;
3997	314	301	3814,	503,NOR3,/SCI_A/CNTBLK_A/U106/I$471,/SCI_A/CNTBLK_A/N$251;
3998	313	301	3815,	503,NOR3,/SCI_A/XCNTBLK_A/U53/I$471,/SCI_A/N$239;
3999	4528	302	3816,	503,NOR3,/SCI_A/XCNTBLK_A/U51/I$471,/SCI_A/N$240;
4000	858	301	3817,	503,NOR3,/SCI_A/XCNTBLK_A/U40/I$471,/SCI_A/N$238;
4001	2645	503	3818,	503,NOR3,/SCI_A/XCNTBLK_A/U52/I$471,/SCI_A/XCNTBLK_A/N$14;
4002	876	301	3819,	503,NOR3,/SCI_A/BUSSM_A/U368/I$471,/SCI_A/N$118;
4003	888	301	3820,	503,NOR3,/SCI_A/BUSSM_A/U373/I$471,/SCI_A/BUSSM_A/N$360;
4004	4534	302	3821,	503,NOR3,/SCI_A/MASTERSM_A/U218/I$471,/SCI_A/N$93;
4005	611	301	3822,	503,NOR3,/SCI_A/MASTERSM_A/U261/I$471,/SCI_A/MASTERSM_A/N$165;
4006	5083	202	3823,	503,NOR3,/TMI_A/PARINCHK_A/U22/I$471,/TMI_A/PARINCHK_A/N$13;
4007	708	301	3824,	503,NOR3,/TMI_A/DIOSTATUS_A/U65/I$471,/TMI_A/N$970;
4008	2652	301	3825,	503,NOR3,/TMI_A/DIOSTATUS_A/U43/I$471,/TMI_A/N$532;
4009	3971	303	3826,	503,NOR3,/TMI_A/DIOSTATUS_A/U64/I$471,/TMI_A/N$976;
4010	5626	302	3827,	503,NOR3,/TMI_A/DIOSTATUS_A/U51/I$471,/TMI_A/DIOSTATUS_A/N$102;
4011	5174	549	3828,	503,NOR3,/TMI_A/MAINSMSLV_A/U459/I$471,/TMI_A/MAINSMSLV_A/N$373;
4012	3847	503	3829,	503,NOR3,/TMI_A/MAINSMSLV_A/U493/I$471,/TMI_A/MAINSMSLV_A/N$369;
4013	618	549	3830,	503,NOR3,/TMI_A/INTRPTSM_A/U66/I$471,/TMI_A/INTRPTSM_A/N$27;
4014	3959	303	3831,	503,NOR3,/TMI_A/INTRPTSM_A/U70/I$471,/TMI_A/INTRPTSM_A/N$31;
4015	5531	450	3832,	503,NOR3,/TMI_A/TMSTAT_A/U59/I$471,/TMI_A/TMSTAT_A/N$189;
4016	654	549	3833,	503,NOR3,/TMI_A/TMADDR_A/U70/I$471,/TMI_A/TMADDR_A/N$141;
4017	730	301	3834,	503,NOR3,/TMI_A/ACKDATLDSM_A/U66/I$471,/TMI_A/ACKDATLDSM_A/N$24;
4018	621	301	3835,	503,NOR3,/TMI_A/IOCNFG_A/U36/I$471,/TMI_A/IOCNFG_A/N$79;
4019	813	301	3836,	503,NOR3,/TMI_A/IOCNFG_A/U49/I$471,/TMI_A/IOCNFG_A/N$75;
4020	808	301	3837,	503,NOR3,/TMI_A/TMBUSSM_A/U314/I$471,/TMI_A/N$83;
4021	2091	201	3838,	503,NOR3,/TMI_A/TMBUSSM_A/U327/I$471,/TMI_A/N$81;
4022	811	301	3839,	503,NOR3,/TMI_A/TMBUSSM_A/U305/I$471,/TMI_A/TMBUSSM_A/N$19;
4023	575	301	3840,	503,NOR3,/TMI_A/TMBUSSMSLV_A/U514/I$471,/TMI_A/N$162;
4024	814	301	3841,	503,NOR3,/TMI_A/TMBUSSMSLV_A/U492/I$471,/TMI_A/TMBUSSMSLV_A/N$416;
4025	5134	549	3842,	503,NOR3,/TMI_A/TMBUSSMSLV_A/U472/I$471,/TMI_A/TMBUSSMSLV_A/N$424;
4026	578	549	3843,	503,NOR3,/TMI_A/TMBUSSMSLV_A/U450/I$471,/TMI_A/TMBUSSMSLV_A/N$408;
4027	1103	302	3844,	503,NOR3,/TMI_A/TMBUSSMSLV_A/U505/I$471,/TMI_A/TMBUSSMSLV_A/N$396;

Figure 4

Instance and Net Pathname Example

Switching Test Timing Path Identification

The Switching Test Algorithm accepts an ASCII file list of path definitions composed of logic names of elements included in the path(s) to be tested. This File may be generated manually or by software that allows a user to define timing paths of interest, such as a static timing analysis program (Figure 5 - 'Static Timing Analysis Example'). The names of the elements in the defined path are used for timing path definition. The content and format of the remaining information is not of interest to the switching test algorithm. Each defined timing path to be tested has a single bistable primary input (source) to the path, and a single bistable (target) primary output from the path. Each path definition starts with the net name that defines the primary input bistable of the path. Each succeeding net name defines succeeding elements in the path and the last net name defines the primary output bistable of the path. Each line of the ASCII path definition file contains a path number, to specify a particular path, followed by the net name of an element in the path.

Figure 7, the 'Timing Path Identification list', provides a partial list of timing path definitions for a design. The first entry in the list 1, N26 defines the path source bistable (N26) for path 1 in Figure 6. The second name in the list (N32) is the name of the first named gate in that timing path. The remaining gates follow in similar fashion until N37, which identifies the target bistable into which the timing path transition will be clocked. The next name in the list identifies the source bistable (/TMI-A/N$802) for the next selected timing path (#4246 in Figure 7).

Simulation Model Generation

A simulation model is generated for the design containing the timing paths to be tested, using the logic model description file as input. When the simulation model is loaded for test generation, the model gate codes are converted to an internal form. The internal form allows the timing test generator to simulate all of the AND, NAND, OR, and NOR gate types as either AND or NAND gates, with varying input pin widths and input inversions to account for the OR/NOR functions. Thus, only AND gates, NAND gates, Bistable elements, and Input/Output pin elements are used in the model (Figure 3 'Internal Gate Codes').

When the model is generated and loaded for simulation, all of the gates in the design set are initialized to the unknown (X) state. The entire model is then simulated so that the gates with inputs which are tied to a known logic state are forced to their correct state. Signals are implicated from any gate states which change to a known state until the simulation model reaches a quiescent state.

>>>> PATH_REPORT Version 91.1.04 DAMSEL_TA version 91.1.04 <<<< Page- 4250

Design: ./design.erel_$3/vds_db Date: 12/09/91 17:26:21
Technology File: /user/hw_vds/sys/tech_tables/hr2210

Path Number: 10304
Path Margin: -6.95 Clock Cycle: 400.00 Path Delay: 406.95
Analysis Type: Long Path Skew Analysis: None Trace Algorithm:
Paths Reported: All Paths Paths Reporting Direction: Forward
Supply: 5.00 Junction Temp: 25.00
Timeup File: ./external.timeup
Delays: Typical Spice Models: Typical Case Radiation: POST_RAD

Source Pathname	Net Pathname	Source Type	Input Name	Output Name	Total Rise	Total Fall	Unate	Cum One	Cum Two
/SCI_A/BUSSM_A/CURR_STATE_REG[2]	/SCI_A/BUSSM_A/N$266	DFF (SEU,1X)	CLK	Q	0.00	0.00	N	0.00	0.00
/SCI_A/BUSSM_A/U288	/SCI_A/BUSSM_A/N$633	AND_NAND_2 (1X)	IN2	OUT2	0.99	1.18	I	0.99	1.18
/SCI_A/BUSSM_A/U349	/SCI_A/BUSSM_A/N$648	NOR_2 (1X)	IN1	OUT1	1.56	0.95	I	2.74	1.94
/SCI_A/BUSSM_A/U333	/SCI_A/BUSSM_A/N$482	NAND_2 (1X)	IN2	OUT1	0.57	0.76	I	2.51	3.50
/SCI_A/BUSSM_A/U384	/SCI_A/BUSSM_A/N$230	AND_NAND_3 (1X)	IN3	OUT1	0.77	0.61	I	4.27	3.12
/SCI_A/BUSSM_A/U304	/SCI_A/BUSSM_A/N$177	NAND_4 (1X)	IN3	OUT1	0.48	0.81	I	3.61	5.08
/SCI_A/BUSSM_A/U366	/SCI_A/BUSSM_A/N$200	AND22OR (1X)	IN4	OUT1	0.63	0.69	N	4.24	5.77
/SCI_A/BUSSM_A/CURR_STATE_REG[3]		DFF (SEU,1X)	D	CLK	1.19	1.19	N	5.43	6.95

Source Clock Delay: 0.00
Destination Clock Delay: 0.00

Figure 5

Static Timing Analysis

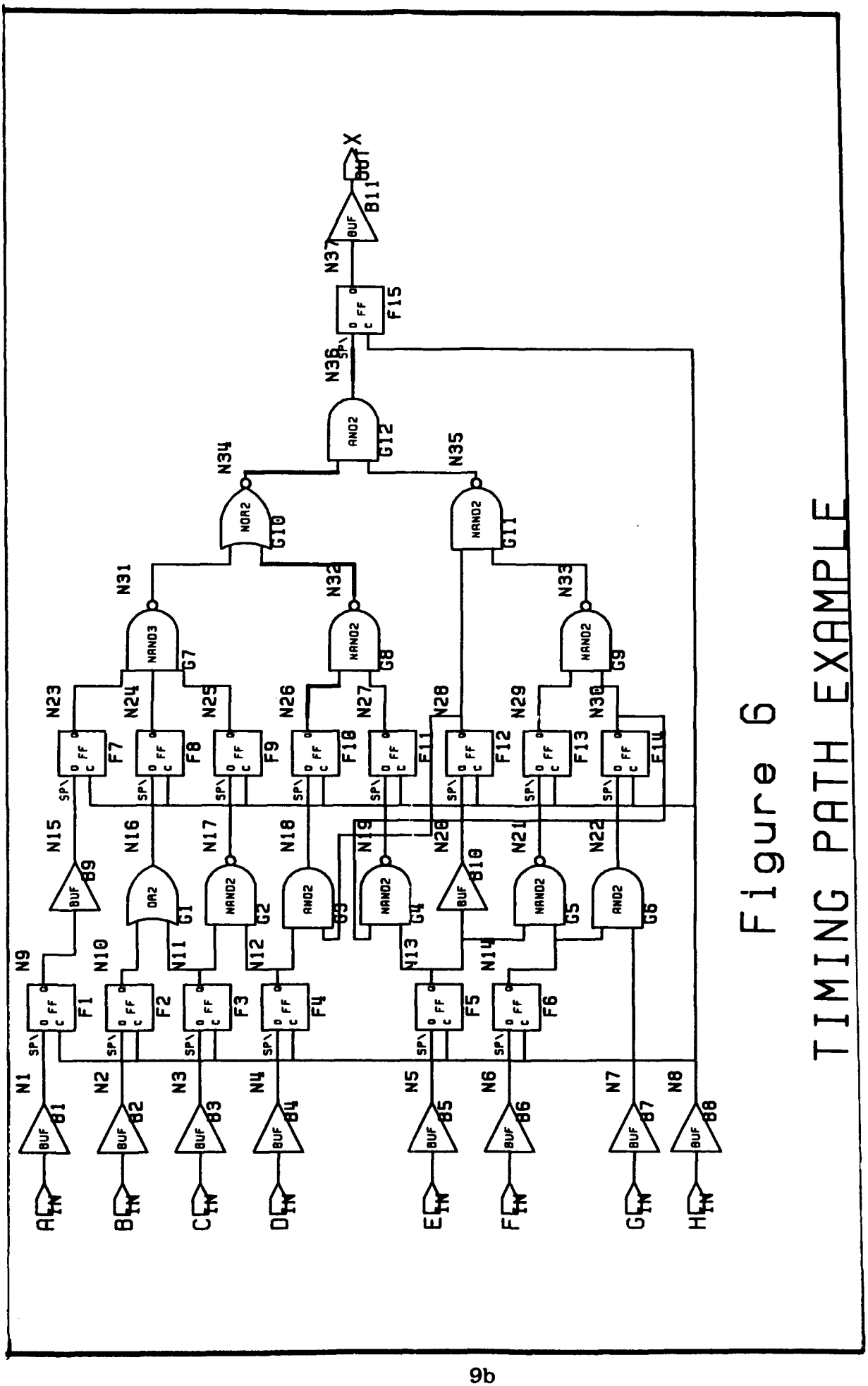

TIMING PATH EXAMPLE
A
B
C
D
E
F
G
H
X
BUF
OR2
NAND2
AND2
NOR2
NAND3
FF
N1
N2
N3
N4
N5
N6
N7
N8
N9
N10
N11
N12
N13
N14
N15
N16
N17
N18
N19
N20
N21
N22
N23
N24
N25
N26
N27
N28
N29
N30
N31
N32
N33
N34
N35
N36
N37
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
B11
G1
G2
G3
G4
G5
G6
G7
G8
G9
G10
G11
G12
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15

Figure 6

```
1 N26
1 N32
1 N34
1 N36
1 N37
4246 /TMI_A/N$802
4246 /TMI_A/TMBUSSM_A/N$475
4246 /TMI_A/TMBUSSM_A/N$347
4246 /TMI_A/N$97
4246 /TMI_A/PKTCNTOT_A/N$120
4246 /TMI_A/PKTCNTOT_A/N$174
4246 /TMI_A/PKTCNTOT_A/N$128
4246 /TMI_A/PKTCNTOT_A/N$21
4246 /TMI_A/PKTCNTOT_A/N$37
4247 /TMI_A/N$802
4247 /TMI_A/TMBUSSM_A/N$475
4247 /TMI_A/TMBUSSM_A/N$156
4247 /TMI_A/N$71
4247 /TMI_A/TMSTAT_A/N$74
4247 /TMI_A/TMSTAT_A/N$12
4247 /TMI_A/TMSTAT_A/N$169
4247 /TMI_A/TMSTAT_A/N$134
4248 /TMI_A/N$802
4248 /TMI_A/TMBUSSM_A/N$475
4248 /TMI_A/TMBUSSM_A/N$156
4248 /TMI_A/N$71
4248 /TMI_A/TMSTAT_A/N$74
4248 /TMI_A/TMSTAT_A/N$12
4248 /TMI_A/TMSTAT_A/N$88
4248 /TMI_A/TMSTAT_A/N$177
4249 /SCI_A/CNTBLK_A/N$32
4249 /SCI_A/CNTBLK_A/N$401
4249 /SCI_A/CNTBLK_A/N$443
4249 /SCI_A/CNTBLK_A/N$460
4249 /SCI_A/CNTBLK_A/N$85
4249 /SCI_A/CNTBLK_A/N$80
4249 /SCI_A/CNTBLK_A/N$279
4249 /SCI_A/CNTBLK_A/N$214
4250 /SCI_A/BUSSM_A/N$266
4250 /SCI_A/BUSSM_A/N$633
4250 /SCI_A/BUSSM_A/N$648
4250 /SCI_A/BUSSM_A/N$482
4250 /SCI_A/BUSSM_A/N$230
4250 /SCI_A/BUSSM_A/N$177
4250 /SCI_A/BUSSM_A/N$200
```

Figure 7

Timing Path Identification List

III. Switching Test Algorithm

The switching test algorithm generates a test which can be used to determine the path delay of an identified logic path between two bistables, provided that a test is possible. The algorithm guarantees that only the intended path is measured. The generated test vector defines the state of the register bits that must be scanned into the design. The test is then executed by applying two clocks with a defined interval. (See "Switching Test Algorithm Flowchart").

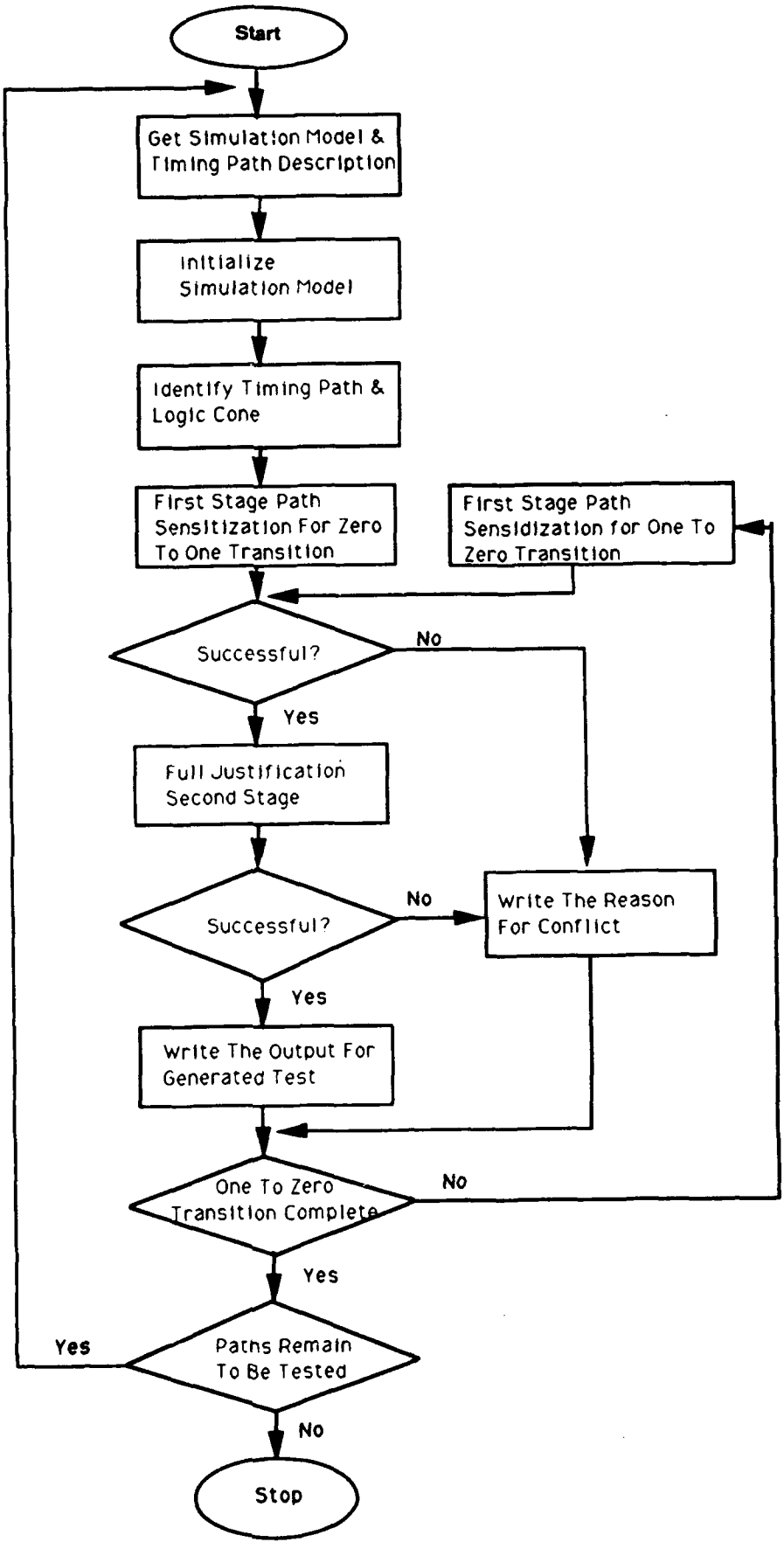
Delay Path Identification

After the logic model has been loaded and initialized, the first step in the switching test algorithm is the identification of every primitive logic gate in the path to be tested. The first timing path descriptor is read and all of the logic gates identified in the timing path including the driving and receiving (source and target) bistables, are marked. The timing path identification list is made at the netlist level before the design is modeled at the primitive gate level. For this reason the identification list usually does not identify a connected sequence of logic gates from the primary input bistable to the primary output bistable. Therefore, after the identified gates are marked, a depth first search is done from every marked gate to identify and mark a path through intervening gates which were not identified in the original path description. This produces a single marked path from the path source bistable to the target bistable. (Figure 6 shows a delay path in bold, however, the example is made from primitive gates so no intermediate gate identification was required). The gates in each path are marked in the following manner:

```
For each user specified path
    For each user specified net
        If first net of path then search backwards for source bistable
        Search forward from current path net to next path net.
        Identify target bistable between last path net and second last net
```

Logic Cone Identification

After the timing path of interest has been identified, all of the gates in the logic cone are identified from the path target bistable back to all of the bistables which drive into logic cone. The bistables that drive into the logic gates in the cone are identified as



Switching Test Algorithm Flowchart

first level bistables, and input pins are identified. In Figure 6 the first level bistables are F7 through F14 and the input pin is H IN. The cone trace is then continued from all of the first-level bistables back to primary input pins or second-level of bistables, where the cone is terminated (bistable F1 through F6 and input pin G IN). Only logic gates in the defined logic cone need be addressed for timing test generation. The first-level bistables are identified because the test generator establishes states in these bistables. These first-level bistables, except for the source bistable, will not change state during the first of the two test clock pulses. The second level bistables provide the logic states required to hold the first-level bistables.

Test Generation

Test generation starts with the initialized state of the logic cone. In the initialized state, all of the controllable primary inputs and bistables have been set to the unknown (X) state, and all logically tied gate inputs have been set to their tied state. The logic cone has then been simulated to establish a quiescent state for all of the gates. The test generation algorithm establishes the bistable states necessary to sensitize the logic gates in the defined timing path. Therefore a state transition on the path source bistable causes a state transition in every gate in the timing path and at the input of the target bistable. Additionally, the test generation algorithm assures that when the first functional clock is applied to the logic, only the source bistable changes state. This assures that the delay path, and only the delay path, is sensitive to the source bistable transition. Two tests are attempted for each delay path, a positive to negative transition, and a negative to positive transition at the source bistable. The reason that two tests are attempted is two-fold. Assuming both attempts are successful, if the path delay of one transition differs from that of the other, the user may want to measure the longest delay, or determine the amount of difference in the delays. Secondly, there are some logic configurations in which only one transition is testable, as when the target bistable is configured as a J/K flip-flop and the timing path drives only the J or K input.

Path Sensitization

The path sensitization approach is as follows. First a logic state is selected which will be scanned into the source bistable as the starting state for the logic transition. Next, opposite logic state is established on the data input to the source bistable, and if the source bistable has a clock enable, the clock enable is established. Each gate in the

delay path is then examined to determine the gate input states required to sensitize the delay path. The algorithm attempts to establish (justify) the gates driving the timing path gate, and subsequently their drive gates. This continues until the necessary states are established for the scannable bistables which drive the timing path cone. When a state is established for a bistable driving the timing path cone (first level bistable), the input state to the bistable is established such that the bistable will not transition on the first functional clock.

The delay path may drive into the target bistable as either a clock enable or as a data input. If the delay path drives into the clock enable, then a state is selected to be scanned into the bistable, and the opposite state is established on the data input of the bistable. If the delay path drives into the data input of the target bistable, its clock enable is established if applicable. If the data input will cause a bistable transition in only one direction, then the scanned in state will be the starting state for the transition. (If the target bistable is a D-Flip-Flop, the starting state of the target bistable is not important because the state on the data input will be clocked into the bistable regardless of its previous state). Thus, the gate states are established such that a state change in the source bistable will cause a state change on the input of the target bistable.

The delay path sensitization process relies heavily on two procedures called gate justification and gate implication. These procedures are used to establish gate states, and to simulate the effect of the logic state established on a gate. The path sensitization process utilizes a set of recursive procedures to establish the gate states in a manner that allows for choices to be made. It also allows for alternate choices (if available) when a choice does not produce the intended result .

Gate Implication

A gate is implicated when a state other than 'X' is to be established on its output. The process of implication simulates each gate driven by the gate which is implicated. Each driven gate is evaluated in the following manner:

```
If the output state of the gate is X, and
        the current input states produce an output other than X,
then apply the proper state to the output of the gate, and
        add this gate's loads to the evaluation queue (if they are not
        already in the queue),
else if the gate is pending a choice and
        the new evaluated state is the same as the required output
then add this gate to the pending_gates_hit list and exit
else conflict
```

The gate implication process for a given gate is completed by processing the evaluation queue. No state input choices are made during this procedure.

Gate Justification

A gate is justified to establish a required state on the output of the gate. When more than one set of input states will produce the required gate output, a choice may be made immediately, or may be deferred until a later time, depending on the requirements placed on the procedure by the calling routine. The procedure of gate justification is performed as follows:

```
If the output state of the gate is 'X' ,
        then call gate implicate to assign the gate state and implicate the gate
        else
        if no choice is available to establish the required gate output state,
                then conflict ( this could be a case, for instance, where an AND
                gate
                requires a ONE output but had a 'B' and an 'X' as inputs allowing
                only a 'B' or a ZERO as possible outputs)
        elseif only one input state choice exists
                then for each input, justify the driving gate if it had an 'X' state
        elseif no choice is to be made, then mark the gate pending
```

If the gate requiring justification is a first level bistable, it is processed like any other

gate except that when evaluating a bistable, the memory capability of the bistable is taken into account, i.e.

```
If clock input is disabled or data input state is same as desired bistable state
then no action is required
elseif the clock is enabled then
        if the data input state is X
        then justify the required data input state
        elseif the data input state is opposite the required state
         then conflict
                elseif the clock input state is X and
        the data input state is opposite the desired state
        then justify the clock disable input
                elseif the data input state is X
                then mark the gate pending
```

This procedure is used extensively during timing path sensitization.

Timing Path Sensitization, First Stage

The identified timing path is sensitized in two stages. Partial justification establishes all of the gate states required to sensitize the timing path without making any gate input choices. During this stage, if a gate is being justified, and has more than one choice of input states, then it is marked as pending an input choice. Its output is established and it is implicated. The timing path is sensitized by first justifying a state in the timing path source bistable, an opposite state on the data input to that bistable, and a clock enable to that bistable if applicable. This assures that the source bistable will change state when the first functional clock occurs. The gate states required to establish this condition on the bistable are justified until a choice is required. For each gate requiring a choice, the required gate output state is established and implicated, but a required input state is not established. Instead, the gate is marked as pending and will be evaluated after this path sensitization step is completed. After the source bistable logic state is established, the gates in the identified timing path are processed as follows:

```
The load gates of the timing-path element previously justified are examined
to locate the next timing path gate to be processed
        If the timing-path gate output state is not 'X'
          and the state is not the required state
        then conflict
        elseif the required logic state cannot be established
        then conflict
        else determine the required logic state and call gate justify
```

At the end of the first stage of Timing sensitization, for a falling edge transition at the source bistable F10, referring to Figure 6 for the timing path in bold, the following element output states are established: F10, G10, and G12 are 'D'; G8 is 'B'; B8, B9, F1, F6, F7, F8, F9, F11, H IN, G1, G2, G4, and G11 are '1'; and G3 and G7 are '0'. Gates G1, G2, G3, G4, and G11 are pending input choices.

Since no logic input choices are made during the path sensitization step, there are no unwinds. Therefore, any logic conflict immediately indicates that a test is not possible for the identified timing path as a single thread path. As each gate in the timing path is sensitized, a check is made to verify that there is only one sensitive input to the gate. This assures that there are no alternate sensitized paths which would invalidate the test. Should an alternate sensitive path be unavoidably established, this is noted and displayed by the test generation algorithm, and test generation proceeds as normal. If this process terminates with a conflict, then a message is printed which identifies the gate at which the conflict occurred.

Full Justification Second Stage

After the first stage of the timing path sensitization is complete, the gates on the pending gate stack are processed to justify their input states. This justification step requires logic choices whenever more than one gate input may be justified to establish the state of a pending gate. The maximum number of choices for any gate is the number of X state inputs on the gate. As each choice is made the choice is recorded on an activity stack along with the gate states established as a result of making that choice. The justification choices may also add gates to the pending gate stack. This occurs recursively. When a choice is made as a result of processing a gate from the pending gate stack, no further choices are made while justifying that choice until a new pending gate is removed from the pending gate stack. Also, as was the case when implicating gate states during timing path sensitization, if a gate state is implicated into a pending gate, the pending gate is evaluated. If this gate's state is not justified, and there is only one choice left, then that justification is immediately made, and the justifications and implications required to perform that justification are made.

To perform delay justification, the assignment stack and the pending stack are marked to indicate the starting point for delay justify. The delay justify routine will cause logic choices to be made, and if alternate choices are required, an unwind process uses the stack marks to determine how far to unwind. The delay justify routine is used to

process all of the pending gates, starting with the gate closest to the target bistable. Each pending gate is justified using the next available logic choice of input states. The implicate function associated with justification may establish the state of existing pending gates, may cause new pending gates to be identified, may result in a logic conflict. In any case, all of the activity is recorded on the activity stack and this stack is used for unwinds if they are necessary. For final justification, the pending gates are processed as follows:

```
Save an index to the top of the assign stack and the pend stack
Find the first pending gate to be justified (return if none found)
Loop through the inputs of the gate and find the first input with an X state
Call gate justify with the driving gate and required state
If justification is successful then recursively call full justify
if no conflict then return
else unwind to last index of the assign stack and the pending stack
        and call gate justify with the drive gate of the next available X input state.
```

If this process returns with a logic conflict, a message is written indicating that no test is possible for the identified timing path. The gate at which the logic conflict occurred is identified in the message.

IV Verification and Demonstration

A number of small test designs were generated for the purpose of verifying the functionality of the algorithm. (See Attachment A for schematics of these designs). These test designs were used for both algorithm verification and regression testing.

To facilitate the examination of test results, three test result output lists are produced. The first output is a summary report of the delay path characteristics (Figure 8 - 'Switching Test Delay Path Summary Report').

The second output is a summary report of the switching test results. (See Figure 9 - 'Switching Test Summary Output Report').

The third output is a detailed report for each switching test generation attempt. (See Figure 10 - "Switching Test Detail Output Report"). This report identifies the path and the logic transition of the test. It notes whether or not a test was successfully generated and, if not, the reason for not obtaining a test, and the gate at which the test conflict occurred.

path	len	ph	cir	cen	ten	L1	L2	L2o	L1ff	L2ff	L2ffo	L1in	L2in	L2ino	envel
1	5	0	NO	NO	NO	16	20	16	8	8	6	1	2	1	4
4246	16	1	YES	NO	NO	462	1505	1043	53	99	46	7	9	2	18
4247	14	0	YES	NO	NO	447	1641	1194	51	118	67	7	11	4	13
4248	12	0	YES	NO	NO	460	1365	905	51	93	42	7	11	4	11
4249	13	1	YES	NO	NO	598	2259	1661	69	160	91	7	10	3	18
4250	11	0	YES	NO	NO	676	2397	1721	80	162	82	7	10	3	36

Header

path : Path number

len: The path length in number of gates, including both the source and target bistables.

ph: Contains a 1 if the path is inverting and a 0 if not inverting.

cir: Contains a NO if the target bistable does not drive back into the delay path cone and a YES if it does.

cen: Identifies whether the delay path drives a clock enable input of the target bistable.

ten: Identifies whether the delay path drives through a tristate enable.

L1: Lists the number of elements in the first level of the delay path cone.

L2: Lists the number of elements in the second level of the cone.

L2o: Lists the number of elements only in the second level of the cone.

L1ff: Lists the number of first level bistables.

L2ff: Lists the number of second level bistables.

L2ffo: Lists the number of bistables that are only second level

L1in: Lists the number of input pins driving the first level of the cone

L2in: Lists the number of input pins driving the second level of the cone

L2ino: Lists the number of input pins driving only the second level of the cone.

envel: Lists the number of gates in the envelope formed by forward tracing from the source bistable and backtracing from the target bistable, including the source bistable. (This is the number of gates in the delay path, plus the number of gates in the paths sourced by, and reconvergent with, the delay path).

Figure 8
Switching Test Delay Path Summary Report

path	trn	rslt	unwind	assign	max_pend	max_depth
1	R	PATH	0	27	2	0
1	F	GOOD	0	30	5	6
4246	R	PATH	0	557	7	0
4246	F	GOOD	35	3737	10	15
4247	R	PATH	0	476	8	0
4247	F	GOOD	0	867	12	10
4248	R	PATH	0	456	8	0
4248	F	GOOD	1	905	12	11
4249	R	PATH	0	140	13	0
4249	F	PATH	0	306	17	0
4250	R	GOOD	0	1190	9	19
4250	F	PATH	0	346	6	0

Header

path: Path number

trn: Contains an 'R' for a test with a rising edge transition at the source bistable, and an 'F' for a test with a falling edge transition at the source bistable.

rslt: Lists one of four possible test results. 'GOOD' indicates that a test was successfully generated. 'PATH' indicates that a test was not generated due to the inability to sensitize the delay path during the first stage of timing path sensitization. 'FULL' indicates that a test was not generated due to an unresolvable conflict during the second stage of timing path sensitization. 'LIMT' indicates that no test was generated due to having exhausted the unwind limit imposed by the user, before a test could be generated.

unwind: Lists the number of unwinds that the test generation attempt used

assign: Lists the total number of gate assignments that were made by the test generation attempt.

max_pend: Lists the maximum number of pending gates.

max_depth: Lists the maximum depth of the gate justification recursion.

Figure 9

Switching Test Summary Output Report

```
-----------------------------------------
Path    1, non inverting, drives data input
Source : /F10
         N26
Target : /I$19
         N37
----------------------
Path    1, rising transition at source
***** warning,
      delay path gate forced to logic 1 or to logic 0 value
      some time during justification of delay path
      at gate index  25
      at net N36
      at instance /G12

***** warning,
      transition could not be pushed through delay path

   unwinds =  0
   assigns =  27

----------------------
Path    1, falling transition at source
     max recursion depth for full justification =  6
Test found
   unwinds =  0
   assigns =  30

-----------------------------------------
-----------------------------------------
Path 4246, inverting, drives data input
Source : /TMI_A/DELAYBLK_A/DELAYREG3_REG/I$1335
         /TMI_A/DELAYBLK_A/DELAYREG3_REG/N$1385
Target : /TMI_A/PKTCNTOT_A/PKTOTREG_REG[1]/I$1335
         /TMI_A/PKTCNTOT_A/PKTOTREG_REG[1]/N$1385
----------------------
Path 4246, rising transition at source
***** warning,
      delay path gate forced to logic 1 or to logic 0 value
      some time during justification of delay path
      at gate index  1113
      at net /TMI_A/PKTCNTOT_A/U56/N$782
      at instance /TMI_A/PKTCNTOT_A/U56/I$767

***** warning,
      transition could not be pushed through delay path


----------------------
Path 4246, falling transition at source
Test found

-----------------------------------------
-----------------------------------------
Path 4247, non inverting, drives data input
Source : /TMI_A/DELAYBLK_A/DELAYREG3_REG/I$1335
         /TMI_A/DELAYBLK_A/DELAYREG3_REG/N$1385
Target : /TMI_A/TMSTAT_A/STATREG_REG[6]/I$1335
         /TMI_A/TMSTAT_A/STATREG_REG[6]/N$1385
----------------------
```

Figure 10

```
Path 4247, rising transition at source
***** warning,
      justification of path gate failed
      at gate index  3269
      at net /TMI_A/N$71
      at instance /TMI_A/TMBUSSM_A/U271/I$1440
      place where conflict happened is
      at gate index  3392
      at net /TMI_A/N$476
      at instance /TMI_A/PKTCNTOT_A/U71/I$1440

***** warning,
      transition could not be pushed through delay path


----------------------
Path 4247, falling transition at source
Test found

-----------------------------------------
-----------------------------------------
Path 4248, non inverting, drives data input
Source : /TMI_A/DELAYBLK_A/DELAYREG3_REG/I$1335
         /TMI_A/DELAYBLK_A/DELAYREG3_REG/N$1385
Target : /TMI_A/TMSTAT_A/STATREG_REG[4]/I$1335
         /TMI_A/TMSTAT_A/STATREG_REG[4]/N$1385
----------------------
Path 4248, rising transition at source
***** warning,
      justification of path gate failed
      at gate index  3269
      at net /TMI_A/N$71
      at instance /TMI_A/TMBUSSM_A/U271/I$1440
      place where conflict happened is
      at gate index  3392
      at net /TMI_A/N$476
      at instance /TMI_A/PKTCNTOT_A/U71/I$1440

***** warning,
      transition could not be pushed through delay path


----------------------
Path 4248, falling transition at source
Test found

-----------------------------------------
-----------------------------------------
Path 4249, inverting, drives data input
Source : /SCI_A/CNTBLK_A/CNTREG_REG[6]/I$1335
         /SCI_A/CNTBLK_A/CNTREG_REG[6]/N$1385
Target : /SCI_A/CNTBLK_A/CNTREG_REG[1]/I$1335
         /SCI_A/CNTBLK_A/CNTREG_REG[1]/N$1385
----------------------
Path 4249, rising transition at source
***** warning,
      delay path gate forced to logic 1 or to logic 0 value
      some time during justification of delay path
      at gate index  2915
      at net /SCI_A/CNTBLK_A/N$485
```

Figure 10

```
      at instance /SCI_A/CNTBLK_A/U87/I$501

***** warning,
      transition could not be pushed through delay path


----------------------
Path 4249, falling transition at source
***** warning,
      justification of path gate failed
      at gate index  3338
      at net /SCI_A/CNTBLK_A/N$401
      at instance /SCI_A/CNTBLK_A/U107/I$632
      place where conflict happened is
      at gate index  2430
      at net /SCI_A/CNTBLK_A/N$111
      at instance /SCI_A/CNTBLK_A/U71/I$471

***** warning,
      transition could not be pushed through delay path


-----------------------------------------
-----------------------------------------
Path 4250, non inverting, drives data input
Source : /SCI_A/BUSSM_A/CURR_STATE_REG[2]/I$1335
         /SCI_A/BUSSM_A/CURR_STATE_REG[2]/N$1385
Target : /SCI_A/BUSSM_A/CURR_STATE_REG[3]/I$1335
         /SCI_A/BUSSM_A/CURR_STATE_REG[3]/N$1385
----------------------
Path 4250, rising transition at source
Test found

----------------------
Path 4250, falling transition at source
***** warning,
      justification of path gate failed
      at gate index  3103
      at net /SCI_A/BUSSM_A/N$648
      at instance /SCI_A/BUSSM_A/U349/I$550
      place where conflict happened is
      at gate index  2935
      at net /SCI_A/BUSSM_A/N$482
      at instance /SCI_A/BUSSM_A/U333/I$950

***** warning,
      transition could not be pushed through delay path


-----------------------------------------
```

Figure 10

Switching Test Detailed Output Report

The Test-Bus Interface Unit (TIU) design for the RH32 program was selected as the demonstration vehicle. This is a design of about 10,000 logic gates, which uses state machines, counters and shift registers for logic control. This design had been coded in synthesizable VHDL code using a vendor-supplied VHDL synthesis program. The design was modeled with primitive logic gate fault models for timing test generation. Honeywell's static timing analyzer was used to identify 5000 of the longest logic paths in the design, for timing test generation. The switching test algorithm was applied to these paths, and only a very small number (~10) of tests were successfully generated.

Upon analysis, it was determined that the high number of unsuccessful attempts were due to the architecture of the TIU design not being within the capability of the algorithm. The algorithm relies on the ability to switch only one bistable in order to cause a transition on a path. Some logic constructs, namely counters, shift registers, and state machines do not lend themselves to this requirement. The long TIU logic paths are virtually all driven by these kind of constructs. For example, a large majority of the longest timing paths identified by the timing analyzer, were sourced by a five bit counter which counts packet bits. The counter counts from zero to seventeen to signify that a packet of data has been received and then clears to zero for the next packet count. The count of sixteen signifies that the data portion of a packet has been received. This count provides the source for a large number of long delay paths. The counter is fully decoded and the decode count of sixteen is enabled by stepping from OF hex to 10 hex. There is no way to reach the 10 hex state by switching a single counter bit. Therefore, all of the timing paths driven by this decode are untestable by the switching test algorithm. It could be argued that timing testability could be improved by using a different decode scheme or gray code counters. However there are some constructs, which are untestable without adding test logic. If under normal operation a timing path is only enabled when multiple bistables are switched then the test should replicate that action. In order to handle these logic constructs, the algorithm would need to be extended to allow multiple path inputs to be simultaneously switched under carefully controlled conditions.

Although schedule and resource limitations prevented the analysis of a second test case, a modification was made to the path selection procedure of the timing analyzer. The modification allowed a limited number of the longest paths between each pair of bistables to be selected for timing test generation. Out of approximately 5000 paths selected by this method, over 150 tests were successfully generated. (Figure 11 - 'Switching Summary Report for Generated Timing Test'). The effect of this modification to the timing path selection process, was the identification of a larger number of timing paths that did not require the simultaneous switching of first level bistables in order to sensitize the path.

path	trn	rslt	unwind	assign	max_pend	max_depth
225	F	GOOD	0	1556	8	8
415	F	GOOD	0	1561	8	8
439	F	GOOD	0	1546	8	6
2261	R	GOOD	0	1118	9	25
2891	R	GOOD	0	1164	9	13
3071	F	GOOD	11	1945	8	14
3349	R	GOOD	1	1475	12	18
3349	F	GOOD	0	1287	10	18
3350	R	GOOD	2	1485	12	19
3350	F	GOOD	1	1297	10	19
3351	R	GOOD	1	1469	12	19
3351	F	GOOD	0	1281	10	19
3352	R	GOOD	2	1486	12	19
3352	F	GOOD	1	1298	10	19
3360	F	GOOD	0	1172	13	8
3409	F	GOOD	11	1936	8	14
3519	R	GOOD	0	1287	11	14
3574	F	GOOD	0	911	5	12
3636	F	GOOD	0	974	15	8
3661	R	GOOD	10	1356	11	8
3765	R	GOOD	0	1173	10	7
3859	F	GOOD	0	825	6	6
3898	F	GOOD	0	1180	13	8
3908	F	GOOD	11	1931	8	14
3924	R	GOOD	0	960	10	6
4014	F	GOOD	0	868	7	7
4060	F	GOOD	0	1180	14	8
4074	F	GOOD	0	941	7	8
4077	F	GOOD	0	1740	14	22
4137	F	GOOD	0	1740	16	22
4176	R	GOOD	1	1346	13	15
4203	R	GOOD	0	1073	10	5
4224	F	GOOD	0	1209	9	12
4239	R	GOOD	1	1379	13	15
4246	F	GOOD	35	3737	10	15
4247	F	GOOD	0	867	12	10
4248	F	GOOD	1	905	12	11
4250	R	GOOD	0	1190	9	19
4287	R	GOOD	0	1090	8	10
4360	F	GOOD	43	4332	8	15
4435	F	GOOD	0	1017	8	8
4530	R	GOOD	0	353	4	4
4530	F	GOOD	0	353	4	4
4537	F	GOOD	0	576	4	2
4539	F	GOOD	0	414	3	2
4540	F	GOOD	0	440	3	2
4541	F	GOOD	0	431	3	2
4542	F	GOOD	0	436	3	2
4546	F	GOOD	0	1090	12	12
4553	F	GOOD	1	839	4	7
4554	F	GOOD	1	756	4	7
4555	F	GOOD	1	843	4	7
4556	F	GOOD	1	819	4	7
4560	R	GOOD	0	1542	11	13
4579	R	GOOD	0	1173	10	7
4605	F	GOOD	0	736	4	2
4606	F	GOOD	0	1188	9	28
4622	F	GOOD	0	1194	13	8

Figure 11

4624	F GOOD	0	684	4	2
4625	F GOOD	0	684	4	2
4647	F GOOD	1	1283	13	18
4651	R GOOD	0	903	11	11
4668	R GOOD	1	1066	15	12
4689	F GOOD	0	837	7	8
4748	F GOOD	0	810	13	12
4774	F GOOD	119	5830	9	9
4779	F GOOD	0	1278	17	10
4782	F GOOD	0	452	6	2
4786	F GOOD	7	876	6	5
4842	F GOOD	0	994	5	5
4843	F GOOD	0	986	5	5
4847	F GOOD	2	1470	12	21
4877	F GOOD	0	476	6	7
4884	R GOOD	0	689	10	7
4887	R GOOD	0	332	4	3
4887	F GOOD	0	332	4	3
4903	F GOOD	143	7818	12	14
4906	F GOOD	0	477	4	6
4908	R GOOD	0	890	9	13
4914	F GOOD	0	477	4	6
4917	R GOOD	1	440	9	7
4917	F GOOD	0	653	9	16
4918	F GOOD	1	605	18	14
4926	F GOOD	0	535	6	7
4927	F GOOD	0	836	8	12
4930	F GOOD	0	819	10	12
4932	F GOOD	0	753	6	4
4940	F GOOD	1	1115	10	23
4942	F GOOD	3	1362	13	30
4949	R GOOD	0	1429	8	19
4952	F GOOD	0	594	8	8
4955	R GOOD	0	457	4	3
4955	F GOOD	0	329	4	3
4977	F GOOD	11	2122	8	16
4979	R GOOD	1	492	9	8
4979	F GOOD	0	490	10	7
4989	R GOOD	0	945	10	11
4989	F GOOD	0	945	10	11
4990	R GOOD	0	418	9	14
4990	F GOOD	0	572	9	19
4994	F GOOD	0	1019	12	15
4995	R GOOD	0	960	10	11
4995	F GOOD	0	960	10	11
4996	F GOOD	0	603	7	9
4997	R GOOD	1	1525	12	24
4997	F GOOD	2	1408	17	25
4998	R GOOD	1	1528	12	24
4998	F GOOD	2	1411	17	25
5010	R GOOD	792	41574	15	19
5022	R GOOD	0	945	10	11
5022	F GOOD	0	945	10	11
5023	R GOOD	0	947	10	11
5023	F GOOD	0	947	10	11
5024	R GOOD	0	961	10	11
5024	F GOOD	0	961	10	11
5025	R GOOD	793	41728	16	20
5027	R GOOD	0	950	10	11
5027	F GOOD	0	950	10	11

Figure 11

5028	R GOOD	0	979	10	11
5028	F GOOD	0	979	10	11
5029	R GOOD	0	948	10	11
5029	F GOOD	0	948	10	11
5030	R GOOD	0	973	10	11
5030	F GOOD	0	973	10	11
5031	R GOOD	0	949	10	11
5031	F GOOD	0	949	10	11
5032	R GOOD	0	985	10	11
5032	F GOOD	0	985	10	11
5034	R GOOD	0	970	10	11
5034	F GOOD	0	970	10	11
5035	R GOOD	0	948	10	11
5035	F GOOD	0	948	10	11
5048	R GOOD	793	41727	20	20
5051	R GOOD	794	41824	21	22
5052	R GOOD	794	41824	21	22
5055	R GOOD	794	41826	14	22
5056	R GOOD	794	41819	14	22
5057	R GOOD	794	41826	14	22
5066	F GOOD	0	336	5	4
5115	R GOOD	0	963	10	11
5115	F GOOD	0	963	10	11
5121	R GOOD	1	642	6	4
5122	R GOOD	792	41575	20	19
5122	F GOOD	792	41575	20	21
5123	R GOOD	792	41576	20	19
5124	R GOOD	792	42175	20	19
5125	R GOOD	792	41575	20	19
5126	R GOOD	792	41575	20	19
5126	F GOOD	794	41784	20	24
5132	F GOOD	2	1158	11	25
5137	R GOOD	794	41849	21	22
5138	R GOOD	794	41843	16	22
5186	R GOOD	0	352	3	3
5186	F GOOD	0	352	4	3
5196	R GOOD	794	41845	21	22

Figure 11

Switching Summary Report for Generated Timing Tests

V Conclusion

The switching test algorithm produces results which fully satisfy the intended algorithmic capability for a limited number of paths on the TIU demonstration case. The algorithm identifies timing paths from an incomplete list of nodes. Identified paths are automatically marked for timing test generation. Automated timing tests are generated for each path for both a source rising edge transition and falling edge transition, if allowed by the configuration of the path. If either or both of the tests cannot be generated, the logic conflict which prohibits test generation is identified. The measurable path delay, for tests which are generated, is determined only by the delay of the elements in the path and does not rely on timing constraints for any control elements driving into the path. The generated timing tests can be applied by use of design-for-test scan paths.

Timing tests could be generated for only a limited number of paths, in the TIU test case, because a large majority of the timing paths required that multiple path control bistables be simultaneously switched in order to achieve path sensitization. This simultaneous switching requirement was due to the heavy use of state machines and counters to provide control signals. Other design architectures such as pipelined logic and data path designs using microcoded control, would have a higher percentage of timing paths which are sensitizable without simultaneous switching. The algorithm would therefore be more effective on these design architectures. Extensions would provide a capability with a wider range of applications. These extensions could provide for the simultaneous switching of first level bistables, when required, to successfully generate a test. This would result in the simultaneous sensitization of multiple paths. The algorithm extension, however, presents some challenges and issues.

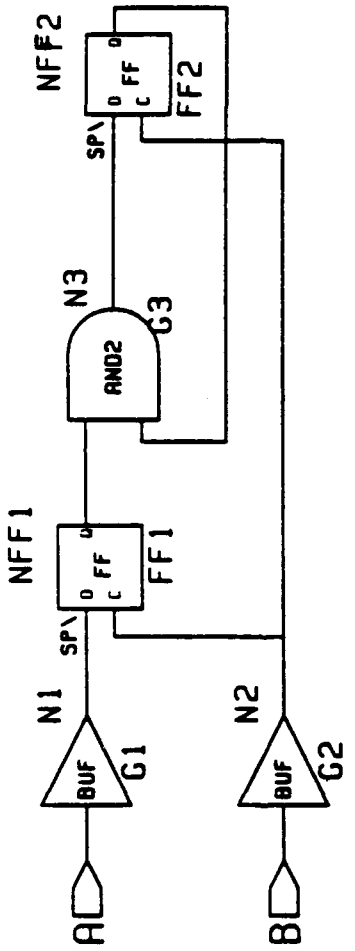
First is the challenge of assuring that only necessary paths are sensitized. The sensitization of multiple paths produces the possibility that the desired path is not the path that is actually tested. The more paths that are sensitized, the higher the chance that an alternate path will be tested. If an unintended alternate path is being tested, and the tested path is shorter than the intended path, misleading results will be obtained. Even if the tested path is longer than the intended path, misleading results would be obtained if the intent of the test is to verify some specific simulation times per Table 1.

Second, when the algorithm is required to choose between paths to be sensitized, timing information may be required in order to make the most appropriate choices. However, the object of the test is to measure actual delay times which are not known with certainty until the test is run. Therefore, the delay times supplied to the algorithm may cause inappropriate choices to be made.

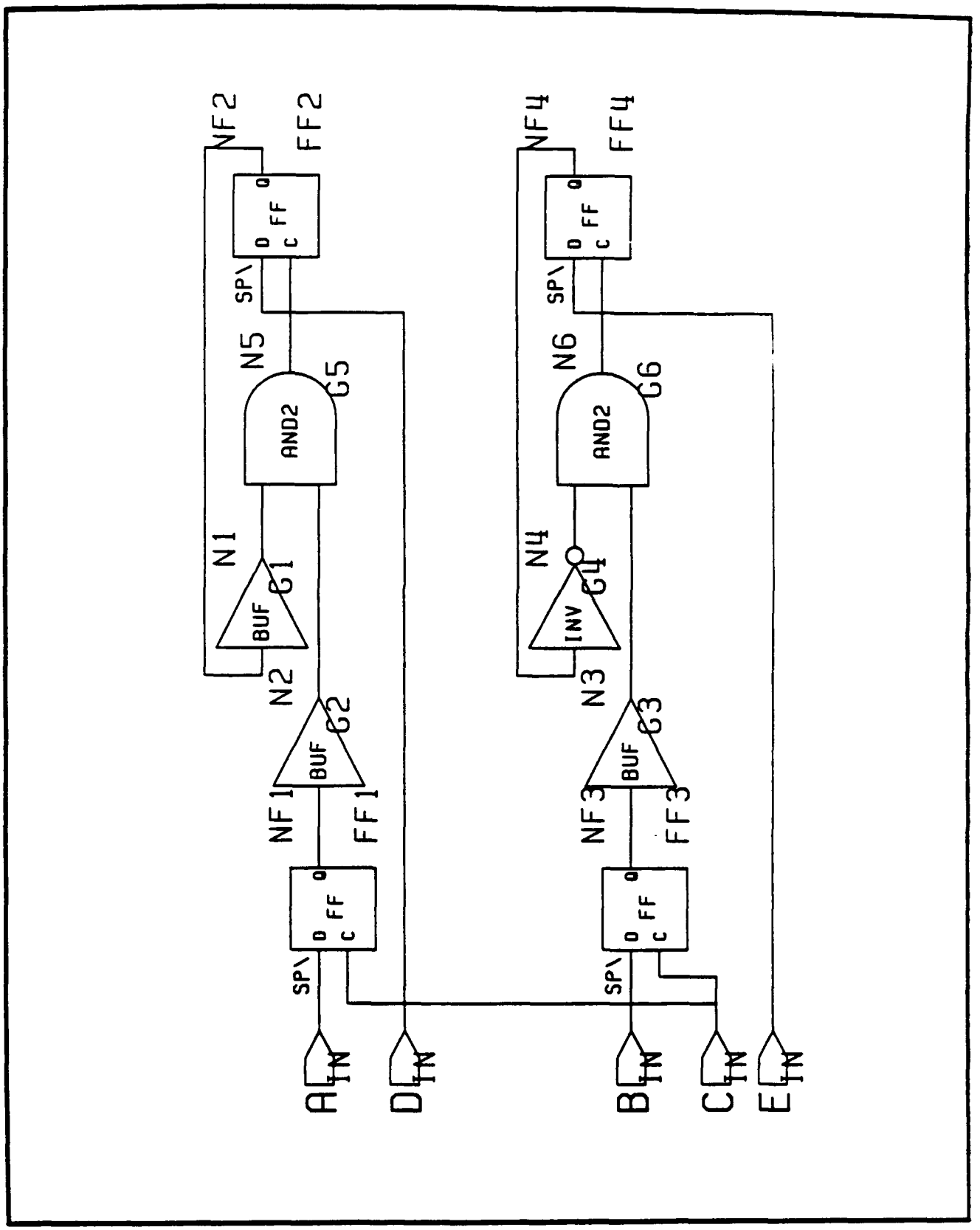
Third, when more than one path is sensitized, it is impossible to know which path was actually tested when the test is run. Therefore, information must be presented to the user describing each sensitized path that could be tested by each test vector.

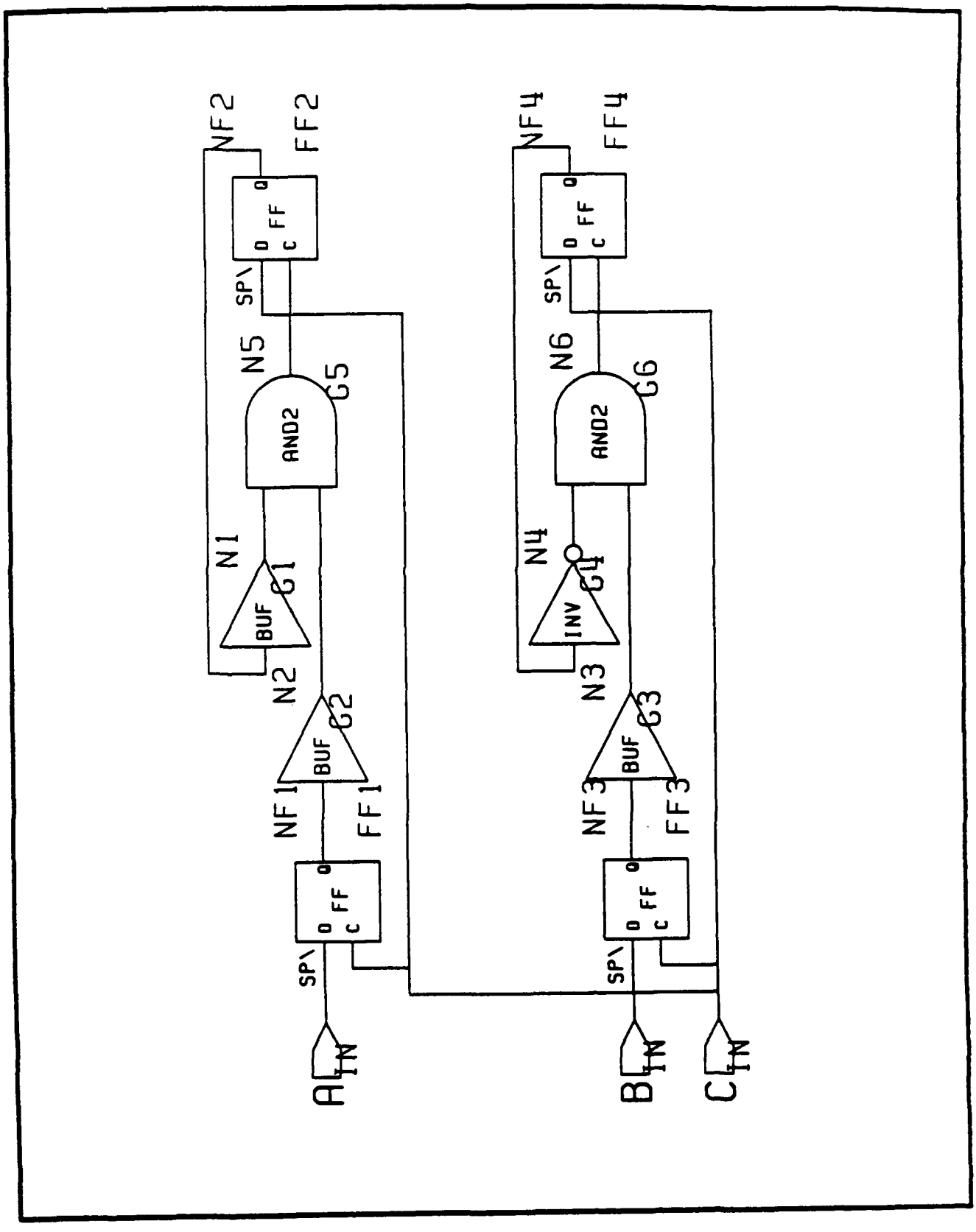
Fourth, the increase in the gate states that would have to be handled by the algorithm could make test generation times prohibitive.

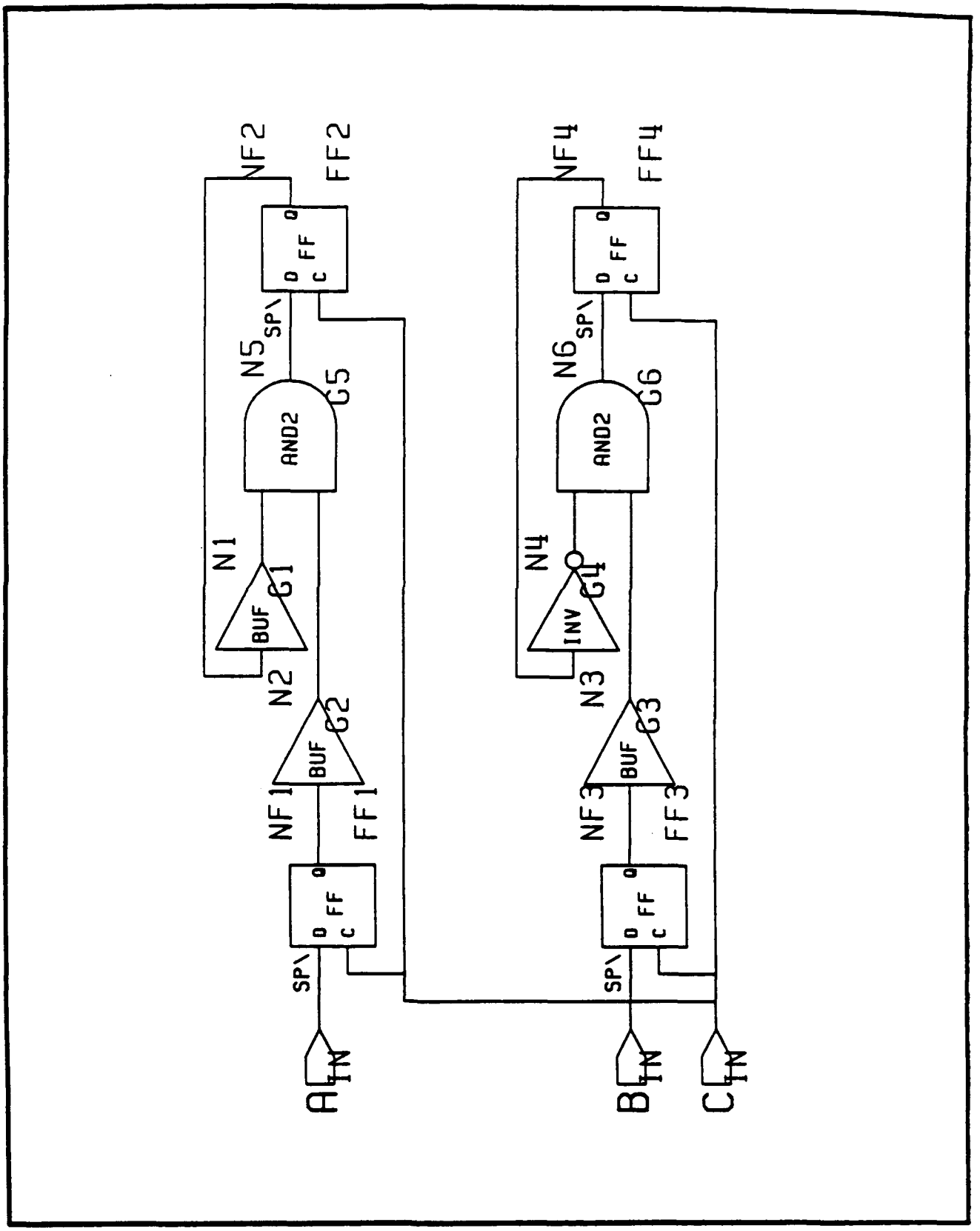
While there may be other reasons preventing timing test generation for certain timing paths, our investigations have identified the requirement for switching multiple bistables as the most prevalent cause for not successfully generating tests for a large percentage of identified paths in the RH32 TIU. Future work should be directed in this area.

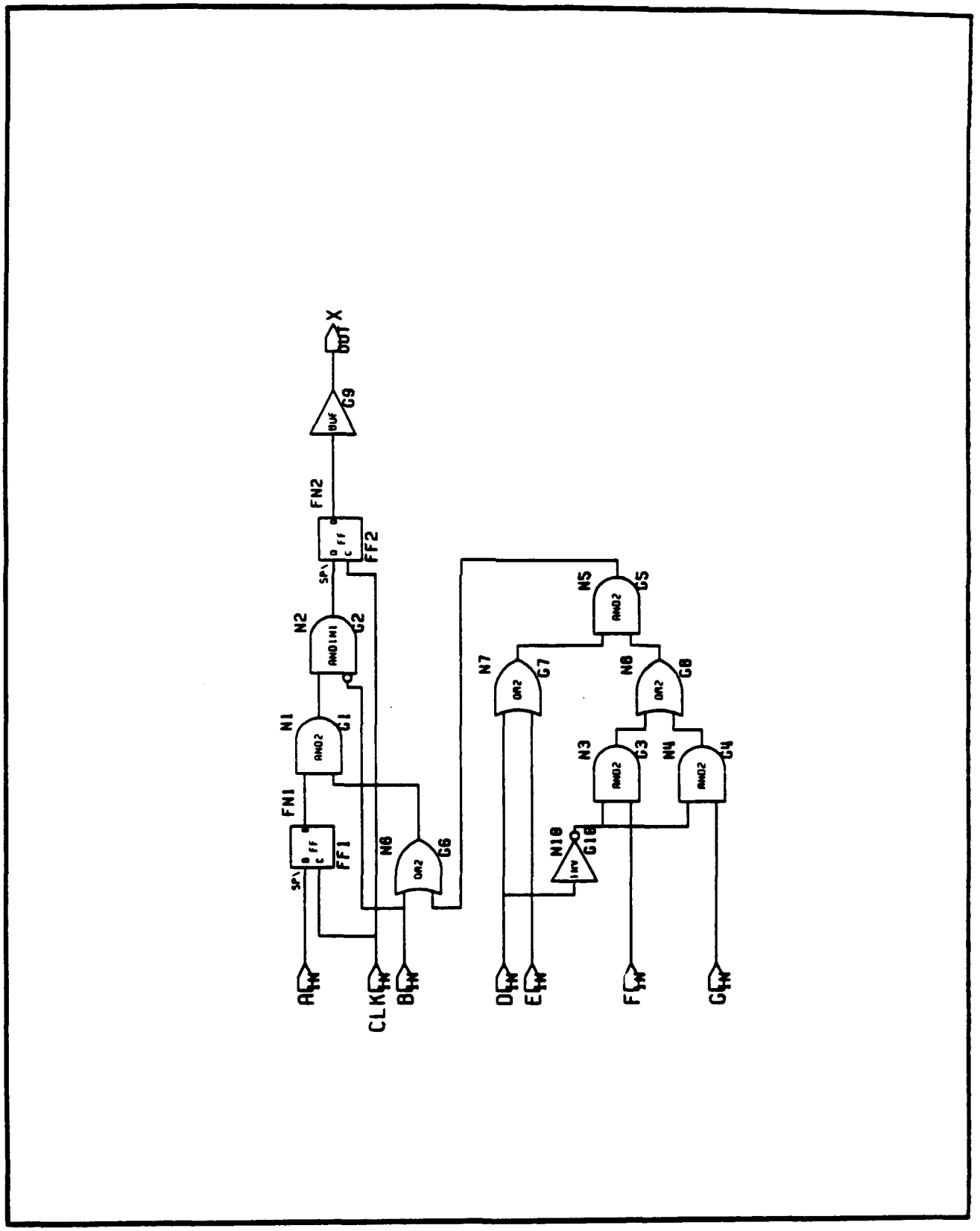


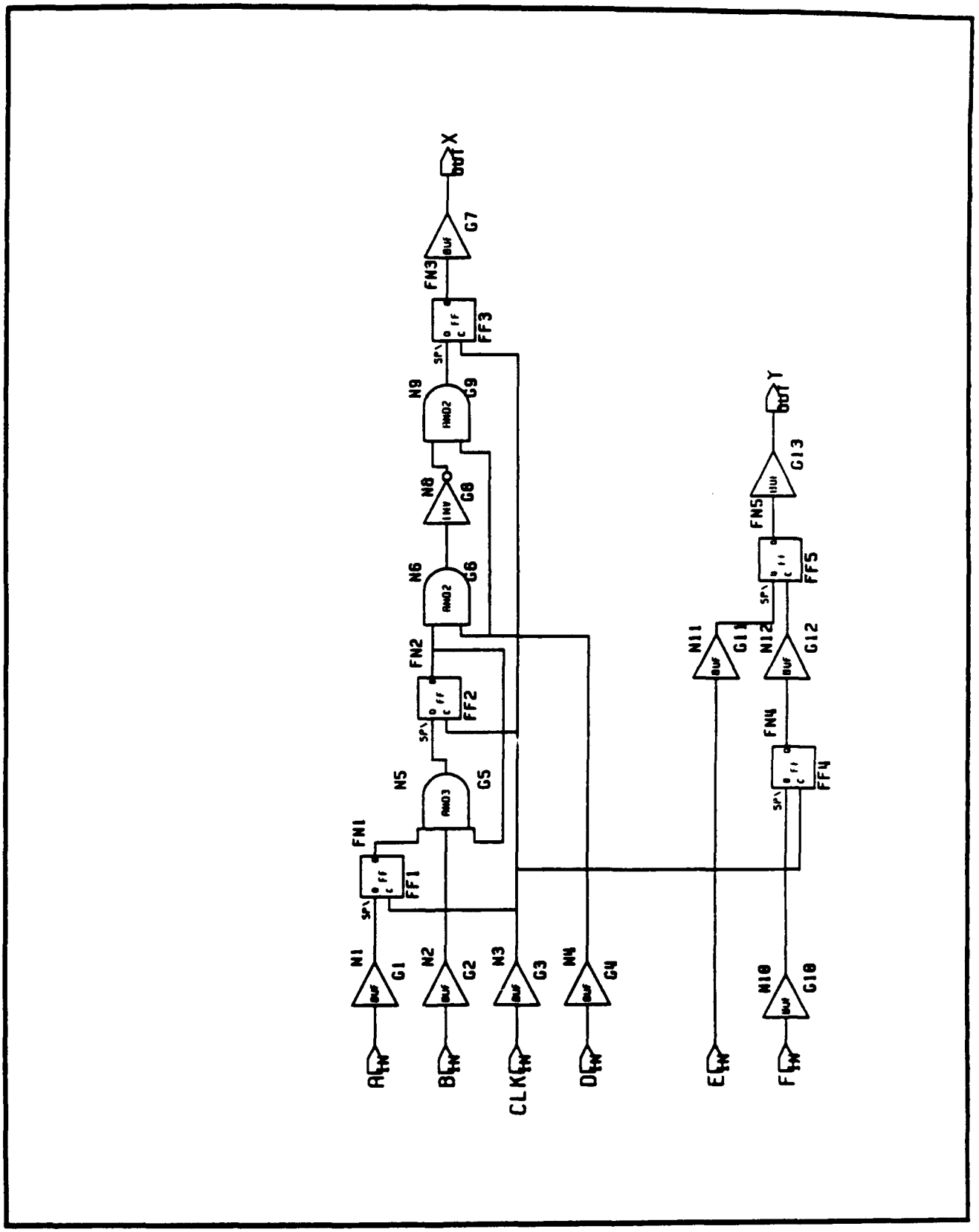
Attachment A Pg 1 of 12
VERIFICATION TEST CASES

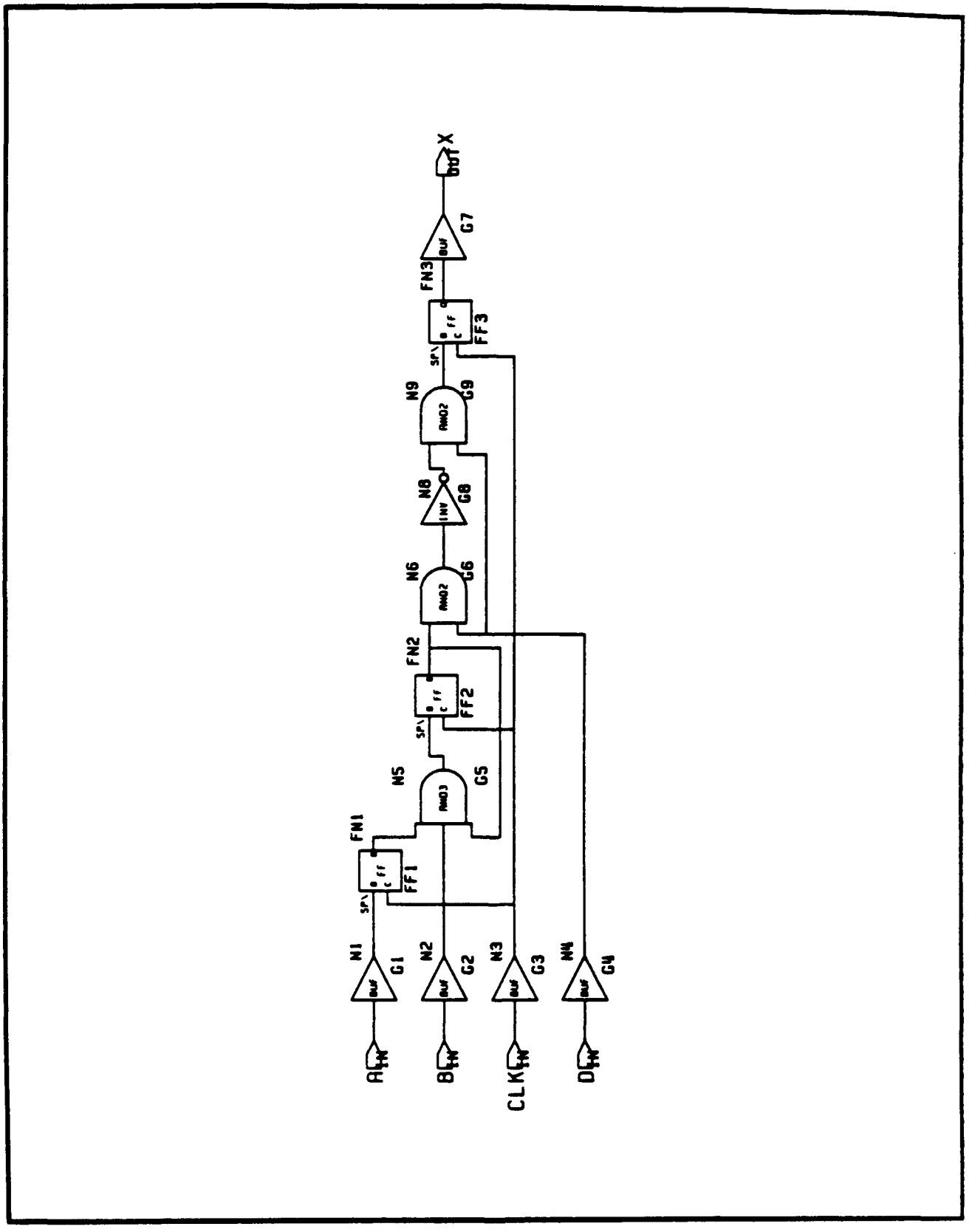

NF2
FF2
NF4
FF4
SP\
D
C
FF
N5
N6
AND2
G5
G6
N1
N4
BUF
INV
G1
G4
N2
N3
G2
G3
NF1
FF1
NF3
FF3
A IN
D IN
B IN
C IN
E IN

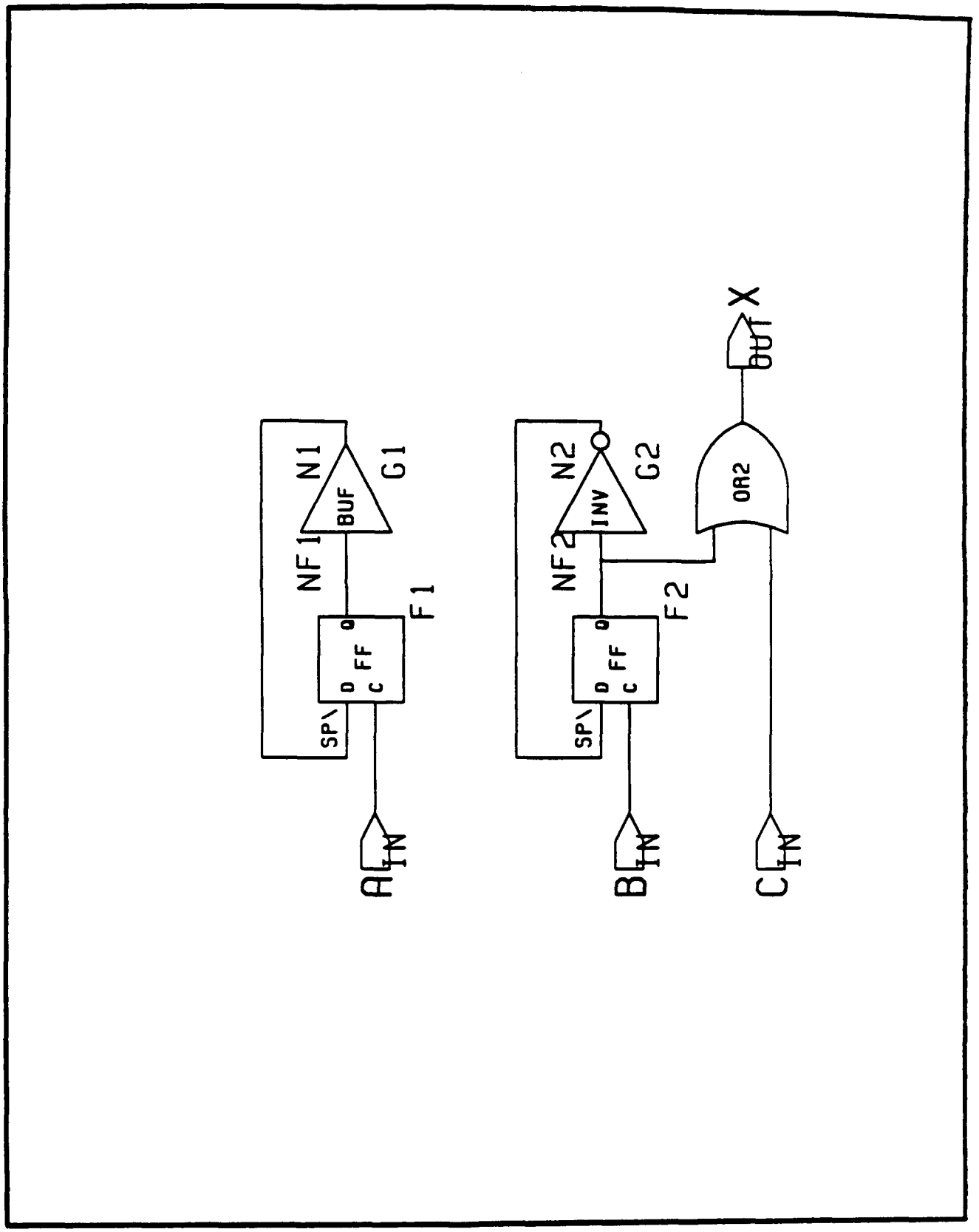

NF2
FF2
NF4
FF4
SP\
D
C
FF
Q
N5
N6
AND2
G5
G6
N1
N4
BUF
INV
G1
G4
N2
N3
G2
G3
NF1
FF1
NF3
FF3
A
B
C
IN

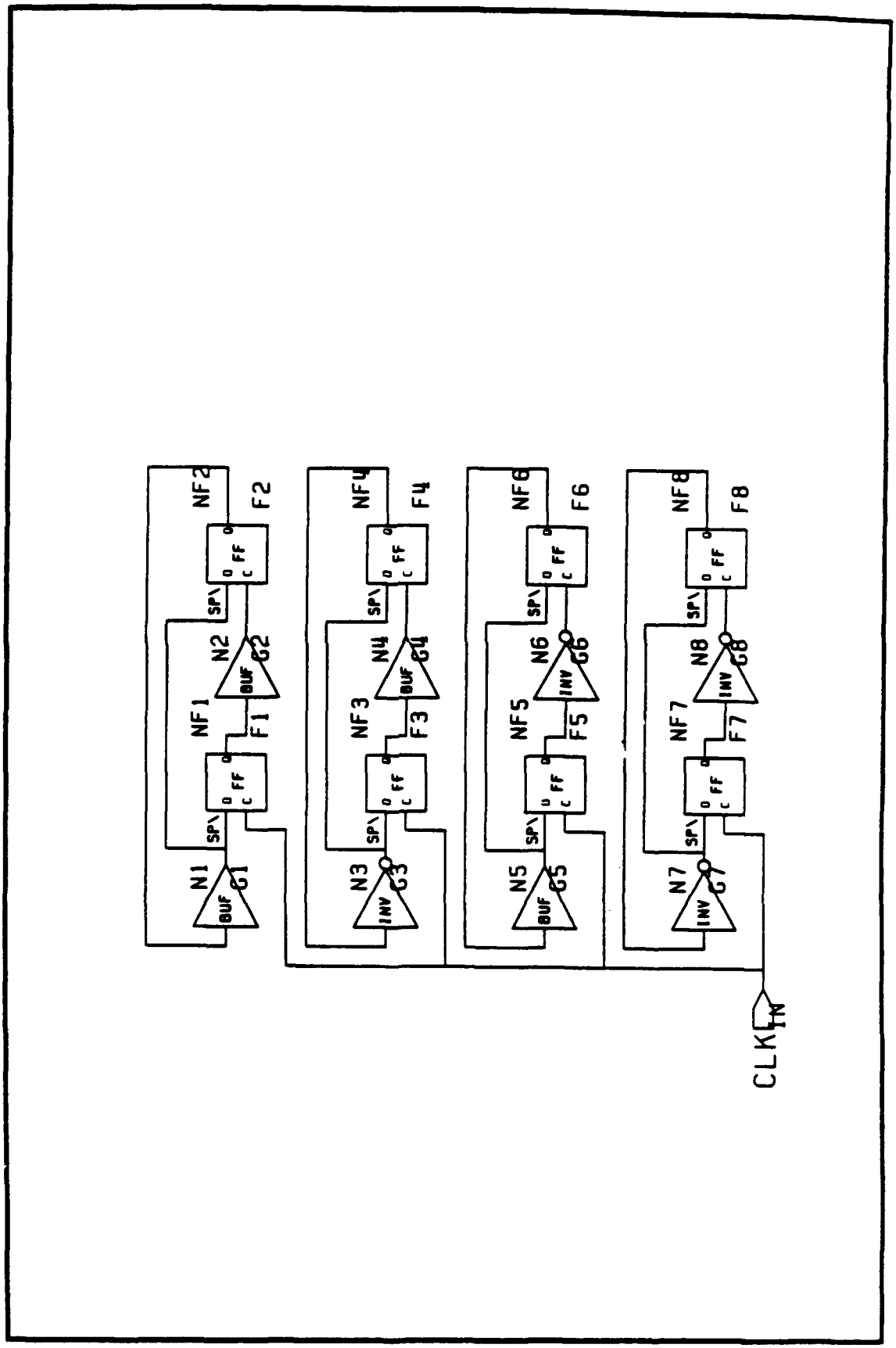

NF2
FF2
NF4
FF4
N5
SP\
N6
SP\
G5
AND2
G6
AND2
N1
G1
BUF
N4
G4
INV
N2
G2
BUF
N3
G3
BUF
NF1
FF1
NF3
FF3
D
C
FF
Q
SP\
A
IN
B
IN
C
IN

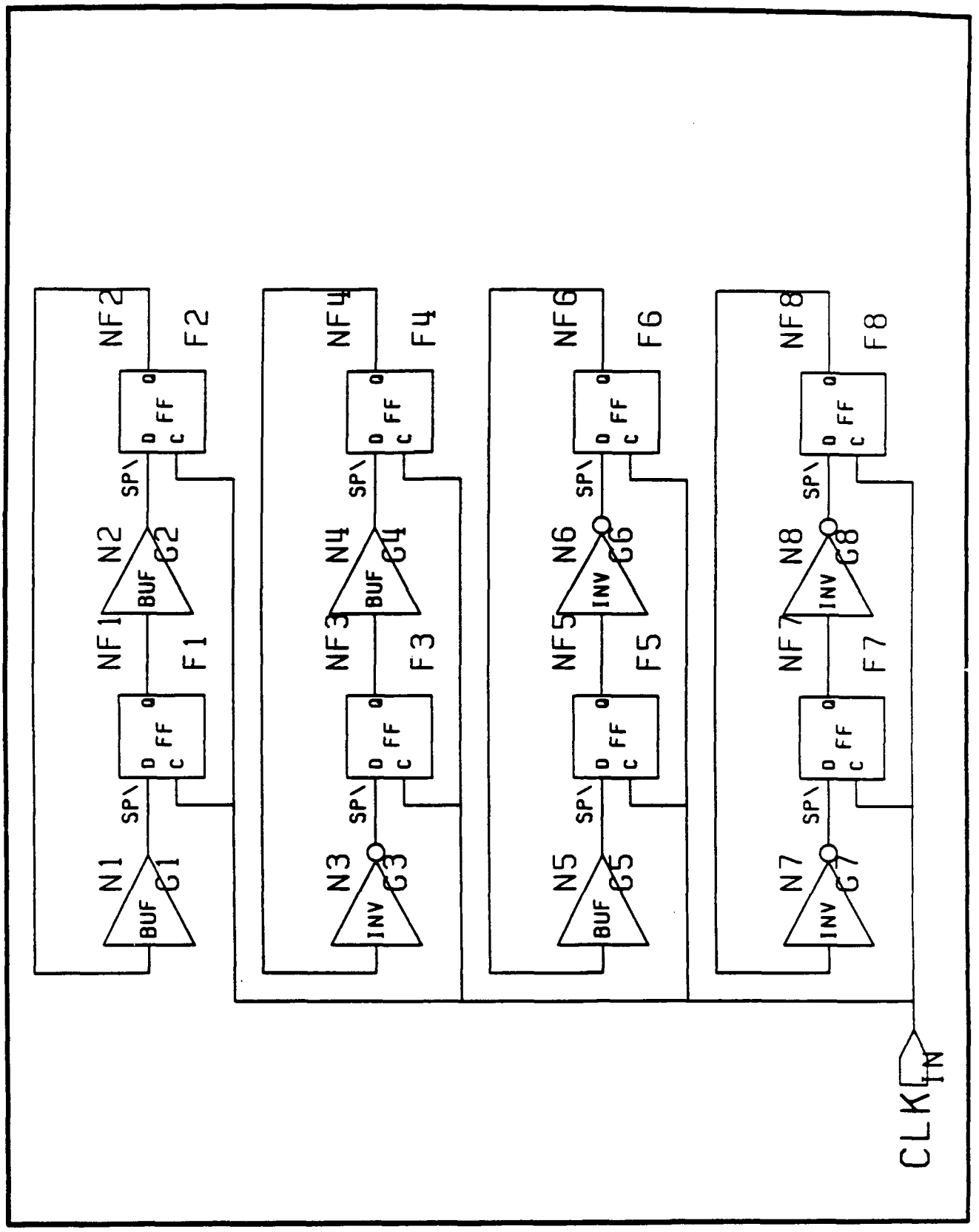

X
buf
G9
BUF
FN2
FF2
SP\
D FF C
N2
G2
AND2INI
N1
G1
AND2
FN1
FF1
N6
G6
OR2
N5
G5
AND2
N7
G7
OR2
N8
G8
OR2
N3
G3
AND2
N4
G4
AND2
N10
G10
INV
A
CLK
B
D
E
F
G

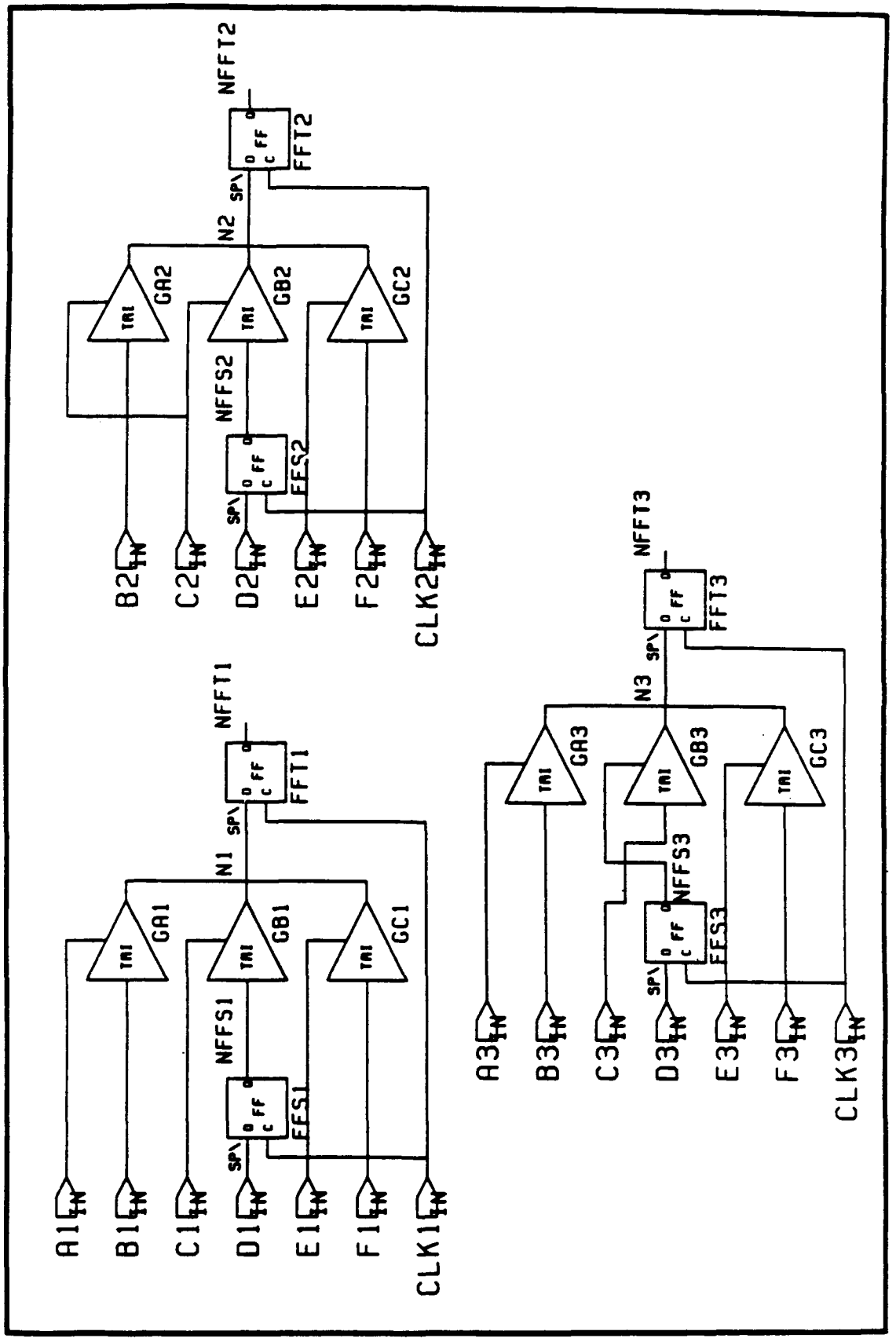

A
B
CLK
D
E
F
X
Y
N1
N2
N3
N4
N5
N6
N8
N9
N10
N11
N12
FN1
FN2
FN3
FN4
FN5
G1
G2
G3
G4
G5
G6
G7
G8
G9
G10
G11
G12
G13
FF1
FF2
FF3
FF4
FF5
BUF
INV
AND2
AND3

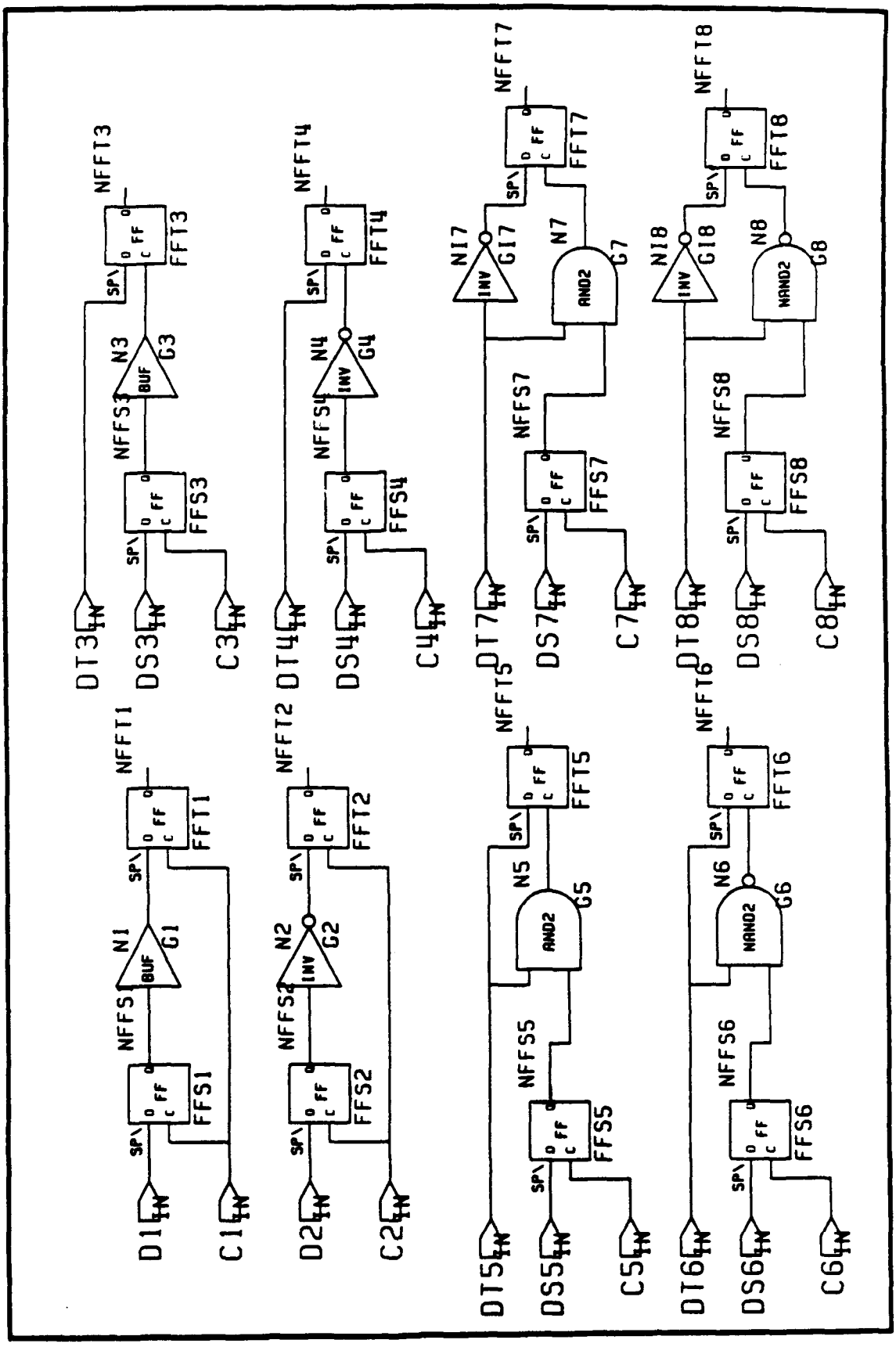

A
B
CLK
D
IN
N1
N2
N3
N4
G1
G2
G3
G4
Buf
FN1
FN2
FN3
FF1
FF2
FF3
ff
SP\
N5
N6
N8
N9
G5
G6
G8
G9
AND3
AND2
INV
X
OUT
G7


A
IN
F1
NF1
N1
G1
BUF
SP\
D
C
FF
Q
B
IN
F2
NF2
N2
G2
INV
C
IN
OR2
X
OUT


CLK IN
N1
G1
BUF
NF1
F1
N2
G2
BUF
NF2
F2
N3
G3
INV
NF3
F3
N4
G4
BUF
NF4
F4
N5
G5
BUF
NF5
F5
N6
G6
INV
NF6
F6
N7
G7
INV
NF7
F7
N8
G8
INV
NF8
F8
SP\
D
C
FF
Q


CLK IN
N1
BUF
G1
NF1
F1
N2
BUF
G2
NF2
F2
N3
INV
G3
NF3
F3
N4
BUF
G4
NF4
F4
N5
BUF
G5
NF5
F5
N6
INV
G6
NF6
F6
N7
INV
G7
NF7
F7
N8
INV
G8
NF8
F8
SP\
D
C
FF
Q


A1
B1
C1
D1
E1
F1
CLK1
A2
B2
C2
D2
E2
F2
CLK2
A3
B3
C3
D3
E3
F3
CLK3
TRI
GA1
GB1
GC1
GA2
GB2
GC2
GA3
GB3
GC3
FF
FFT1
FFT2
FFT3
FFS1
FFS2
FFS3
NFFS1
NFFS2
NFFS3
NFFT1
NFFT2
NFFT3
N1
N2
N3
SP



*U.S. GOVERNMENT PRINTING OFFICE: 1992-612-093-60048