

NASA-CR-190847

SBIR - 06.11-0126
release date 4/18/92 ✓

FINAL REPORT
OF
ERROR DETECTION AND CORRECTION UNIT
WITH
BUILT-IN SELF-TEST CAPABILITY
FOR
SPACECRAFT APPLICATIONS

June 1990

Principal Investigator: Constantin Timoc

Signature: Constantin Timoc

Contract: NAS7-1028

Spaceborne, Inc.
742 Foothill Blvd., Suite 2B
La Canada, CA 91011
(818) 952-0126

(NASA-CR-190847) ERROR DETECTION
AND CORRECTION UNIT WITH BUILT-IN
SELF-TEST CAPABILITY FOR SPACECRAFT
APPLICATIONS Final Report
(Spaceborn) ~~1~~ p

N93-17771

Unclas

373

G3/18 0121170

PROJECT SUMMARY

The objective of this project was to research and develop a 32-bit single chip Error Detection and Correction unit capable of correcting all single bit errors and detecting all double bit errors in the memory systems of a spacecraft.

We designed the 32-bit EDAC (Error Detection and Correction unit) based on a modified Hamming code and according to the design specifications and performance requirements. We constructed a laboratory prototype (breadboard) which was converted into a fault simulator. The correctness of the design was verified on the breadboard using an exhaustive set of test cases. A logic diagram of the EDAC was delivered to JPL Section 514 on October 4, 1988.

The EDAC operates on a 32 bit data. It corrects all single errors and detects all double errors and some triple errors in a memory system. The function of our design is compatible with that of the IDT (49C460) chip and the TI (54ALS632) chip with the exception of expandability to 64 bits. Our design features Byte Writes with separate Byte Enables, a Read Detect Mode, a Pass Thru Mode, and a Built-In Self-Test. In addition, the EDAC deactivates the error signals during Write Mode and can be easily configured to operate on a 16-bit data.

The EDAC circuit is packaged in an 84-pin pin grid array and comprises 32 bi-directional data pins, 7 bi-directional check bit pins, 15 control input pins, and 3 output flags pins. The complete design consists of 2103 equivalent gates (i.e. 4206 transistor pairs). The worst case power dissipation is 300 mW. The maximum propagation delay is approximately 75 ns, well within the specified performance requirements.

We performed stuck-at fault simulation on the complete design and were able to achieve a 100% fault coverage using 380 BIST (Built-In Self-Test) cycles. We also performed stuck-open fault simulation and achieved a fault coverage of 94.8% with the same number of BIST cycles. By increasing the number of BIST cycles to 10000 and by extracting deterministic patterns from the logic diagram, we were able to demonstrate that all of stuck-open faults could be detected.

A critical design review of EDAC was held on June 12, 1989. The design was accepted by the review board members and Spaceborne, Inc. was recommended to proceed with layout and fabrication of the EDAC in silicon.

The layout (placement and routing) of the silicon breadboard was performed on an IBM PS/2 running a layout editor software (L-EDIT) from Tanner Research, Inc. The logic blocks used to implement this silicon breadboard were selected from the standard cell library distributed by MOSIS. During the layout, special attention was paid to the guidelines of a radiation hardened design and to the power distribution inside the chip. The layout of the EDAC was first submitted to MOSIS for fabrication on August 14, 1989. A design rule checking performed at MOSIS identified four types of violations. These errors were corrected and the EDAC was released for fabrication on September 4, 1989.

We received ten silicon breadboard chips of the EDAC from MOSIS in November and tested them immediately. Only eight chips out of the ten chips passed the DC and functional tests. The EDAC chips were delivered to NASA/JPL on November 28, 1989.

TABLE OF CONTENTS

1	EXECUTIVE SUMMARY.....	1
2	INTRODUCTION.....	3
	2.1 Project Goals.....	3
	2.2 Design Specifications.....	3
	2.2.1 Modified Hamming Code.....	3
	2.2.2 Syndrome Decode.....	3
	2.2.3 TCME: Three or More Errors.....	4
	2.2.4 Functional Equations.....	4
	2.3 Performance Requirements.....	4
	2.4 Design Approach.....	4
	2.5 Project Schedule.....	4
3	ERROR DETECTION AND CORRECTION DESIGN.....	5
	3.1 Features.....	5
	3.2 Architecture Summary.....	5
	3.2.1 Data Input Latch.....	6
	3.2.2 Check Bit Input Latch.....	6
	3.2.3 Check Bit Generation Logic.....	6
	3.2.4 Syndrome Generation Logic.....	6
	3.2.5 Error Detection Logic.....	6
	3.2.6 Error Correction Logic.....	7
	3.2.7 Data Output Latch and Output Buffers.....	7
	3.2.8 Check Bit / Syndrome Output Latch.....	7
	3.2.9 ERROR and MULT_ERROR Output Latches.....	7
	3.2.10 Built-in Self-Test Logic.....	7
	3.3 Detailed Product Description.....	8

3.4	Operating Modes.....	8
3.4.1	Write Mode.....	8
3.4.2	Read Detect Mode.....	8
3.4.3	Read Correct Mode.....	9
3.4.4	Pass Thru Mode.....	9
3.4.5	Self-Test Mode.....	9
3.5	32-Bit Data Word Configuration.....	9
3.6	16-Bit Data Word Configuration.....	10
4.	EDAC LABORATORY PROTOTYPE.....	11
4.1	PC-Interface Board.....	11
4.1.1	Address Decoder.....	11
4.1.2	Bus Transceiver.....	11
4.1.3	I/O Ports.....	11
4.1.4	Operation of the PC-Interface Board.....	12
4.2	Breadboard Description.....	12
4.3	Design Verification.....	12
4.3.1	Description of Test Case Programs.....	12
4.3.1.1	Check Bit Generator (PT-TEST.C).....	13
4.3.1.2	Error Detector (ED-TEST.C).....	13
4.3.1.3	Error Corrector (EC-TEST.C).....	13
4.3.1.4	Random Pattern (EDAC.C).....	13
4.4	Functional Compatibility.....	14
4.4.1	Integrated Device Technology (49C460) Chip...14	
4.4.2	Texas Instruments (SN 74ALS632) Chip.....14	
4.4.3	Results of Functional Comparison.....14	
4.5	Fault Simulation.....	15
4.5.1	Microsimulators.....	15

4.5.2	Fault Simulator Breadboard.....	15
4.5.3	Stuck-at Fault Simulation.....	16
4.5.3.1	Stuck-at Fault Mechanism.....	17
4.5.3.2	Stuck-at Fault Simulation Hardware.....	17
4.5.3.3	Description of the Stuck-at Fault Simulation.....	17
4.5.3.4	Results of the Stuck-at Fault Simulation.....	18
4.5.4	Stuck-open Fault Simulation.....	18
4.5.4.1	Stuck-open Fault Mechanism.....	18
4.5.4.2	Stuck-open Fault Simulation Hardware....	19
4.5.4.3	Description of the Stuck-open Fault Simulation.....	19
4.5.4.4	Results of the Stuck-open Fault Simulation.....	20
5	SILICON BREADBOARD.....	21
5.1	Schematic Diagram.....	21
5.2	Layout.....	21
5.2.1	Problems During Layout.....	21
5.2.2	Radiation Hardened Design.....	21
5.2.3	Power Distribution.....	21
5.3	Layout Verification.....	22
5.4	Fabrication of the Silicon Breadboards.....	22
5.5	Testing of the Silicon Breadboards.....	22
5.5.1	Random Test.....	22
5.5.2	Functional Test.....	23
5.5.3	Built-In Self-Test.....	23
5.5.4	Quiescent Current.....	23

5.5.5	Propagation Delay.....	24
5.6	Summary of EDAC Features.....	24
5.7	Delivery to NASA/JPL.....	24

APPENDIXES

A INTRODUCTION

A.1 Design Specifications

A.1.1 Modified Hamming Code.....A-1

A.1.2 Syndrome Decode.....A-2

A.1.3 TOME: Three Or More Errors.....A-3

A.1.4 Logic Equations.....A-4

A.2 Performance Requirements.....A-5

A.3 Design Approach

A.3.1 Figure 1.....A-6

A.3.2 Figure 2.....A-7

A.3.3 Figure 3.....A-8

A.3.4 Figure 4.....A-9

A.3.5 Figure 5.....A-10

A.4 SBIR Project Schedule.....A-11

B ERROR DETECTION AND CORRECTION DESIGN

B.1 EDAC Block Diagram

B.1.1 Map of EDAC Block Diagram.....B-1

B.1.2 Figure 1.....B-2

B.1.3 Figure 2.....B-3

B.1.4 Figure 3.....B-4

B.1.5 Figure 4.....B-5

B.1.6 Figure 5.....B-6

B.1.7 Figure 6.....B-7

B.1.8 Figure 7.....B-8

B.1.9 Figure 8.....B-9

B.1.10	Figure 9.....	B-10
B.2	Built-In Self-Test Block Diagram.....	B-11
B.3	EDAC Operating Modes.....	B-12
B.4	Switching Waveforms	
B.4.1	Write Mode.....	B-13
B.4.2	Read Mode.....	B-14
B.4.3	Self Test Mode.....	B-15
B.5	BIST Test Vectors.....	B-16
B.6	16-Bit Configuration.....	B-17
C	FAULT SIMULATOR BREADBOARD	
C.1	Logic Diagram of Fault Simulator Breadboard	
C.1.1	Map of Fault Simulator Breadboard.....	C-1
C.1.2	Figure 1.....	C-2
C.1.3	Figure 2.....	C-3
C.1.4	Figure 3.....	C-4
C.1.5	Figure 4.....	C-5
C.1.6	Figure 5.....	C-6
C.1.7	Figure 6.....	C-7
C.1.8	Figure 7.....	C-8
C.1.9	Figure 8.....	C-9
C.1.10	Figure 9.....	C-10
C.1.11	Figure 10.....	C-11
C.1.12	Figure 11.....	C-12
C.1.13	Figure 12.....	C-13
C.1.14	Figure 13.....	C-14
C.1.15	Figure 14.....	C-15
C.1.16	Figure 15.....	C-16

C.1.17	Figure 16.....	C-17
C.1.18	Figure 17.....	C-18
C.1.19	Figure 18.....	C-19
C.1.20	Figure 19.....	C-20
C.1.21	Figure 20.....	C-21
C.1.22	Figure 21.....	C-22
C.1.23	Figure 22.....	C-23
C.1.24	Figure 23.....	C-24
C.1.25	Figure 24.....	C-25
C.1.26	Figure 25.....	C-26
C.1.27	Figure 26.....	C-27
C.1.28	Figure 27.....	C-28
C.1.29	Figure 28.....	C-29
C.1.30	Figure 29.....	C-30
C.1.31	Figure 30.....	C-31
C.1.32	Figure 31.....	C-32
C.1.33	Figure 32.....	C-33
C.1.34	Figure 33.....	C-34
C.1.35	Figure 34.....	C-35
C.1.36	Figure 35.....	C-36
C.1.37	Figure 36.....	C-37
C.1.38	Figure 37.....	C-38
C.1.39	Figure 38.....	C-39
C.1.40	Figure 39.....	C-40
C.1.41	Figure 40.....	C-41
C.1.42	Figure 41.....	C-42
C.1.43	Figure 42.....	C-43

C.1.44	Figure 43.....	C-44
C.1.45	Figure 44.....	C-45
C.1.46	Figure 45.....	C-46
C.1.47	Figure 46.....	C-47
C.1.48	Figure 47.....	C-48
C.1.49	Figure 48.....	C-49
C.1.50	Figure 49.....	C-50
C.1.51	Figure 50.....	C-51
C.1.52	Figure 51.....	C-52
C.1.53	Figure 52.....	C-53
C.1.54	Figure 53.....	C-54
C.1.55	Figure 54.....	C-55
C.1.56	Figure 55.....	C-56
C.1.57	Figure 56.....	C-57
C.1.58	Figure 57.....	C-58
C.1.59	Figure 58.....	C-59
C.1.60	Figure 59.....	C-60
C.1.61	Figure 60.....	C-61
C.2	Detailed Diagram of XOR/XNOR Gates.....	C-62
C.2.1	Position of XOR Gates.....	C-63
C.2.2	Position of XNOR Gates.....	C-67
C.3	Detailed Diagram of Scannable Latches.....	C-68
C.3.1	Position of Scannable Latches.....	C-69
C.4	Connector Assignment.....	C-72
C.5	Pin Assignment of CMOS chips.....	C-73
C.6	Compatibility	
C.6.1	Integrated Devices Technology (IDT 49C460) Chip	
C.6.1.1	Map of IDT 49C460 Adapter.....	C-74

C.6.1.2	Figure 1.....	C-75
C.6.1.3	Figure 2.....	C-76
C.6.2	Texas Instruments (SN 74ALS632) Chip	
C.6.2.1	Map of SN 74ALS632 Adapter.....	C-77
C.6.2.2	Figure 1.....	C-78
C.6.2.3	Figure 2.....	C-79
C.7	Results of Design Verification	
C.7.1	Parity Tree Network.....	C-80
C.7.2	Error Detector.....	C-81
C.7.3	Error Locator.....	C-82
C.7.4	Random Test.....	C-83

D FAULT SIMULATION

D.1 Fault Simulator Breadboard Layout

D.1.1	Map of Fault Simulator Breadboard Layout....	D-1
D.1.2	Figure 1.....	D-2
D.1.3	Figure 2.....	D-3
D.1.4	Figure 3.....	D-4
D.1.5	Figure 4.....	D-5
D.1.6	Figure 5.....	D-6
D.1.7	Figure 6.....	D-7
D.1.8	Figure 7.....	D-8
D.1.9	Figure 8.....	D-9
D.1.10	Figure 9.....	D-10
D.1.11	Figure 10.....	D-11
D.1.12	Figure 11.....	D-12
D.1.13	Figure 12.....	D-13

D.1.14	Figure 13.....	D-14
D.1.15	Figure 14.....	D-15
D.1.16	Figure 15.....	D-16
D.1.17	Figure 16.....	D-17
D.1.18	Figure 17.....	D-18
D.1.19	Figure 18.....	D-19
D.1.20	Figure 19.....	D-20
D.1.21	Figure 20.....	D-21
D.1.22	Figure 21.....	D-22
D.1.23	Figure 22.....	D-23
D.1.24	Figure 23.....	D-24
D.1.25	Figure 24.....	D-25
D.2	Microsimulator	
D.2.1	SP4HFS011: 2-Input NAND Gates.....	D-26
D.2.2	SP4HFS023: 3-Input NAND Gates.....	D-29
D.2.3	SP4HFS071: 2-Input OR Gates.....	D-32
D.2.4	SP4HFS081: 2-Input AND Gates.....	D-35
D.3	Fault Simulation	
D.3.1	Combinational Logic.....	D-38
D.3.2	Scannable Latches.....	D-39
D.3.3	Results of Stuck-at Fault Simulation	
D.3.3.1	Combinational Logic: 10 cycles.....	D-40
D.3.3.2	Combinational Logic: 20 cycles.....	D-41
D.3.3.3	Combinational Logic: 100 cycles.....	D-42
D.3.3.4	Combinational Logic: 300 cycles.....	D-43
D.3.3.5	Combinational Logic: 400 cycles.....	D-44
D.3.3.6	Scannable Latches: 2 cycles.....	D-45

D.3.3.7	Scannable Latches: 4 cycles.....	D-46
D.3.3.8	Scannable Latches: 6 cycles.....	D-47
D.3.4	Results of Stuck-open Fault Simulation	
D.3.4.1	Combinational Logic: 10 cycles.....	D-48
D.3.4.2	Combinational Logic: 100 cycles.....	D-49
D.3.4.3	Combinational Logic: 400 cycles.....	D-50
D.3.4.4	Combinational Logic: 1000 cycles.....	D-51
D.3.4.5	Combinational Logic: 10000 cycles.....	D-52
D.3.4.6	Combinational Logic: 100000 cycles.....	D-53

E SILICON BREADBOARD

E.1 Logic Diagram of Silicon Breadboard

E.1.1	Map of Fault Simulator Breadboard.....	E-1
E.1.2	Figure 1.....	E-2
E.1.3	Figure 2.....	E-3
E.1.4	Figure 3.....	E-4
E.1.5	Figure 4.....	E-5
E.1.6	Figure 5.....	E-6
E.1.7	Figure 6.....	E-7
E.1.8	Figure 7.....	E-8
E.1.9	Figure 8.....	E-9
E.1.10	Figure 9.....	E-10
E.1.11	Figure 10.....	E-11
E.1.12	Figure 11.....	E-12
E.1.13	Figure 12.....	E-13
E.1.14	Figure 13.....	E-14
E.1.15	Figure 14.....	E-15
E.1.16	Figure 15.....	E-16

E.1.17	Figure 16.....	E-17
E.1.18	Figure 17.....	E-18
E.1.19	Figure 18.....	E-19
E.1.20	Figure 19.....	E-20
E.1.21	Figure 20.....	E-21
E.1.22	Figure 21.....	E-22
E.1.23	Figure 22.....	E-23
E.1.24	Figure 23.....	E-24
E.1.25	Figure 24.....	E-25
E.1.26	Figure 25.....	E-26
E.1.27	Figure 26.....	E-27
E.1.28	Figure 27.....	E-28
E.1.29	Figure 28.....	E-29
E.1.30	Figure 29.....	E-30
E.1.31	Figure 30.....	E-31
E.1.32	Figure 31.....	E-32
E.1.33	Figure 32.....	E-33
E.1.34	Figure 33.....	E-34
E.1.35	Figure 34.....	E-35
E.1.36	Figure 35.....	E-36
E.1.37	Figure 36.....	E-37
E.1.38	Figure 37.....	E-38
E.1.39	Figure 38.....	E-39
E.1.40	Figure 39.....	E-40
E.1.41	Figure 40.....	E-41
E.1.42	Figure 41.....	E-42
E.1.43	Figure 42.....	E-43

E.1.44	Figure 43.....	E-44
E.1.45	Figure 44.....	E-45
E.1.46	Figure 45.....	E-46
E.1.47	Figure 46.....	E-47
E.1.48	Figure 47.....	E-48
E.1.49	Figure 48.....	E-49
E.1.50	Figure 49.....	E-50
E.1.51	Figure 50.....	E-51
E.1.52	Figure 51.....	E-52
E.1.53	Figure 52.....	E-53
E.1.54	Figure 53.....	E-54
E.1.55	Figure 54.....	E-55
E.1.56	Figure 55.....	E-56
E.1.57	Figure 56.....	E-57
E.1.58	Figure 57.....	E-58
E.1.59	Figure 58.....	E-59
E.1.60	Figure 59.....	E-60
E.1.61	Figure 60.....	E-61
E.2	Pin Assignment.....	E-62
E.3	Pin Description.....	E-63
E.4	Propagation Delays.....	E-65

F PROGRAM LISTINGS

F.1 Design Verification

F.1.1	EDAC.C: Random Test.....	F-1
F.1.2	PT-TEST.C: Parity Tree Test.....	F-18
F.1.3	ED-TEST.C: Error Detector Test.....	F-24
F.1.4	EC-TEST.C: Error Corrector Test.....	F-28

F.2 Fault Simulation

F.2.1 FAULT.C: Combinational Logic.....F-32
F.2.2 FAULT1.C: Scannable Latches.....F-48

G PC-INTERFACE BOARD

G.1 Logic Diagram of PC-Interface Board

G.1.1 Map of PC-Interface Board.....G-1
G.1.2 Figure 1.....G-2
G.1.3 Figure 2.....G-3
G.1.4 Figure 3.....G-4
G.1.5 Figure 4.....G-5
G.1.6 Figure 5.....G-6
G.1.7 Figure 6.....G-7
G.1.8 Figure 7.....G-8
G.1.9 Figure 8.....G-9
G.1.10 Figure 9.....G-10
G.1.11 Figure 10.....G-11
G.1.12 Figure 11.....G-12
G.1.13 Figure 12.....G-13
G.1.14 Figure 13.....G-14
G.1.15 Figure 14.....G-15
G.1.16 Figure 15.....G-16
G.1.17 Figure 16.....G-17
G.1.18 Figure 17.....G-18
G.1.19 Figure 18.....G-19
G.1.20 Figure 19.....G-20
G.1.21 Figure 20.....G-21

1 EXECUTIVE SUMMARY

The objective of this project was to research and develop a 32-bit single chip Error Detection and Correction unit capable of correcting all single bit errors and detecting all double bit errors in the memory systems of a spacecraft.

We designed the 32-bit EDAC (Error Detection and Correction unit) based on a modified Hamming code and according to the design specifications and performance requirements. We constructed a laboratory prototype (breadboard) which was converted into a fault simulator. The correctness of the design was verified on the breadboard using an exhaustive set of test cases. A logic diagram of the EDAC was delivered to JPL Section 514 on October 4, 1988.

The EDAC operates on a 32 bit data. It corrects all single errors and detects all double errors and some triple errors in a memory system. The function of our design is compatible with that of the IDT (49C460) chip and the TI (54ALS632) chip with the exception of expandability to 64 bits. Our design features Byte Writes with separate Byte Enables, a Read Detect Mode, a Pass Thru Mode, and a Built-In Self-Test. In addition, the EDAC deactivates the error signals during Write Mode and can be easily configured to operate on a 16-bit data.

The EDAC circuit is packaged in an 84-pin pin grid array and comprises 32 bi-directional data pins, 7 bi-directional check bit pins, 15 control input pins, and 3 output flags pins. The complete design consists of 2103 equivalent gates (i.e. 4206 transistor pairs). The worst case power dissipation is 300 mW. The maximum propagation delay is approximately 76 ns, well within the specified performance requirements.

We performed stuck-at fault simulation on the complete design and were able to achieve a 100% fault coverage using 380 BIST (Built-In Self-Test) cycles. We also performed stuck-open fault simulation and achieved a fault coverage of 94.8% with the same number of BIST cycles. By increasing the number of BIST cycles to 10000 and by extracting deterministic patterns from the logic diagram, we were able to demonstrate that all of stuck-open faults could be detected.

A critical design review of EDAC was held on June 12, 1989. The design was accepted by the review board members and Spaceborne, Inc. was recommended to proceed with layout and fabrication of the EDAC in silicon.

The layout (placement and routing) of the silicon breadboard was performed on an IBM PS/2 running a layout editor software (L-EDIT) from Tanner Research, Inc. The logic blocks used to implement this silicon breadboard were selected from the standard cell library distributed by MOSIS. During the layout, special attention was paid to the guidelines of a radiation hardened design and to the power distribution inside the chip. The layout

of the EDAC was first submitted to MOSIS for fabrication on August 14, 1989. A design rule checking performed at MOSIS identified four types of violations. These errors were corrected and the EDAC was released for fabrication on September 4, 1989.

We received ten silicon breadboard chips of the EDAC from MOSIS in November and tested them immediately. Only eight chips out of the ten chips passed the DC and functional tests. The EDAC chips were delivered to NASA/JPL on November 28, 1989.

2 INTRODUCTION

2.1 Project Goals

The main objective of this project was the research and development of a 32-bit single chip Error Detection and Correction unit capable of correcting all single bit errors and detecting all double bit errors in memory systems of a spacecraft. The development efforts include:

1. Design of an Error Detection and Correction unit according to specifications and performance requirements
2. Development of a laboratory prototype
3. Verification of the laboratory prototype
4. Fault simulation with at least a 95% fault coverage
5. Layout of a silicon breadboard
6. Fabrication of the silicon breadboard
7. Testing of the silicon breadboard
8. Delivery of the silicon breadboard to NASA/JPL

2.2 Design Specifications

2.2.1 Modified Hamming Code

The EDAC (Error Detection and Correction unit) generates check bits on a 32-bit data field according to a modified Hamming code. The modified Hamming code table on page A-1 in Appendix A indicates the data bits participating in generating each of the check bits. For example, check bit C₀ is the Exclusive-OR function of the 13 data input bits marked with an "X". The modified Hamming code is designed such that the EDAC can be easily configured for a 16-bit operation. This is achieved through a design in which the 16 least significant data bits of the 32-bit Hamming code affect only the 6 least significant check bits.

2.2.2 Syndrome Decode

The Syndrome Decode table on page A-2 in Appendix A shows the complete decoding of the seven syndrome bits. A 7-bit syndrome may indicate: the location of the single bit-in-error, or the detection of double bit errors, or the detection of triple bit errors. The exception is the all zero combination which is used to indicate no error detected.

2.2.3 TOME: Three or more Errors

The TOME table on page A-3 in Appendix A shows all cases of the 7-bit syndrome that indicates three or more errors. It is used to generate the functional equation for the TOME output signal.

2.2.4 Functional Equations

The functional equations for the generation of the error signals are shown on page A-4 in Appendix A. These equations were used to implement the error detection logic.

2.3 Performance Requirements

A summary of the performance requirements for the EDAC silicon breadboard is listed on page A-5 in Appendix A.

2.4 Design Approach

A detailed summary of the design approach is shown on pages A-6 through A-10 in Appendix A.

2.5 Project Schedule

The SBIR project schedule as of 28 November 1989 is shown on page A-11 and page A-12 in Appendix A.

3 ERROR DETECTION AND CORRECTION DESIGN

The EDAC is a 32-bit single chip error detection and correction unit. The EDAC generates check bits on a 32-bit data field according to a modified Hamming code and corrects the data word when check bits are supplied. It corrects all single errors and detects all double errors and some triple errors in a memory system. The function of our design is compatible with that of the IDT (49C460) chip and the TI (54ALS632) chip with the exception of expandability to 64 bits. The design features Byte Writes with separate Byte Enables, a Read Detect Mode, a Pass Thru Mode, and a Built-In Self-Test. The circuit also deactivates the error signals during Write Mode and can be easily configured to operate on a 16-bit data. A 32-bit system uses 7 check bits while a 16-bit system uses 6 check bits. In either configurations, the error syndrome is made available. The EDAC also incorporates a Built-in-self-test logic to perform self-testing. This self-test mode simplifies testing and allows a user to determine if the EDAC is failing. A block diagram of the EDAC Design is shown in Figures 1 through 9 of Appendix B.

3.1 Features

- 32 input data bits, 7 check bits
- Corrects all single data or check bit errors, detects all double and some triple bit errors
- Byte Write with separate Byte Enables
- Deactivates "ERROR" and "MULT_ERROR" signals during writing operation
- Read Detect Mode (Data is not corrected in Read Detect Mode)
- Pass Thru Mode
- Built-In-Self-Test

3.2 Architecture Summary

The EDAC is a high-performance device used for check bits generation, error detection, error correction and diagnostics. The functional blocks of this 32-bit device are:

- Data Input Latch
- Check Bit Input Latch
- Check Bit Generation Logic
- Syndrome Generation Logic

- Error Detection Logic
- Error Correction Logic
- Data Output Latch
- Check Bit / Syndrome Output Latch
- Error Output Latch
- Built-In Self-Test

3.2.1 Data Input Latch

The Data Latch Enable Input, LE DATA IN, controls the loading of a 32-bit input data into the Data Input Latch.

3.2.2 Check Bit Input Latch

Seven check bits are loaded into this latch under the control of LE DATA IN. Check bits are used in the Error Detection and Error Correction modes and in the Self-Test mode.

3.2.3 Check Bit Generation Logic

This logic generates the appropriate check bits for the 32-bit data stored in the Data Input Latch. A modified Hamming Code is the basis for generating the proper check bits.

3.2.4 Syndrome Generation Logic

In both the Detect and Correct modes, this logic compares the check bits read from memory against the internally generated check bits produced from the input data read from memory. Matching sets of check bits means no error is detected. If there is a mismatch, then one or more of the data or check bits is in error. Each Syndrome bits is produced by an Exclusive-OR of the corresponding bits of the two sets of check bits. Identical sets of check bits produces all zeroes syndrome bits. Otherwise, this 7-bit syndrome must be decoded to determine the number of errors and the specific bit-in-error in the case of a single error.

3.2.5 Error Detection Logic

This part of the device decodes the syndrome bits generated by the Syndrome Generation Logic. With no errors in either the input data or the check bits, both the ERROR and MULT_ERROR output signals are high. The ERROR signal will go low if a single error is detected. The MULT_ERROR and ERROR signals will both go low if

two or more errors are detected.

3.2.6 Error Correction Logic

In single bit error cases, this logic complements (corrects) the single data bit-in-error. This corrected data is loaded into the Data Output Latch, which can then be read onto the bi-directional data lines. If the single bit error is in one of the check bits, the corrected check bits are not placed on the check bit / syndrome outputs. To generate these corrected check bits, the EDAC must be switched to the Write Mode.

3.2.7 Data Output Latch and Output Buffers

The result of the error correction logic is loaded into Data Output Latch under the control of the Data Output Latch Enable, LE DATA OUT. The Data Output Latch buffer is split into 4 independent 8-bit buffers each enabled by -OE BYTE 0-3 respectively. These buffers are used to place data onto the bi-directional data lines.

3.2.8 Check Bit / Syndrome Output Latch

The Check Bit / Syndrome Output Latch is used for storing the syndrome generated during a read operation. The latch is loaded from the Syndrome Generation Logic under the control of the Syndrome / Check Bit Output Latch Enable, LE SC. The Check Bit / Syndrome Output Latch buffer is enabled by -OE CB/SY for writing data onto the bi-directional Check Bit / Syndrome lines.

3.2.9 ERROR and MULT_ERROR Output Latches

The ERROR and MULT_ERROR Output Latches are used for storing the ERROR and MULT_ERROR signals generated by the Error Detection Logic. These latches are controlled by the ERROR and MULT_ERROR Output Latch Enable signals LE ERROR and LE MULT_ERROR respectively. The ERROR and MULT_ERROR signals can be made asynchronous by connecting their corresponding Latch Enable lines to VDD (latches are transparent).

3.2.10 Built-In Self-Test Logic

The block diagram for the built-in-self-test structure is shown on page B-11 of Appendix B. In Self-Test Mode, this logic performs a self-testing of the complete EDAC circuit. After initialization, a number of non-overlapping clock pulses (CLK_A, CLK_B) are applied and a signature comprising 32 data bits, 7 check bits, and 2 error bits is generated. This signature is available at the output latches and can be compared to the so called "Good Machine Signature". Matching signatures indicates

that no fault is detected in the EDAC circuit.

3.3 Detailed Product Description

The EDAC unit contains the logic necessary to generate check bits on a 32-bit input data according to a modified Hamming Code. The EDAC compares a set of internally generated check bits against those read from memory to allow correction of any single bit data error and detection of all double bit and some triple bit errors. The EDAC can be used with either 32-bit data words (7 check bits) or 16-bit data words (6 check bits).

The EDAC provides either check bits or syndrome bits on the tri-stated output pins, CB/SY 0-7. Each check bit is generated from a combination of input data bits, while each syndrome is produced by comparing internally generated check bits with those read from memory. Decoded syndrome bits may indicate one of the following: the location of a single bit-in-error or the detection of a double/triple bits error. The check bits are labeled:

CB0, CB1, CB2, CB3, CB4, CB5, CB6 for the 32-bit configuration

CB0, CB1, CB2, CB3, CB4, CB5 for the 16-bit configuration

Syndrome bits are similarly labeled S0 through S6 and S0 through S5 respectively.

3.4 Operating Modes

A table on page B-12 of Appendix B lists the operating modes of the EDAC. These operating modes are defined by the R/-W and the CORRECT signals. The corresponding waveforms of Write mode, Read mode, and Self Test mode are shown on Page B-13, B-14, and B-15 respectively of Appendix B.

3.4.1 Write Mode

Write Mode is used to generate the check bits of a 32-bit input data. The data bits are latched into the data input latches and the generated check bits are latched into the CB/SY output latches. After enabling the three-state output buffer, the check bits are available on the bi-directional CB/SY bus. During Write Mode the error signals ERROR and MULT_ERROR remain inactive (high). The waveforms of this mode of operation are illustrated on page B-13 of Appendix B.

3.4.2 Read Detect Mode

In Read Mode, both input data and the corresponding check bits are latched into the input latches. The waveforms are shown on page

B-14 of Appendix B. The syndrome is generated by bit-wise comparisons of the check bits read from memory with the check bits produced from the 32 data bits. This syndrome is decoded to generate the error signals ERROR and MULT_ERROR. In Read Detect mode, the 32 data bits remain unchanged. The syndrome bits are provided on the CB/SY outputs.

3.4.3 Read Correct Mode

The Read Correct mode is similar to the Read Detect mode except that single bit errors will be complemented (corrected) and made available to the Data Output Latches. In this mode of operation, double errors may produce meaningless outputs at the Data Output Latches.

3.4.4 Pass Thru Mode

The Pass Thru Mode can be used in both, Read and Write Mode. Data latched into the Data Input Latches remains unchanged. The CB/SY Output Latches contain the generated check bits in the case of Write Mode and the syndrome in the case of Read Mode.

3.4.5 Self-Test Mode

The Self-Test Mode switching waveforms are shown on page B-15 of Appendix B. The Self-Test Mode is initiated by writing the seed according to the table on page B-16 of Appendix B. The seed consists of 32 data bits and 7 check bits. Data and check bits in this table are in hexadecimal format. The seed is loaded into the input latches.

The applied data propagates through the combinational logic and is then loaded into the output latches. In Self-Test mode, these output latches form the signature register. At this point the number of cycles is zero. The Self-Test procedure is started by setting LE R/-W to low and by applying the non-overlapping clocks CLK_A and CLK_B starting with CLK_B. One "shift", as defined in the waveforms illustration, is a sequence of one CLK_B and one CLK_A pulse. One cycle consists of 41 shifts. When a cycle is completed, the generated data is latched into the signature register. After a certain number of cycles has elapsed, the signature is read and compared with the good machine signature. If both signatures are identical, no fault is detected. A large number of possible faults can be detected with the first three test vectors shown in the table. However, some faults are more difficult to detect and require special sequences. Most of these sequences consist of a seed for initialization and a second seed to generate the signature. With the test vectors shown in the table, a 100% stuck-at fault coverage can be accomplished.

3.5 32-Bit Data Word Configuration

The 32-bit format of the EDAC unit consists of 32 Data bits and 7 Check bits. It is referred to as 32/39 code. A single EDAC unit provides all the logic needed for single bit error correction and double bit error detection of a 32-bit data field.

The table in page A-1 of Appendix A indicates the data bits participating in the check bit generation. For example, check bit CB0 is the exclusive-OR function of the 13 data inputs marked with an "X". Check bits are generated and output in the Write Mode.

Syndrome bits are generated by an Exclusive-OR (XOR) of the generated check bits with the read check bits. For example, S0 is the XOR of check bits CB0 from those read with those generated. The table in page A-2 of Appendix A indicates the decoding of the seven syndrome bits to identify the location of bit-in-error for a single bit error or the detection of a double or triple bit error. The all zero case indicates no error detected.

In the Correct Mode, the syndrome bits are used to complement (correct) single bit errors in the data bits. For double or multiple error detection, the data latched into the Data Output Latch is not defined.

3.6 16-Bit Data Word Configuration

The 16-bit format of the EDAC unit consists of 16 Data bits and 6 Check bits. It is referred to as 16/22 code.

An EDAC unit can be configured for single bit error correction and double bit error detection of a 16-bit data field. The necessary connections is shown on page B-17 of Appendix B. Only the least significant 16 bits are used as data input, the 16 most significant data inputs are grounded like the seventh check bit input. The Output Enable lines for the most significant two data bytes (-OE BYTE 2,3) are disabled (tied to VDD). The operation in 16-bit mode is the same as in 32-bit mode, except that only 16 data and 6 check/syndrome bits are used.

4 EDAC LABORATORY PROTOTYPE

4.1 PC-Interface Board

This PC-Interface board is designed to allow an IBM PC/AT to communicate with the 32-bit EDAC prototype board. It consists of an Address Decoder, a Bus Transceiver, and Input/Output Ports. The complete logic diagram of the PC interface board is shown in figures 1 through 20 of Appendix G. Five connectors labeled C1 through C5 are used to interface the board with the EDAC. All connecting wires between the EDAC unit and the interface board are twisted pair cables to minimize noise.

4.1.1 Address Decoder

The Address Decoder receives the 10 least significant bits address of the IBM PC and the +AEN signal. It generates the -CE signal when the +AEN signal is "low" and the 10-bit address is in the range of 300H-31FH. This -CE signal in turn activates the Bus Transceiver. Together with the lower 5 bits of the address, it selects one of the 32 available ports in the above stated range.

4.1.2 Bus Transceiver

The Bus Transceiver (74HCT245) is enabled by the Address Decoder and the direction of the data flow is determined by the -IOR signal of the IBM PC. When -IOR is "low", the Bus Transceiver allows data to be transferred from a selected port of the interface card to the IBM PC's data bus. When -IOR is "high" data flows in the opposite direction. The 74HCT245 transceiver buffers data in groups of 8 bits (i.e. a byte) and converts the TTL level of the IBM PC to the CMOS level used in the prototype EDAC unit.

4.1.3 I/O Ports

Each output port consists of a 74HC373 chip while each input port consists of a 74HC374 chip. The open drain output ports are designed with 74HC74 chips and NMOS transistors. Altogether, there are 16 ports each providing up to 8 data lines. In the interface card, these ports are configured as:

- 96 bi-directional data lines
- 16 input data lines
- 16 output data lines
- 2 open drain output data lines

Each data line is mapped to the input and output signals of the 32-bit EDAC prototype unit. The two available open drain output data lines and the rest of unused data lines are reserved for

later use with Spaceborne, Inc.'s microsimulator chips.

4.1.4 Operation of the PC-Interface Board

The interface card supplies the necessary control signals, data, and check bits to the 32-bit EDAC laboratory prototype as specified by a test program. It also receives the appropriate data, check/syndrome bits, and error signals from the prototype board to be read by the test program.

The control signals to the EDAC prototype board are send out through ports 304H and 305H. The Error signals from the prototype 32-bit EDAC unit are received through port 300H.

The 32 data bits and 7 check bits are transferred via bi-directional data lines. The direction of the data flow on the 96 bi-directional data lines is controlled by bit 0 of port 305H for ports 1 through 3, and by bit 7 of port 305H for ports 6 through 8. The transfer of data from the EDAC prototype board to the interface card is controlled by port 313H.

4.2 EDAC Breadboard Description

The EDAC laboratory prototype was implemented on a wire-wrap board. All the interconnections were made based on the netlist produced from the logic diagram of the EDAC laboratory prototype. The schematic diagram of the laboratory prototype is shown in Appendix C, figures 1 through 60. The prototype board is designed with 4000 series CMOS chips which can be replaced by their functional counterpart Spaceborne, Inc.'s microsimulator chips for fault simulation. The EDAC prototype board is controlled by an IBM PC/AT which communicates with the prototype board through an interface card as described in 4.1.

4.3 Design Verification

To perform a thorough functional verification of the EDAC design, several test programs have been developed. The design was divided into several functional units and each unit was tested separately. This method called "sensitized partitioning" enabled us to exhaustively test each functional block (i.e. check bit generator, error detector, and error corrector) contained in the EDAC. We applied an exhaustive set of test vectors to each of the functional blocks and compared the circuit's response with an expected set of responses derived from the specifications of the EDAC. To verify the function of the complete design, another test program was used to apply random data to the EDAC. With over 1 million test vectors applied, the EDAC breadboard passed the test without any discrepancy.

4.3.1 Description of Test Case Programs

The test case programs were developed in the "C" language under the DOS operating system. A listing of these test case programs is provided in Appendix F. Printouts of the verification results for each individual functional units are listed on pages C-79 through C-82 in Appendix C.

4.3.1.1 Check Bit Generator (PT-TEST.C)

The check bit generator comprises seven parity networks each with either 13 or 14 inputs. Each parity network generates one check bit. In order to complete an exhaustive test of the checkbit generator in a reasonable amount of time, only one parity network is exhaustively tested at a time. The test program generates and applies an exhaustive set of vectors to each parity network. The resulting check bits are compared with expected check bits according to the design specifications. In the case of a disagreement, an error message is produced. The listing of this program can be found on pages F-18 through F-23 of Appendix F.

4.3.1.2 Error Detector (ED-TEST.C)

The error detector unit detects single and multiple errors and generates the syndrome and the error signals ERROR and MULT ERROR. In order to exhaustively verify the function of the error detector, the circuit is sensitized so that every possible syndromes is generated. The resulting error signals are compared with expected error signals of a correct design. This program is listed on pages F-24 through F-27 of Appendix F.

4.3.1.3 Error Corrector (EC-TEST.C)

The error corrector block corrects a faulty bit in the case of a single error. To verify this block, an exhaustive set of test vectors simulating all possible single errors is applied to the circuit. The corrected data bits are examined to make sure that only the faulty bit has been toggled. The listing of this program is included on pages F-28 through F-31 of Appendix F.

4.3.1.4 Random Pattern (EDAC.C)

This program is designed to work with our EDAC design and with two commercial EDAC chips. (i.e., SN 74ALS632 chip from Texas Instruments and IDT 49C460 chip from Integrated Device Technology). This program enables one to write any data to the EDAC and to display the response of the EDAC on a CRT (screen). It also allows one to inject faults into written data to determine the circuit's behavior in regard to a faulty data. Last, but not least, the program has a feature that enables one to test the EDAC automatically (Function key F8-Test).

When this feature is used, random data is applied to the EDAC in WRITE mode and the generated check bits are read back. The data and the corresponding check bits are then applied to the EDAC in READ mode so that the processed data and the resulting error signals can be compared to the expected values. Finally, every possible single and double errors is systematically injected into the data applied to the EDAC. To probe further the algorithm of this program, consult pages F-1 through F-17 of Appendix F.

4.4 Functional Compatibility

In order to ensure functional compatibility, we compared the the operation of our design with commercially available EDAC chips. Two commonly used EDAC chips were purchased from Texas Instruments (SN 74ALS632) and Integrated Device Technology (49C460). Both chips are 32-bit Error Detection and Correction Units with the capability of correcting a single error and detecting all double errors.

4.4.1 Integrated Device Technology (49C460) Chip

The IDT 49C460 chip is a high speed, low power, 32-bit Error Detection and Correction unit which generates check bits on a 32-bit data field according to a modified Hamming Code and corrects the data word when check bits are supplied.

In WRITE mode data is latched into the input latches. Then the output of the generated check bits is enabled and the check bits are available at the bi-directional CB/SY pins. In READ mode data and check bits are latched into the input latches. The data output latches are transparent during READ mode. Data output is enabled and the corrected data and the syndrome are available at the outputs.

4.4.2 Texas Instruments (SN 74ALS632) Chip

The SN 74ALS632 chip is a 32-bit parallel Error Detection and Correction circuit in a 52 pin 600mil package. The EDAC uses a modified Hamming code to generate a 7-bit check word. This check word is stored along with the data word during the memory write cycle. During a memory read cycle, this 39-bit word from memory is processed by the EDAC to determine if the 32-bit data has been corrupted while it was stored in the memory.

The operation of the TI EDAC is quite different from our design, because of the use of two external decoding signals (S0, S1). In order to operate this EDAC with our test programs, we had to add a decoding logic to the adapter to generate the necessary control signals to this TI chip.

4.4.3 Results of Functional Comparison

A thorough comparison with our design showed that our design comprises all operation modes and features that are implemented in these two commercial circuits. In fact, our design is much more flexible in operation than either one of the commercial chips. However, we discovered two features that were not included in our early design.

The first feature that both commercial circuits possess, is the disabling of the error signals during the WRITE operation. Without this precaution, the state of the error signals would depend on the data written to memory during WRITE mode. Since the required additional circuitry for this feature is small, it is added into our design.

The second feature is an operation mode, called "Detect Mode". In this mode single and double errors are detected (i.e., error signals are active) but single errors are not corrected. To retain compatibility with the commercial circuits this feature is also implemented in our design.

Since it is desirable to operate these commercial chips using the same interface and the same programs that we used for our EDAC design, we build adapters for these commercial chips and developed a generic test program. Schematic diagrams of the adapters are shown on pages C-73 through C-78 in Appendix C. The listing of the test case program, EDAC.C, is provided in Appendix F, pages F-1 through F-17. This test program performs a functional verification of the EDAC units.

4.5 Fault Simulation

Because of the limited number of microsimulator chips at our disposal and because we included scannable latches in our design, we had to divide the fault simulation into two parts: simulation of the combinational logic of the EDAC and simulation of the scannable latches.

4.5.1 Microsimulators

In the EDAC fault simulator, Spaceborne, Inc.'s proprietary microsimulator chips were used in place of their corresponding functional equivalent 4000 series CMOS family of chips. These microsimulator chips are capable of simulating stuck-at, stuck-open and bridging faults. Throughout this report, each type of microsimulator chips are prefixed by MS. Thus, MS4071, MS4011, MS4081 and MS4023 are the microsimulator chips equivalent of 4071, 4011, 4081 and 4023 CMOS chips respectively. Data sheets of the microsimulators are provided on pages D-26 through D-37 in Appendix D.

4.5.2 Fault Simulator Breadboard

The chips layout of the breadboard for the EDAC fault simulator is shown in figures 1 through 24 (see pages D-2 through D-25) of Appendix D. Notice that each microsimulator chips has a shift register to propagate an injected fault to each and every inputs/outputs of logic gates inside it. Connected together, these microsimulator chips form a shift register chain that will be used for fault injection.

In this breadboard layout the vertical (Y) and horizontal (X) numbers are coordinates used to indicate a chip's location. Looking at figure 1 on page D-2 of Appendix D, the chip on the upper left hand corner (X=11, Y=32) is of type 74HC75 located at 3211. FE1 and FE2 are control signals to select the operational mode of the microsimulator chips. For stuck-at fault simulation, FE1 = 0 and FE2 = 1. For stuck-open fault simulation, FE1 = 1 and FE2 = 1.

FCL is the clocking signal to shift the injected fault along the shift register chain. Looking at figure 24 on page D-25 of Appendix D, FDI is the input at which a "1" (simulating a fault) is injected and propagated along the shift register chain as indicated by the arrows. FDO is the output of this shift register chain as shown in figure 14 on page D-15 of Appendix D. FDO is used to verify that only one fault is being injected into the shift register chain. With this layout, we generated a fault list by assigning a fault number to every inputs and output of the gates used to implement the EDAC.

Throughout this report, the notation YX(Z) will be used to denote a test point (i.e. a fault to be checked) where YX is the Y and X coordinates of a chip and Z is its PIN location. For example, 3208(20) refers to the PIN 20 of a chip located at 3208.

A summary of the fault locations, unused faults, and undetectable faults of the EDAC is shown in Appendix D on page D-38 for the combinational logic and on page D-39 for the scannable latches. For the combinational logic, for example, the shift register chain starts at 0105, hence, 0105(1) is assigned fault number 0, 0105(2) is assigned fault number 1, 0105(3) is assigned fault number 2, ... etc. By tracing the direction of shift, we can see that the shift register chain ends at 1109. The test point 1109(19) is the last fault in the combinational logic of the EDAC fault simulator. There is a total of 1464 (fault number 0 to 1463) faults to be checked, with 19 of them not used.

The 19 faults not used are associated with unused gates of the microsimulator chips. For example, an MS4011 comprises four 2-input NAND gates with a total of 12 possible faults. If there is one gate not used the 3 faults of that gate are called "unused faults".

4.5.3 Stuck-at Fault Simulation

4.5.3.1 Stuck-at Fault Mechanism

Since a single stuck-at fault models the logical behavior of the most frequently occurring physical failures, it is widely used in test generations and fault simulation of a digital circuit. A stuck-at fault is typically a short circuit between a signal wire with either the power or the ground.

4.5.3.2 Stuck-at Fault Simulation Hardware

We converted the laboratory prototype (breadboard) of the EDAC (Error Detection and Correction) into a hardware fault simulator by replacing the 4000 series CMOS family chips with Spaceborne, Inc.'s proprietary microsimulator chips.

The fault simulator is capable of injecting stuck-at and stuck-open faults. The EDAC is designed with build-in-self-testing capability. We use a simplified linear feedback shift register (LFSR) for test pattern generation and a simplified shift register for signature compression. This simplified approach proved to be as effective as the linear feedback register (LFSR) approach but with significant savings in hardware.

4.5.3.3 Description of the Stuck-at Fault Simulation

The programs used for the stuck-at fault simulation are FAULT.C for the EDAC logic and FAULT1.C for the scannable latches. These programs are listed in Appendix F on pages F-32 through F-47 and pages F-48 through F-65 respectively.

As the program starts, the EDAC chip is initialized. This ensures that no faults has been injected. Then a user is asked to choose the number of cycles for the simulation. To save time, the first pass should comprise only a small number of cycles (e.g. 10), since in this pass all detectable faults will be injected. For subsequent passes, only those faults not yet detected are injected.

First, the initial seed is written to the EDAC. Then, the shift register formed by the scannable latches is shifted 41 times. This is equivalent to one cycle. After the required number of cycles elapses, the signature which is reflected by the state of the output latches is read back. This signature is called "good machine signature". Now the first fault is injected and the seed is written again. This newly generated signature is then compared with the good machine signature. In the case of a disagreement, the particular fault injected has been detected. If both signatures matches, then the fault was not detected. This process is then repeated for the next fault.

To ensure a correct simulation, the location of the injected fault is monitored throughout the test. In the case of the

occurrence of a fault at a wrong location or the detection of more than one injected faults, the simulation is interrupted and the obtained results are invalidated.

4.5.3.4 Results of Stuck-at Fault Simulation

The results of the stuck-at fault simulation are shown in Appendix D, pages D-40 through D-47 for different numbers of cycles. During this simulation we discovered that the shift registers did not cycle properly. A solution was found by adding an additional D-latch to each and every scannable latch. The effect of these extra latches on the propagation delay is negligible since they are not part in the data path of the EDAC. After this modification, the cycling and the fault simulation worked as expected.

During the stuck-at fault simulation, we achieved a 100% fault coverage with 10 undetectable faults. Three of these undetectable faults were located within a macrocell (scannable latch LS1) from LSI Logic and therefore not accessible for modification. The other 7 faults were located within the decoder in the error corrector block. After further examination of this circuit we discovered that the reconvergent part of the decoder could be simplified. By replacing two NAND gates (each containing one undetectable fault) in this decoder by inverters, the redundancy is eliminated. The use of inverters make the two faults detectable, while the other five faults became detectable once the redundancy has been eliminated.

After implementing these changes into the logic diagram and into the fault simulator breadboard we repeated the stuck-at fault simulation. By choosing a particular seed for the fault simulation of the EDAC logic we were able to achieve a 100% fault coverage with less than 400 cycles.

The fault simulation of the second part (i.e. the scannable latches) proved to be more difficult because it contains latches. To eliminate the dependence on the state of the latches, the circuit has to be initialized before any new cycle is started. Eight faults were only detectable with a special sequence of test vectors. Finally, a 100% stuck-at fault coverage was achieved using only six cycles. The necessary test vectors are listed in a table on page B-16 of Appendix B.

4.5.4 Stuck-open Fault Simulation

4.5.4.1 Stuck-open Fault Mechanism

Besides the classical stuck-at faults, one other important non-classical fault is an open connection. Open connection failures in CMOS process may be caused by faulty manufacturing steps (i.e. overetching, poor contacts, etc) or by electrical stress (i.e.

electromigration of metal).

A failure due to an open connection that turns a MOS transistor permanently off is called a stuck-open fault. Since this open connection behaves like a memory element, a stuck-open fault converts a combinational circuit into a sequential circuit. A stuck-open fault can only be detected by a sequence of test vectors. Depending on the type of a logic gate, the first vector has to charge or discharge the load capacitance so that a subsequent test vector can detect this fault.

An example using a 2-input NAND gate is given in the following. To detect a stuck-open fault at any one of the two parallel PMOS transistors, a test vector must set inputs to both PMOS transistors to "high" to discharge the output load capacitance to ground. Then, a second test vector has to set inputs to both PMOS transistors to "low". If a stuck-open fault exists in any one the PMOS transistors, the output of the NAND gate will remain "low" as opposed to "high" of a functional NAND gate.

4.5.4.2 Stuck-open Fault Simulation Hardware

The 4000 series CMOS family of chips in the breadboard were replaced by Spaceborne, Inc.'s proprietary microsimulator chips. These microsimulator chips are capable of simulating stuck-at, stuck-open, and bridging faults. The selection of the fault type is controlled by the control signals FE1 and FE2. For stuck-open fault simulation, both signals are set to "1". Each fault simulator chip consists of a 12 stage shift register with the shift input FDI and the shift output FDO.

The clock input FCL allows shifting within the shift registers. Each position in this shift register is equivalent to an input or an output of the gates located on the chip. A "1" at one of these positions means a fault is being injected at that position. The shift registers of the different chips are interconnected to form one shift register chain. To inject a fault at any position in the shift register chain a "1" is shifted from the input to that position.

The EDAC circuit is designed with Build-In Self-Test (BIST) capability. This feature reduces considerably the number of steps necessary to perform fault simulation. The BIST circuit consists of two shift register chains, constructed from the input latches and the output latches respectively, and an exclusive OR gate. The Exclusive-OR gate and the shift registers form a simplified Linear Feedback Shift Register (LFSR) which generates the test pattern.

The fault simulator breadboard is connected to an IBM PC/AT to control the fault simulation.

4.5.4.3 Description of the Stuck-open Fault Simulation

To perform stuck-open fault simulation, we modified the test program we used for stuck-at fault simulation accordingly. As the program starts, the EDAC chip is initialized. This ensures that no faults has been injected. Then a user is asked to choose the number of cycles for the simulation. To save time, the first pass should comprise only a small number of cycles (e.g. 10), since in this pass all detectable faults will be injected. For subsequent passes, only those faults not yet detected are injected.

First, the initial seed is written to the EDAC. Then, the shift register formed by the scannable latches is shifted 41 times. This is equivalent to one cycle. After the required number of cycles elapses, the signature which is reflected by the state of the output latches is read back. This signature is called "good machine signature". Now the first fault is injected and the seed is written again. This newly generated signature is then compared with the good machine signature. In the case of a disagreement, the particular fault injected has been detected. If both signatures matches, then the fault was not detected. This process is then repeated for the next fault.

To ensure a correct simulation, the location of the injected fault is monitored throughout the test. In the case of the occurrence of a fault at a wrong location or the detection of more than one injected faults, the simulation is interrupted and the obtained results are invalidated.

4.5.4.4 Results of the Stuck-open Fault Simulation

The results of the stuck-open fault simulation for different number of cycles is shown in Appendix D, pages D-48 through D-53. As mentioned in 4.5.4.1, it is necessary to apply a sequence of test vectors rather than a single vector to detect stuck-open faults. Since it is less likely for a pseudo random test vector generator to produce a sequence of test vectors required to detect stuck-open faults, the fault coverage for stuck-open faults is expected to be significantly lower than the fault coverage for stuck-at faults.

To keep the program as simple as possible we used the same initial conditions used in stuck-at fault simulations for this stuck-open fault simulation. Compared to the stuck-at fault coverage, the fault coverage for stuck-open faults is about 4-5% lower. At 400 cycles, for example, the stuck-open's fault coverage is only 94.8% compared to stuck-at's fault coverage of 100%. This fault coverage can be improved by simply increasing the number of cycles or by applying relatively simple deterministic test patterns. Our engineers were able to show that all stuck-open faults in the EDAC circuit can be detected.

5 SILICON BREADBOARD

5.1 Schematic Diagram

The schematic diagram of the silicon breadboard was derived directly from the schematic diagram of the fault simulator breadboard. Each cell of the fault simulator breadboard was replaced by a corresponding cell from the 2-micron standard cell library supplied by MOSIS. This ensured a functionally correct logic diagram of the silicon breadboard, since the schematic diagram of the laboratory prototype has already been verified. The schematic diagram of the silicon breadboard is provided in figures 1 through 60 of Appendix E.

5.2 Layout

The placement and routing of the silicon breadboard was performed on an IBM PS/2 using a layout editor software called L-Edit from Tanner Research, Inc. The design was manually verified for logic design correctness, and was design rule checked for layout violations. Each cells in the standard cell library supplied by MOSIS was also verified. A standard cell layout approach was combined with customized cells to optimize the performance of the design. Analog circuit simulations to estimate the propagation delay of the critical logic paths were also performed using a software called PSPICE from MicroSim Corp.

5.2.1 Problems During Layout

Due to the limited memory (maximum 640 Kbytes of RAM) accessible by DOS (the operating systems used by L-EDIT) several problems were encountered during the layout. One immediate constraint is that the whole EDAC layout must use less than 640 Kbytes of memory. The solution is a layout technique called "hierarchical layout" in which a larger design is built from many smaller cells that can be verified easily. Finally, careful placement and routing of these cells allow us to circumvent this problem.

5.2.2 Radiation Hardened Design

Following the guidelines for designing radiation hardened devices, we used only 2- or 3-input NAND gates and 2-input Exclusive-OR (Exclusive-NOR) gates to implement the EDAC circuit. Fan-out of logic gates is limited by buffers. Inputs and outputs of latches are also buffered. Finally, the availability of inverting outputs of latches enabled us to save area (no need for inverters) and reduce the propagation delay.

5.2.3 Power Distribution

The power distribution internal to the chip was done such as to sustain large currents that might occur during exposure of the chip to total dose radiation. Moreover, the drive capabilities of buffers and drivers were enhanced by connecting up to three of them in parallel. This was necessary to maintain the performance level of the chip when operated under total dose radiation.

The power and ground interconnections of the standard cells were implemented in first layer of metal, while the global power distribution was laid out in an interleaved network of second level of metal. The power busses of the pad drivers were isolated from those of logic cells to reduce power and ground noise in the logic cells.

To minimize power and ground noise in the pad area, as many pins as possible were allocated for VDD and VSS pads. The lateral interconnect wiring between the functional cells was routed almost exclusively in first level metal while the vertical interconnect wiring was routed almost exclusively in second level of metal as recommended by MOSIS.

5.3 Layout Verification

After the placement and routing was completed, an independent verification of this layout was performed by two other engineers. They inspected the layout manually for open wires and missing interconnections and employed the design rule checker of L-Edit to ensure no layout rules was violated. Since the L-Edit design rule checker had a few restrictions due to memory limitations, only single hierarchical blocks can be checked. Consequently, we decided to have a much more comprehensive design rule checking on the complete layout done at MOSIS.

5.4 Fabrication of the Silicon Breadboards

This layout of the EDAC was submitted for fabrication on August 14, 1989. A checking performed at MOSIS identified four types of design rule violations at the interfaces between hierarchical blocks. These mishaps were corrected immediately and the EDAC design was released for fabrication on September 4, 1989.

5.5 Testing of the Silicon Breadboards

We received ten silicon breadboard chips of the EDAC (Error Detection and Correction) from MOSIS and tested them. Eight out of the ten chips passed the test. The quiescent current range of working chips varied from 0.8 mA to 3.3 mA while the defective ones varied from 3.8 mA to 4.7 mA.

5.5.1 Random Test

In the random test, random data was applied to the EDAC in WRITE mode and the generated check bits were read back. Then the data and the corresponding check bits were applied to the EDAC in READ mode and the output data and the resulting error signals were compared with the expected values. Every possible single error and double error was injected into the data applied to the EDAC. With over 1 million test vectors applied, eight chips passed the test.

5.5.2 Functional Test

The second test was a comprehensive functional test using a method called sensitized partitioning. This method enabled us to test exhaustively each functional block of the EDAC. We applied an exhaustive set of test vectors to each of the functional blocks (i.e. check bit generator, error detector, and error corrector), and compared the response with a set of responses derived from the specifications of the EDAC. These exhaustive tests were repeated three times on each chip and no errors were detected in all eight chips that passed the previous random test.

5.5.3 Built-In Self-Test

The final testing was done using the BIST feature (Built-In Self-Test). We applied the initial seed to each of the silicon prototypes and used 380 clock cycles to generate the signature. This signature was then compared with the good machine signature generated earlier during the stuck-at fault simulation of the EDAC breadboard. Again, the signature of each of the eight chips that passed the previous two functional tests was identical to the good machine signature. The respective signatures of the two chips that failed in the two functional tests again did not match the good machine signature. Thus, faulty chips can be screened out by both functional testing and built-in-self-testing.

5.5.4 Quiescent Current

Finally, the results of quiescent current measurements of each of the silicon prototypes are tabulated in the following:

Chip No.	Quiescent Current I_{DDQ}	passed/failed
1	1.15 mA	passed
2	4.67 mA	failed
3	3.85 mA	failed
4	1.37 mA	passed
5	1.95 mA	passed
6	2.78 mA	passed
7	1.96 mA	passed
8	2.10 mA	passed
9	3.31 mA	passed
10	0.79 mA	passed

5.5.5 Propagation Delay

A measurement of the actual propagation delays of the silicon breadboards showed that the maximum propagation delay is within the limits of the performance requirements. A detailed table of the measured propagation delays of each of the eight working chips is provided on page E-65 of Appendix E.

5.6 Summary of EDAC Features

The EDAC circuit consists of 32 bi-directional data pins, 7 bi-directional check bit pins, 15 input pins, and 3 output pins. The 84 pin package still offers plenty of available pins to be used as power and ground pins. The complete design consists of 2103 equivalent gates (i.e. 4206 transistor pairs). The estimated power dissipation is 300 mW. The measured maximum propagation delay of approximately 76 ns is within the limits of the performance requirements. The pin configuration and a description of the pins can be found in Appendix E on page E-62 and pages E-64 through E-65, respectively.

5.7 Delivery to NASA/JPL

The EDAC chips were delivered to NASA/JPL on November 28, 1989.

NOTE:

The complete figures, computer programs, and logic diagrams included in the appendixes (page 25 through page 353) were delivered only to Mr. Larry Hess of JPL.

APPENDIX A: INTRODUCTION

- A.1 Design Specifications
 - A.1.1 Modified Hamming Code.....A-1
 - A.1.2 Syndrome Decode.....A-2
 - A.1.3 TOME: Three Or More Errors.....A-3
 - A.1.4 Logic Equations.....A-4
- A.2 Performance Requirements.....A-5
- A.3 Design Approach
 - A.3.1 Figure 1.....A-6
 - A.3.2 Figure 2.....A-7
 - A.3.3 Figure 3.....A-8
 - A.3.4 Figure 4.....A-9
 - A.3.5 Figure 5.....A-10
- A.4 SBIR Project Schedule.....A-11

**MODIFIED HAMMING CODE
FOR 16 OR 32 BIT EDAC UNIT**

CHECK BIT	DATA BITS																																			
	B31	B30	B29	B28	B27	B26	B25	B24	B23	B22	B21	B20	B19	B18	B17	B16	B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0				
CB0			X					X				X								X																
CB1			X				X														X															
CB2						X					X												X													
CB3	X				X					X										X						X									X	
CB4					X											X										X										X
CB5	X	X	X													X	X	X					X				X								X	
CB6	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X																					

Each check bit (CB0 to CB6) is generated as an exclusive-OR of either the thirteen or fourteen data bits noted by an "X".

Note: Some triple-bit errors are not detectable as such and may be interpreted as single-bit errors and falsely corrected as single-data errors. This is true for all standard EDAC circuits using a modified Hamming-code matrix.

[M.Y. Hsiao, IBM, US Patent 3623155, 1969]

SYNDROME DECODE

SYNDROME BITS S0 S1 S2	S6	S5	S4	S3	C5	C4	C3	C2	C1	C0	C6
0 0 0	0	1	0	0	T	0	0	*	T	T	T
0 0 1	0	1	1	0	T	1	0	C2	T	T	T
0 1 1	0	1	1	1	T	1	1	T	11	T	T
0 1 0	0	1	0	1	T	0	0	C1	T	T	T
1 1 0	1	1	0	0	T	1	1	T	9	T	T
1 1 1	1	1	1	0	T	1	1	M	T	M	T
1 0 1	1	0	1	0	T	1	0	T	10	T	T
1 0 0	1	0	0	0	T	0	0	C0	T	15	T

- * : No error detected
- Number: The location of the single bit-in-error
- T : Two or even number of errors detected
- M : Three or more errors detected
- C_ : The location of the check bit in error

THREE OR MORE THAN TWO ERRORS (TOME)

SYNDROME BITS	S6	S5	S4	S3	S0	S1	S2											
0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 1 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0 1 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 1 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 1 1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$TOME = S0.S3.S6 + S0.S1.S2 + S4.S5.S6 + S0.S3.S4.S5 + S1.S3.S4.S5 + S1.S3.S6 (S4 + S5)$$

LOGIC EQUATIONS

$$\text{ERROR} = S_0 + S_1 + S_2 + S_3 + S_4 + S_5 + S_6$$

$$\overline{\text{ERROR}} = \overline{S_0.S_1.S_2.S_3.S_4.S_5.S_6}$$

$$\overline{\text{MULT ERROR}} = \overline{(S_0 \oplus S_1 \oplus S_2 \oplus S_3 \oplus S_4 \oplus S_5 \oplus S_6) \cdot (\text{ERROR})} + \text{TOME}$$

$$\begin{aligned} \text{TOME} = & S_0.S_3.S_6 + S_0.S_1.S_2 + S_4.S_5.S_6 + S_0.S_3.S_4.S_5 \\ & + S_1.S_3.S_4.S_5 + S_1.S_3.S_6 (S_4 + S_5) \end{aligned}$$

PERFORMANCE REQUIREMENTS
FOR ENGINEERING PARTS

- POWER SUPPLY: 4.5V to 5.5V
- BURN-IN POWER SUPPLY: NONE
- AMBIENT TEMPERATURE: 0°C to +70°C
- TOTAL DOSE RADIATION: NONE
- LATCH-UP RESISTANCE TEST: NONE
- SINGLE EVENT UPSET
THRESHOLD LET: NONE
- FAILURE RATE: NONE
- MAXIMUM PROPAGATION DELAY
(CMOS LEVELS, $C_L = 50\text{pF}$) 100nsec

EDAC UNIT DESIGN APPROACH

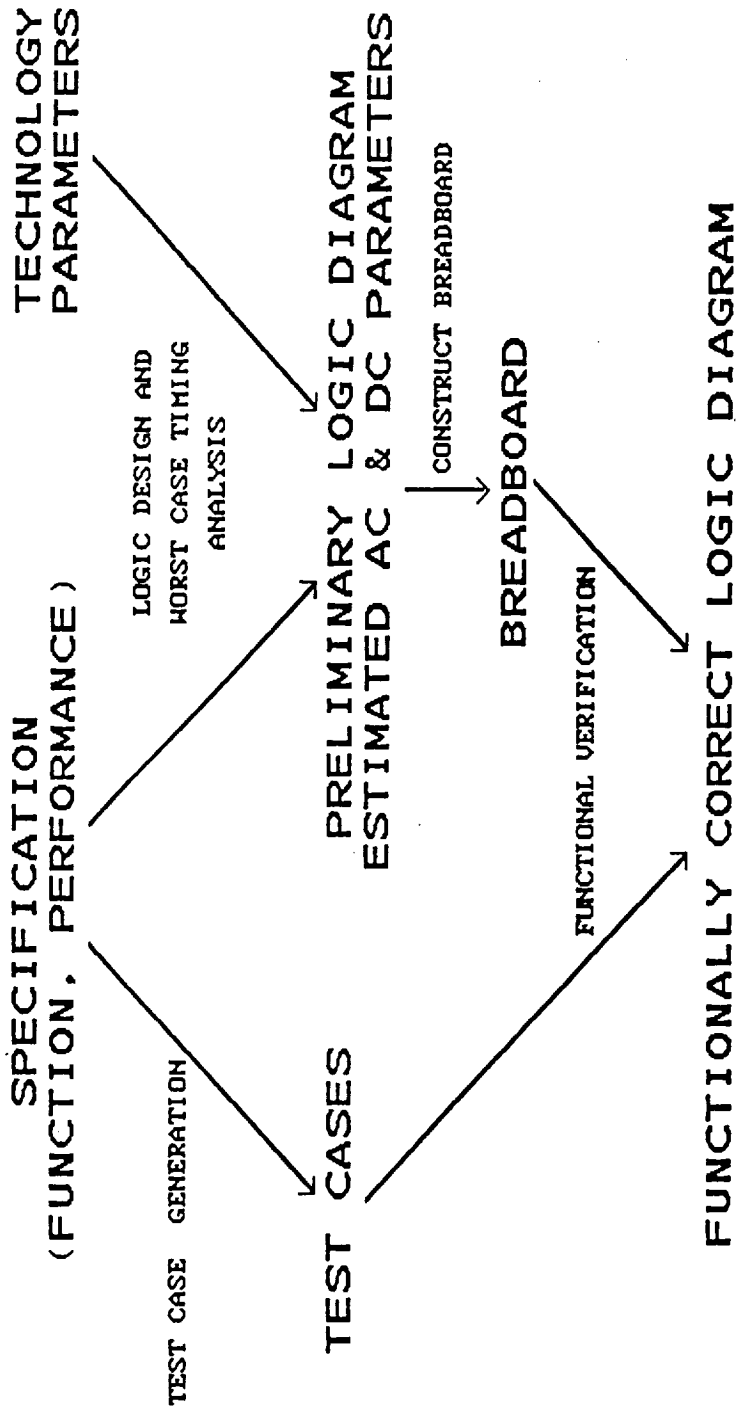


Figure 1

EDAC UNIT DESIGN APPROACH

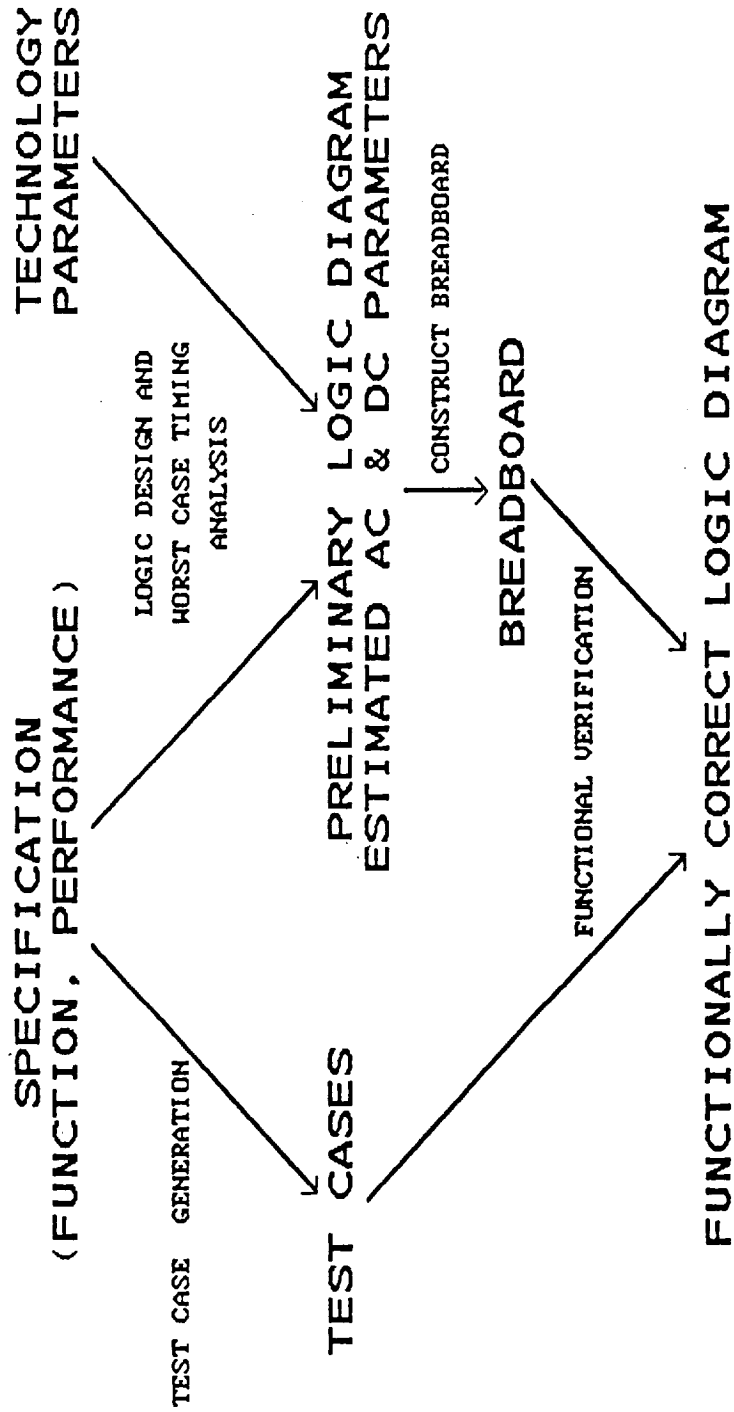


Figure 2

EDAC UNIT DESIGN APPROACH

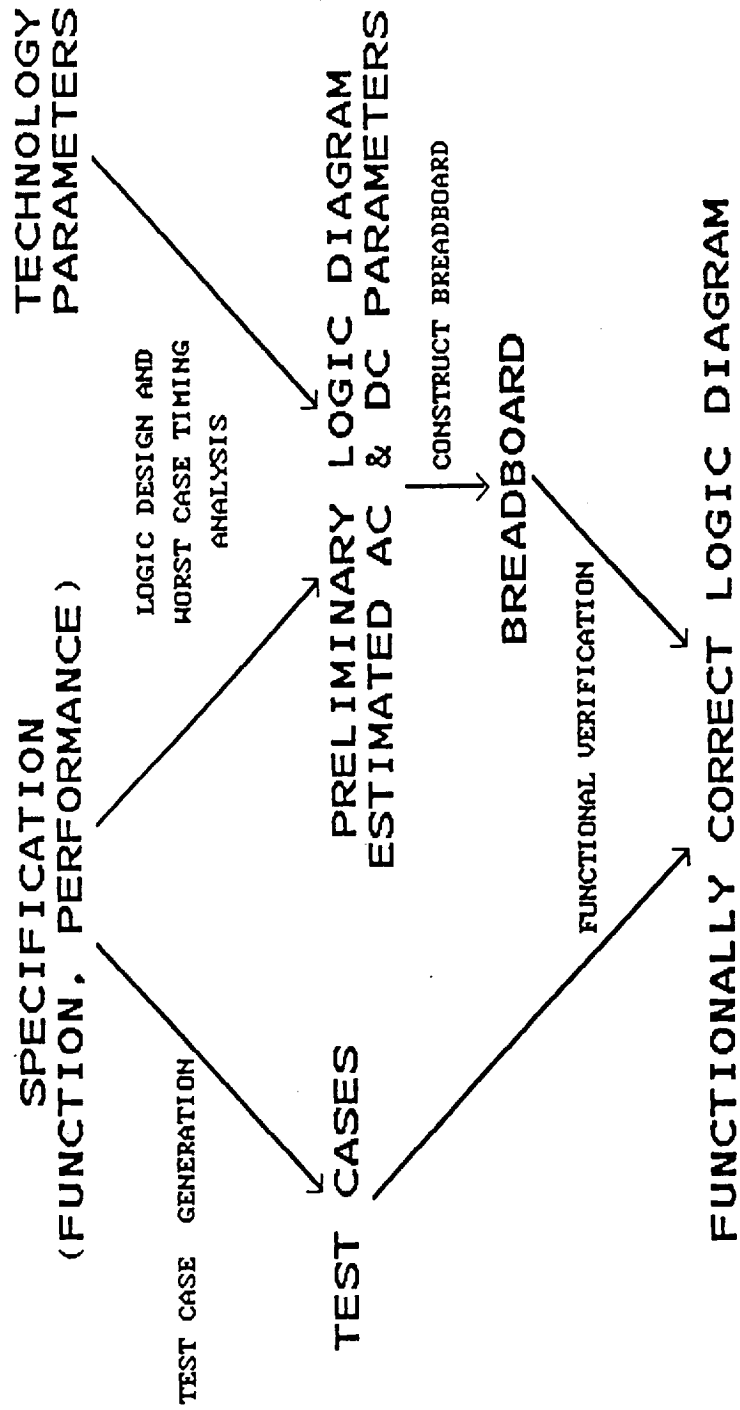


Figure 3

EDAC UNIT DESIGN APPROACH

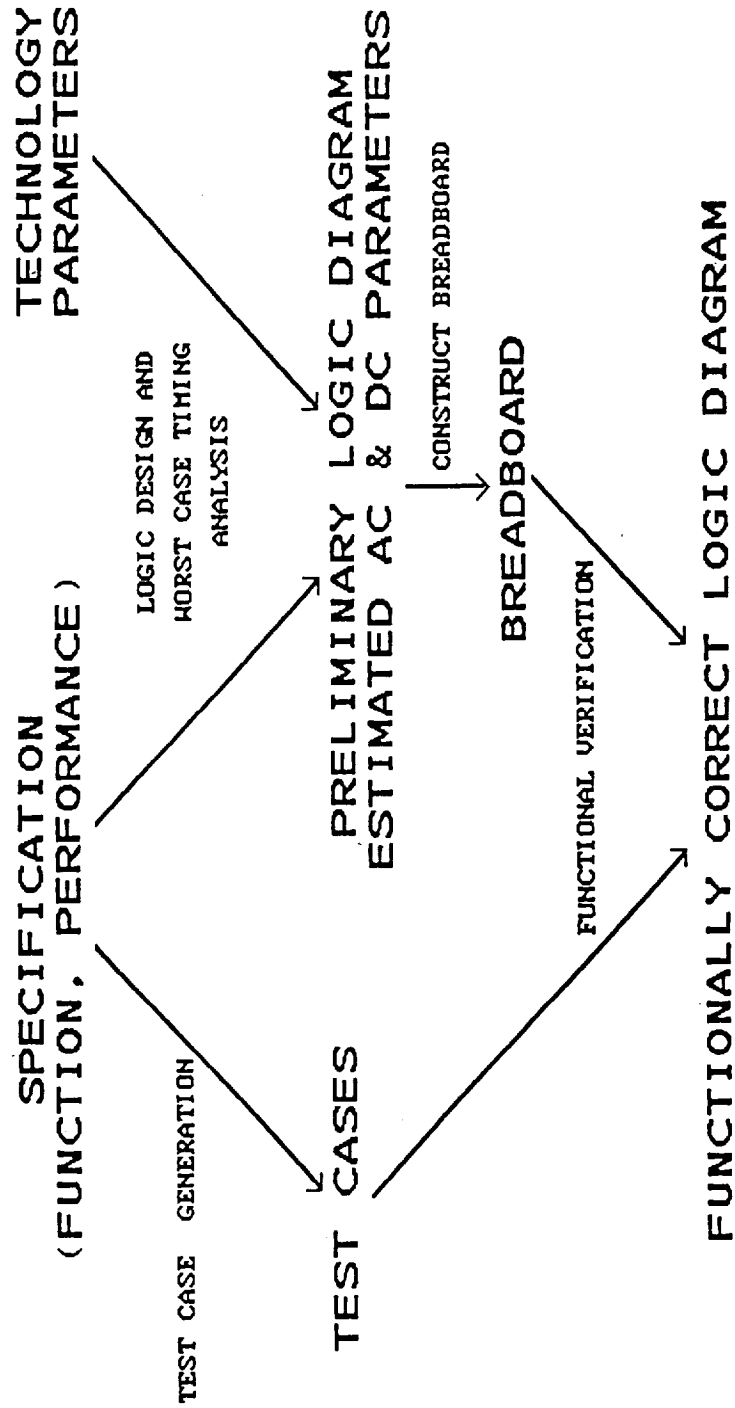


Figure 4

EDAC UNIT DESIGN APPROACH

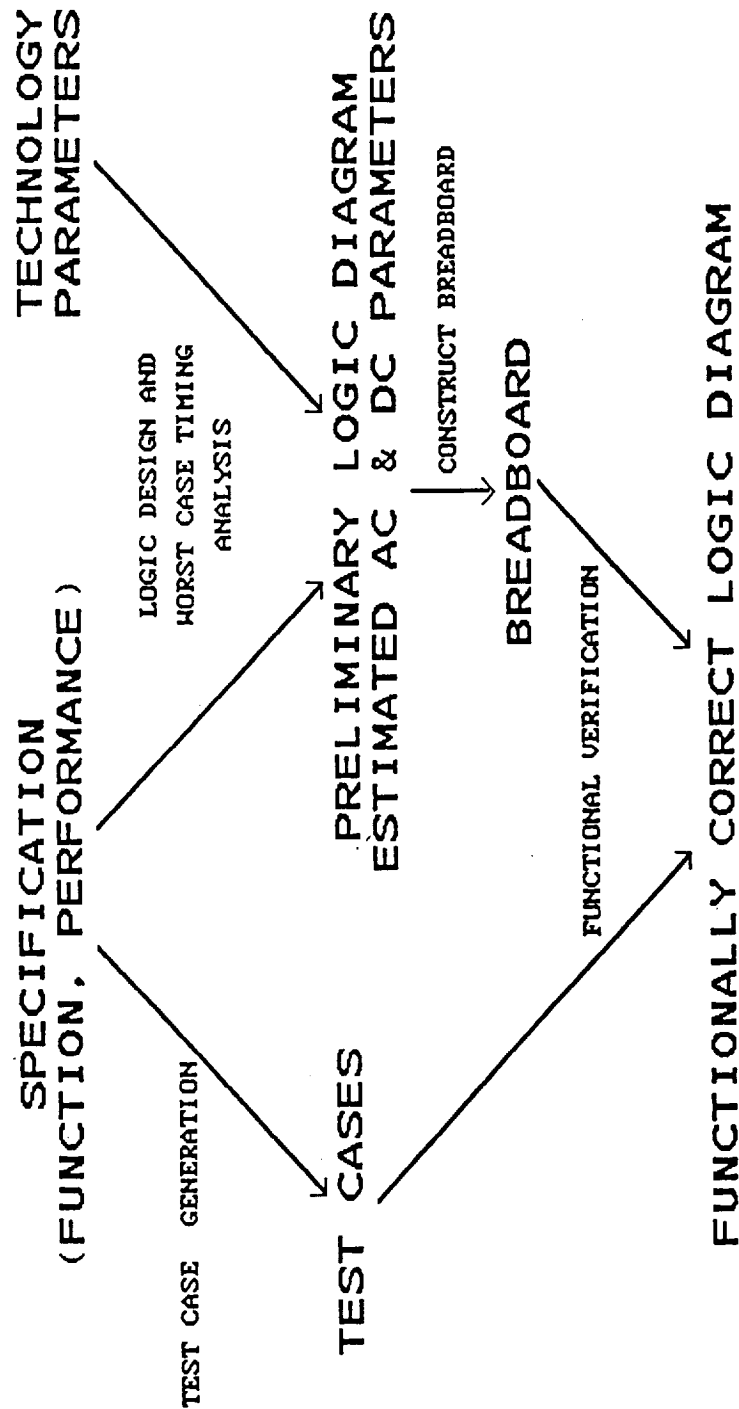


Figure 5

SPACEBORNE, INC.
 742 Foothill Blvd.
 La Canada, CA 91011
 (818) 952-0126

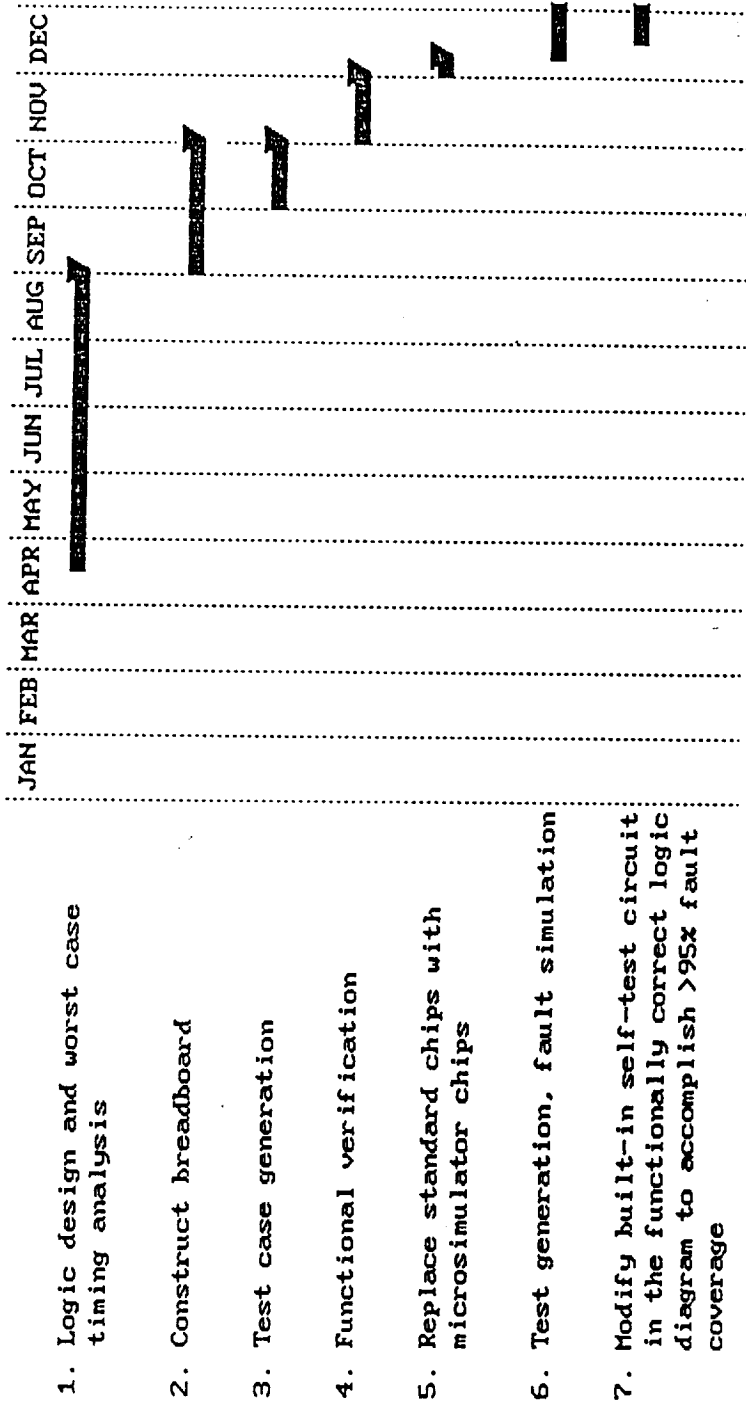
Status as of: 28 November 1989
 Creation Date: 18 April 1988

Achievement Responsibility: Constantin Timoc

SBIR SCHEDULE

TASK

1988

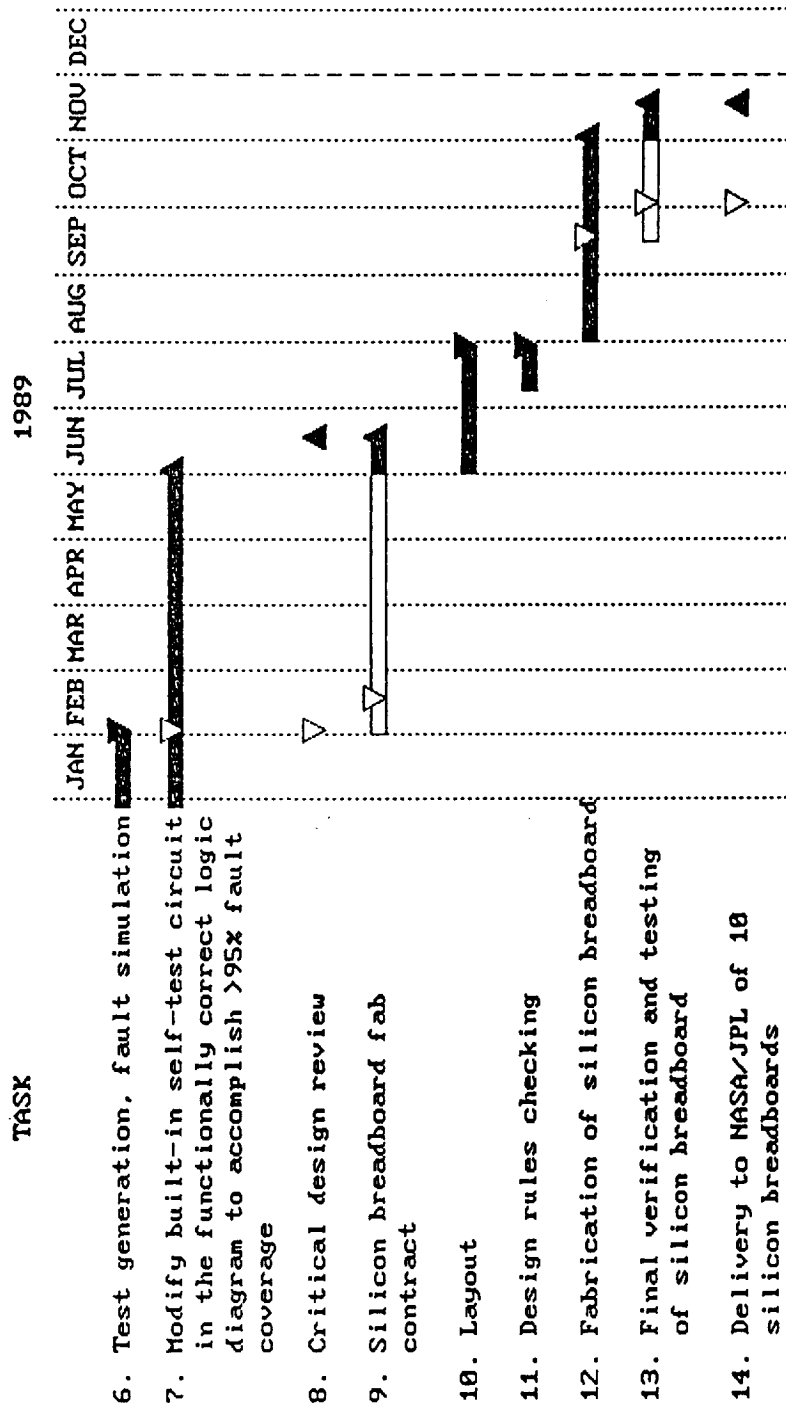


EDCSCHI . DMG

SPACEBORNE, INC.
 742 Foothill Blvd.
 La Canada, CA 91011
 (818) 952-0126

Status as of: 28 November 1989
 Creation Date: 18 April 1988
 Achievement Responsibility: Constantin Timoc

SBIR SCHEDULE (CONT'D)



APPENDIX B: ERROR DETECTION AND CORRECTION DESIGN

- B.1 EDAC Block Diagram
 - B.1.1 Map of EDAC Block Diagram.....B-1
 - B.1.2 Figure 1.....B-2
 - B.1.3 Figure 2.....B-3
 - B.1.4 Figure 3.....B-4
 - B.1.5 Figure 4.....B-5
 - B.1.6 Figure 5.....B-6
 - B.1.7 Figure 6.....B-7
 - B.1.8 Figure 7.....B-8
 - B.1.9 Figure 8.....B-9
 - B.1.10 Figure 9.....B-10
- B.2 Built-In Self-Test Block Diagram.....B-11
- B.3 EDAC Operating Modes.....B-12
- B.4 Switching Waveforms
 - B.4.1 Write Mode.....B-13
 - B.4.2 Read Mode.....B-14
 - B.4.3 Self Test Mode.....B-15
- B.5 BIST Test Vectors.....B-16
- B.6 16-Bit Configuration.....B-17

Map of EDAC Block Diagram

Figure 1	Figure 2	Figure 3
Figure 4	Figure 5	Figure 6
Figure 7	Figure 8	Figure 9

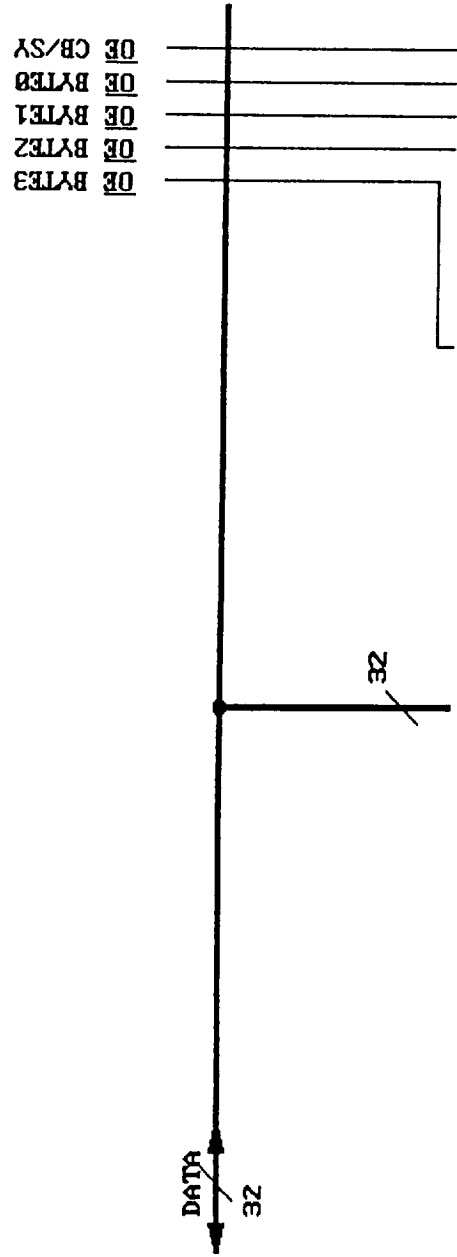


Figure 1

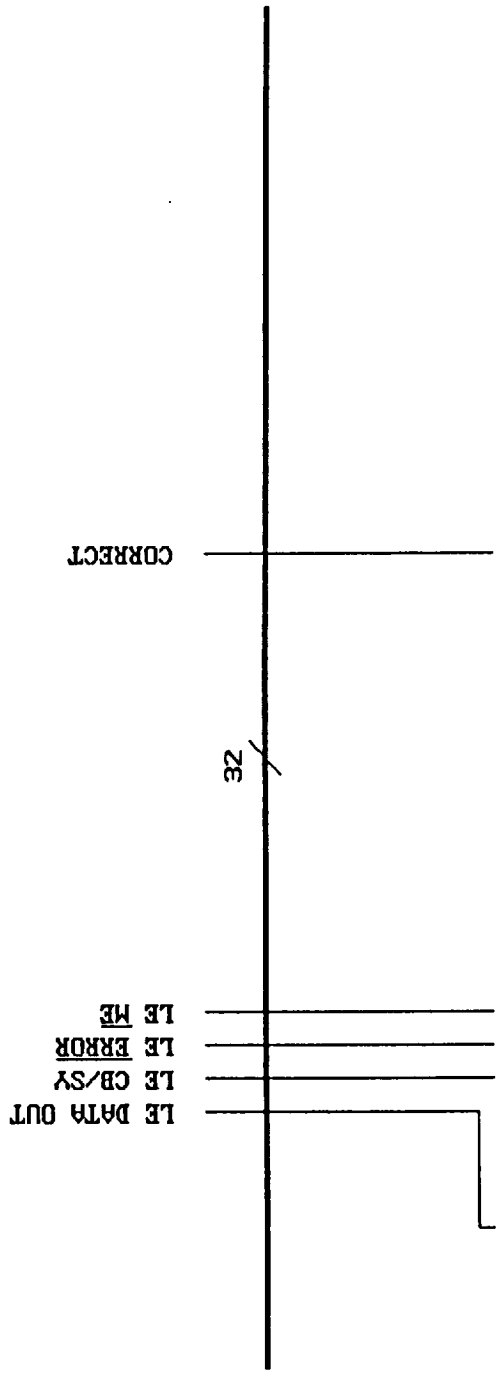


Figure 2

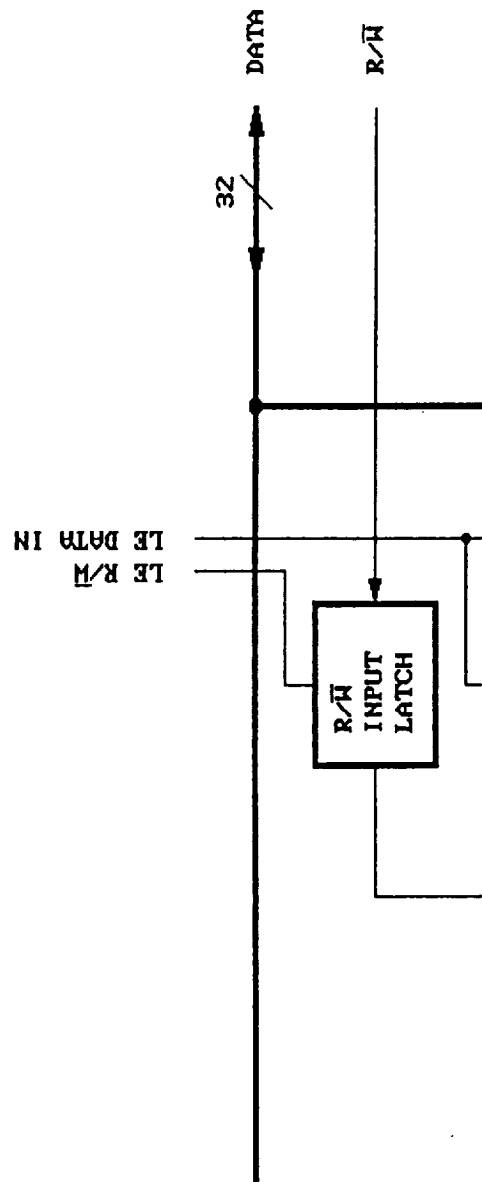


Figure 3

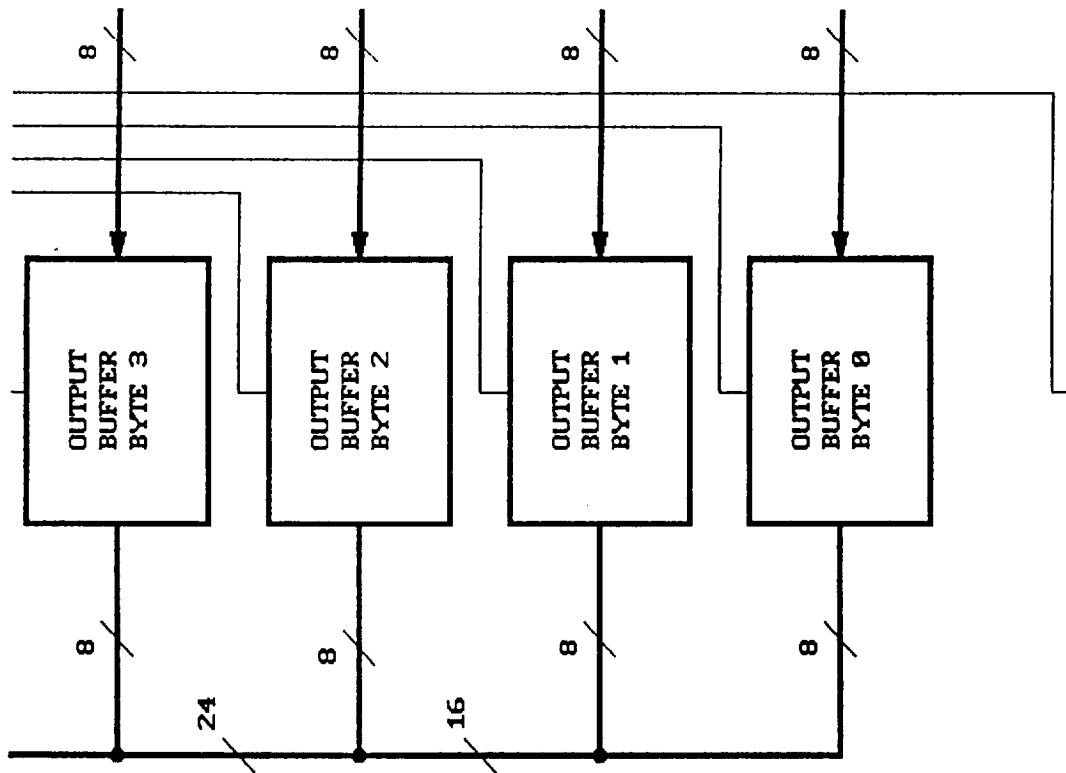


Figure 4

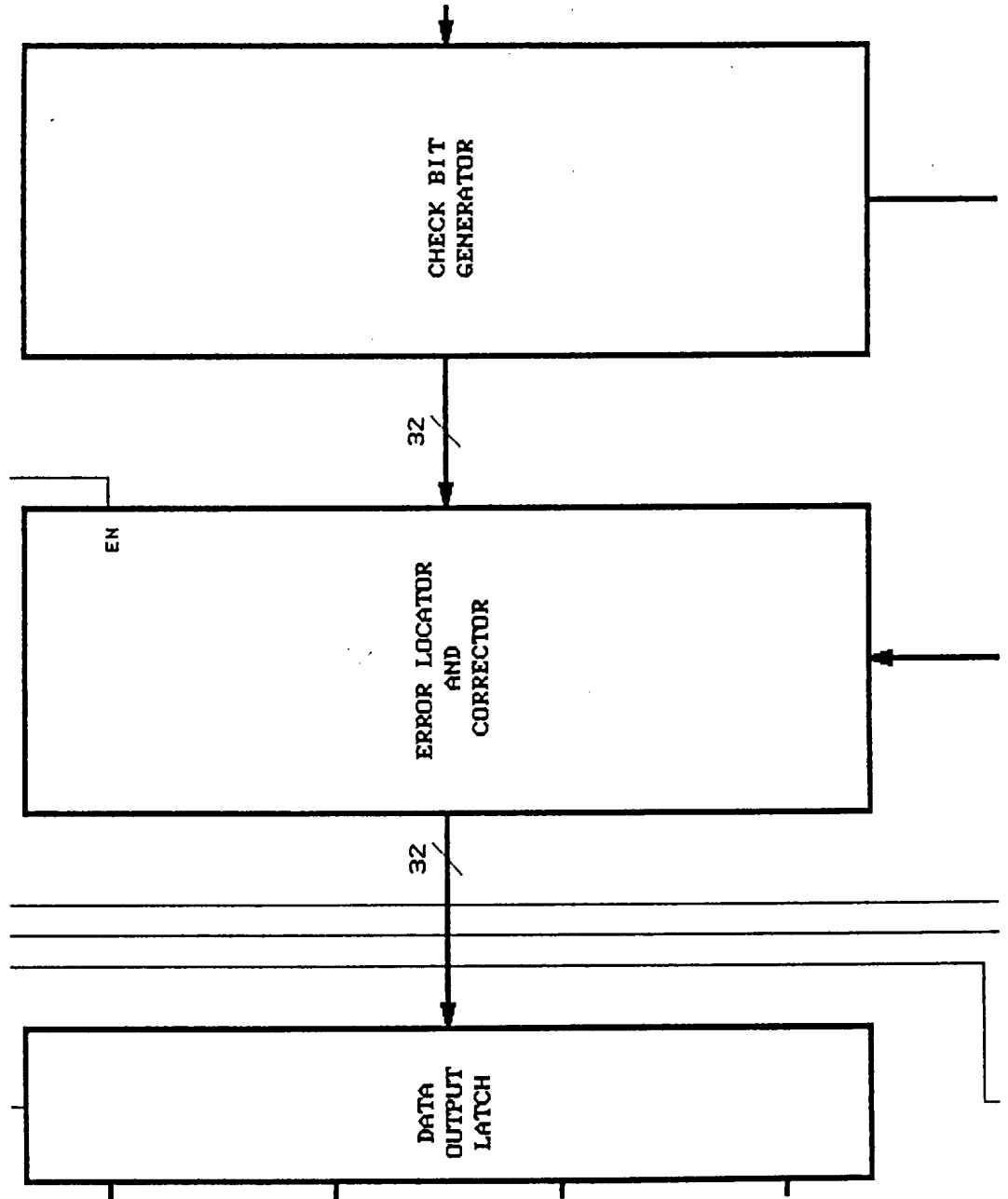


Figure 5

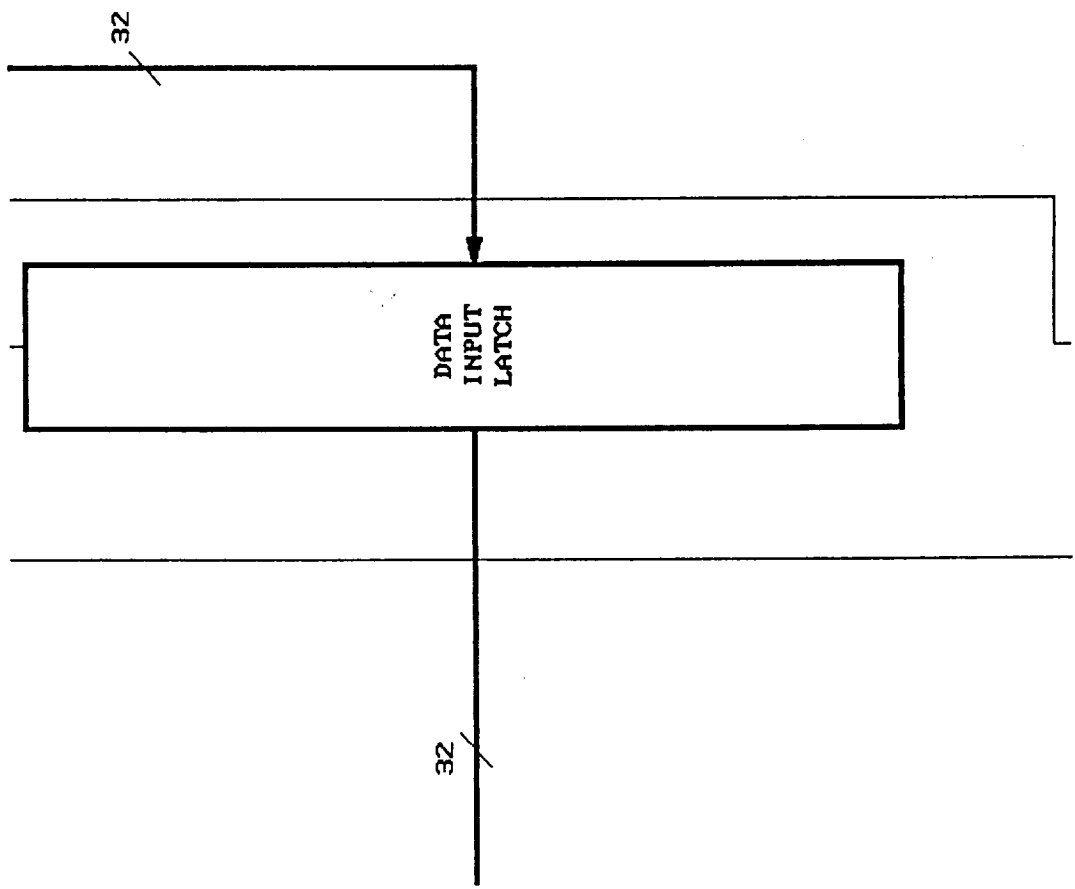


Figure 6

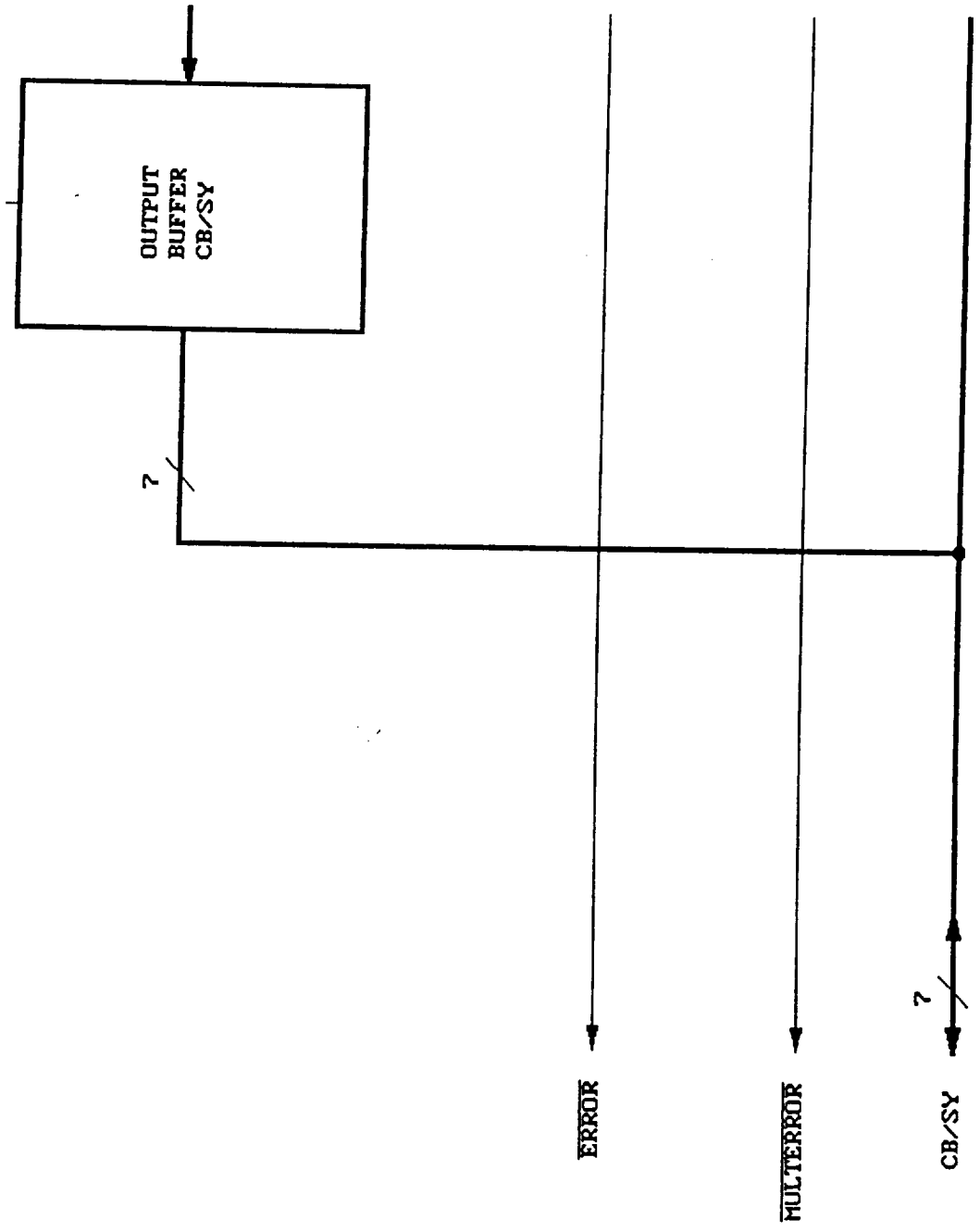


Figure 7

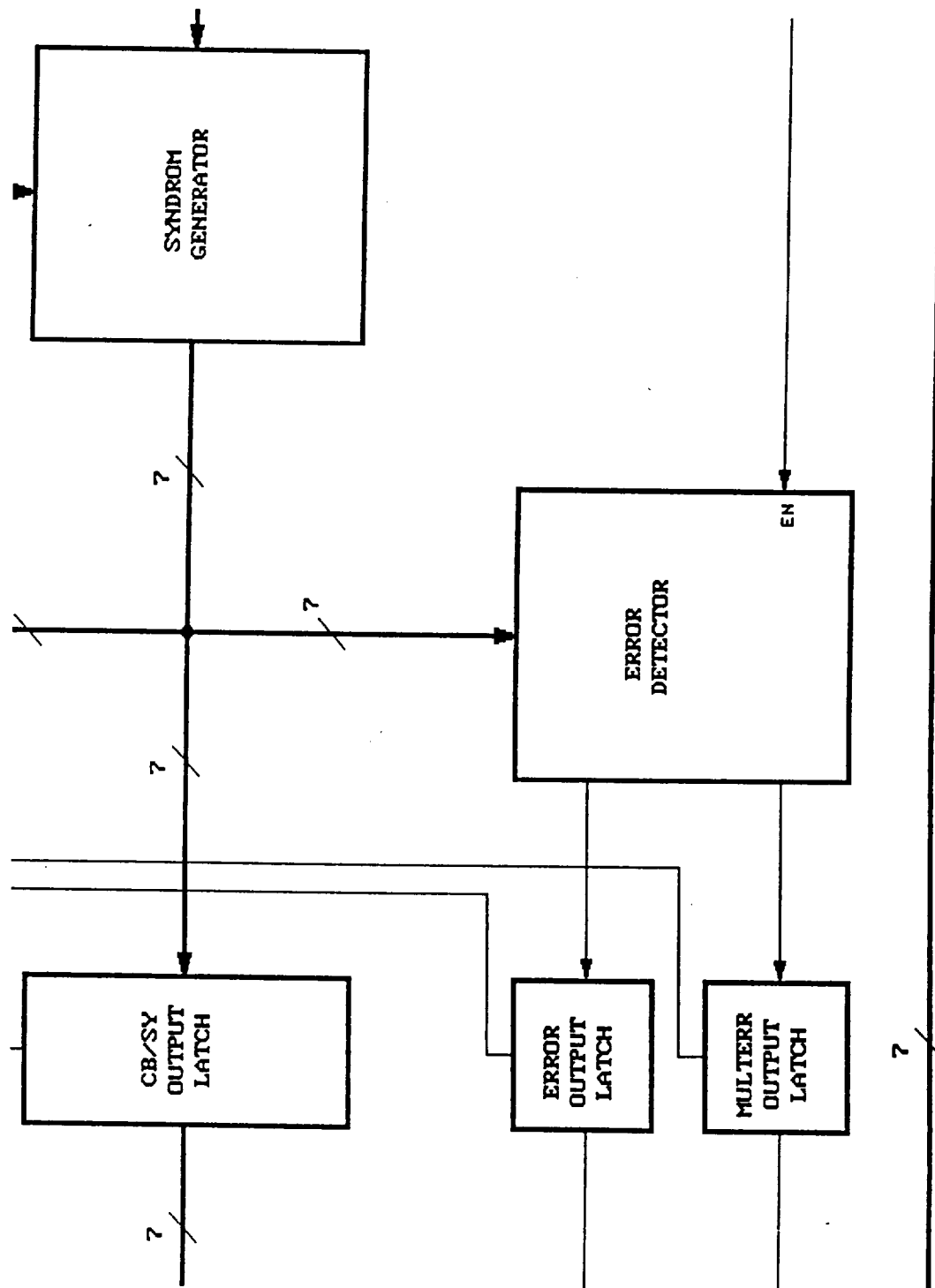


Figure 8

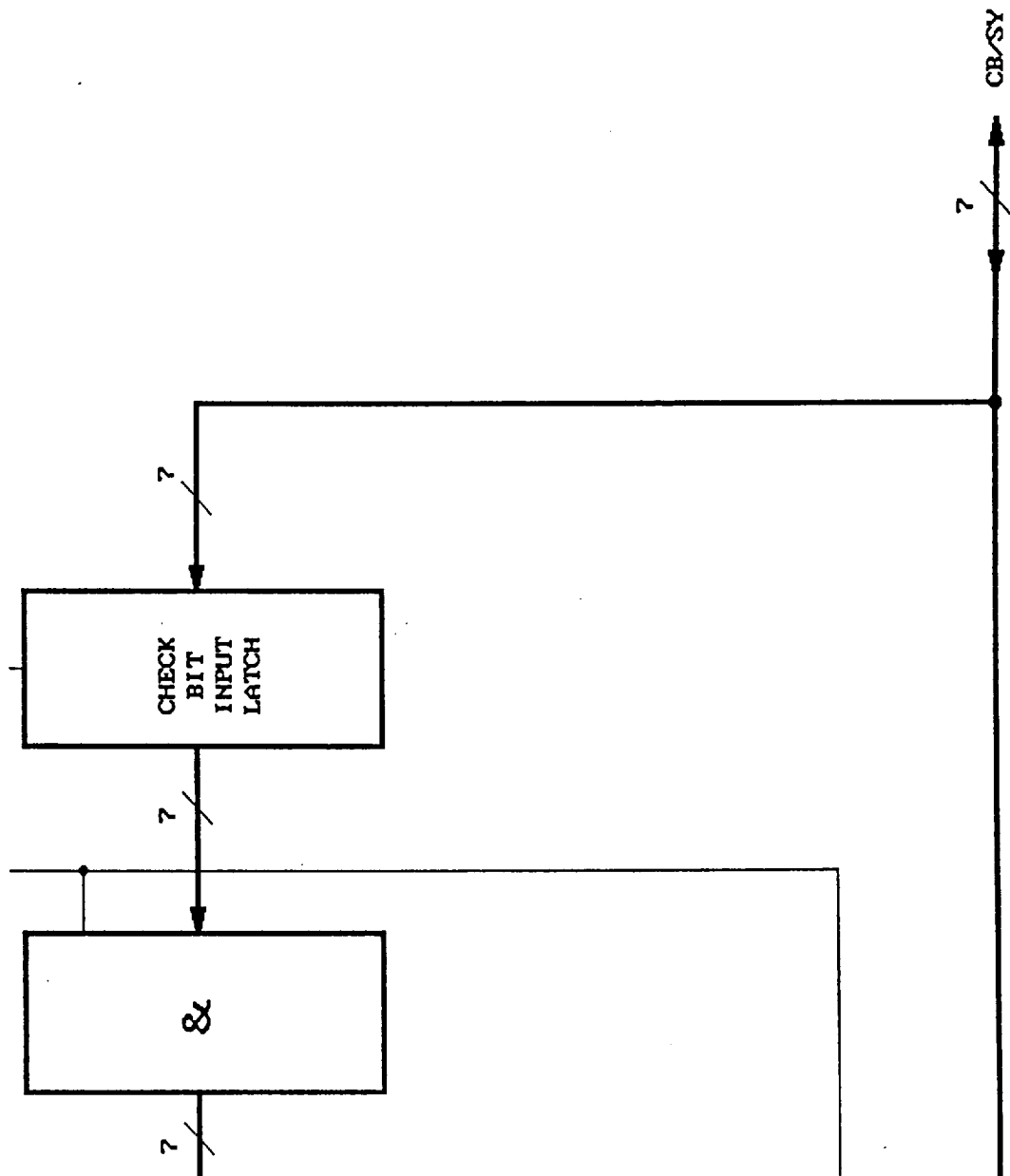
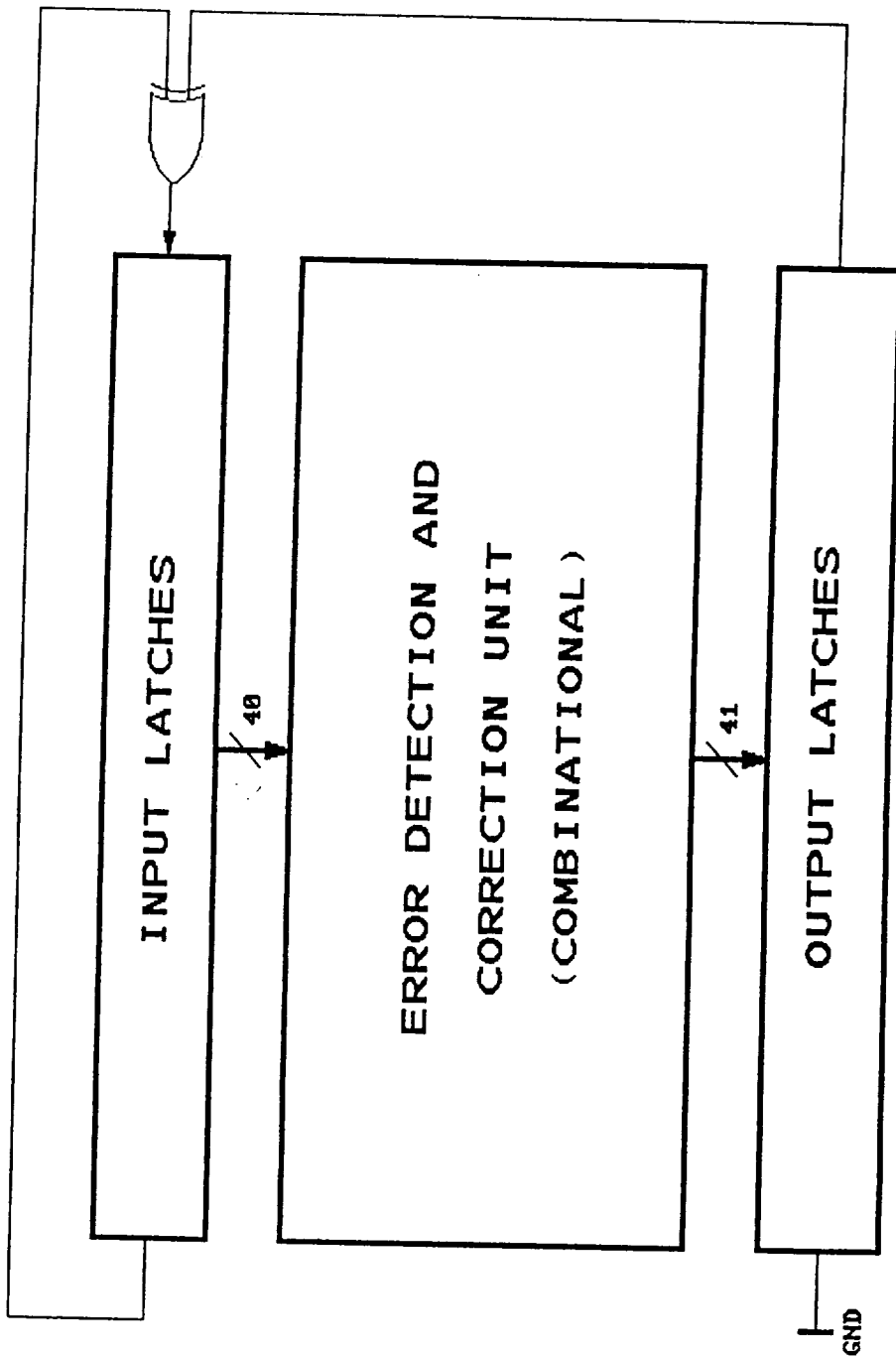


Figure 9

BUILT-IN SELF-TEST



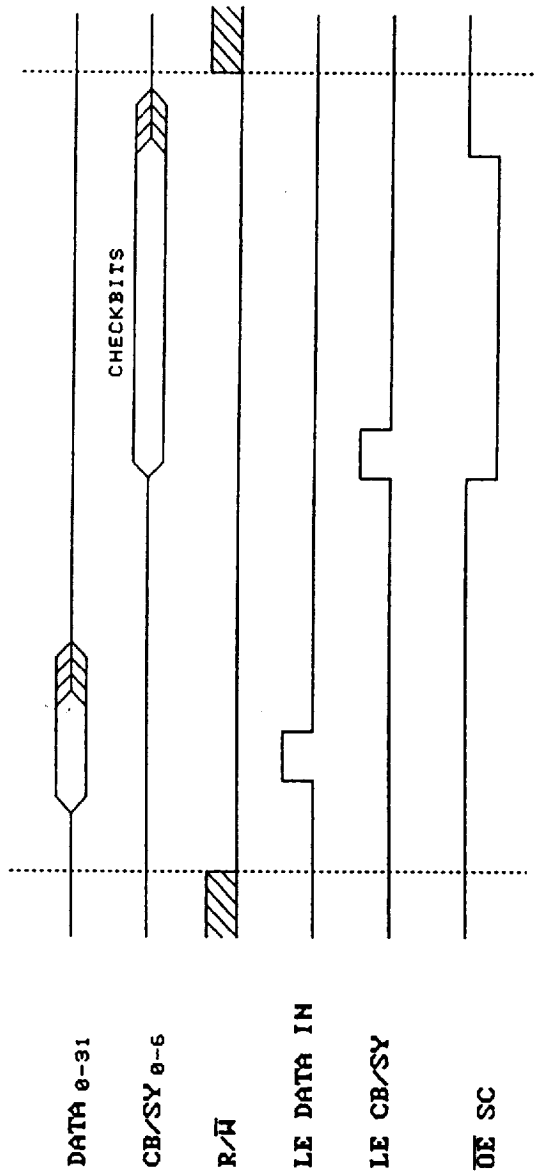
EDAC OPERATING MODES

MODE	R/ \bar{H}	CORRECT	DATA OUT	CB/SY OUT	ERROR MULT_ERROR
WRITE	0	X	-	Check bit generated from Data in	HIGH
READ DETECT	1	0	Data in	Syndrome	Error dependend
READ CORRECT	1	1	Data in with single bit correction	Syndrome	Error dependend
SELF-TEST	X ^{**}	X ^{**}	Signature (32-bits)	Signature (7-bits)	Signature (2-bits)
PASS THRU	X	0	Data in	CB/SY ^{**}	-

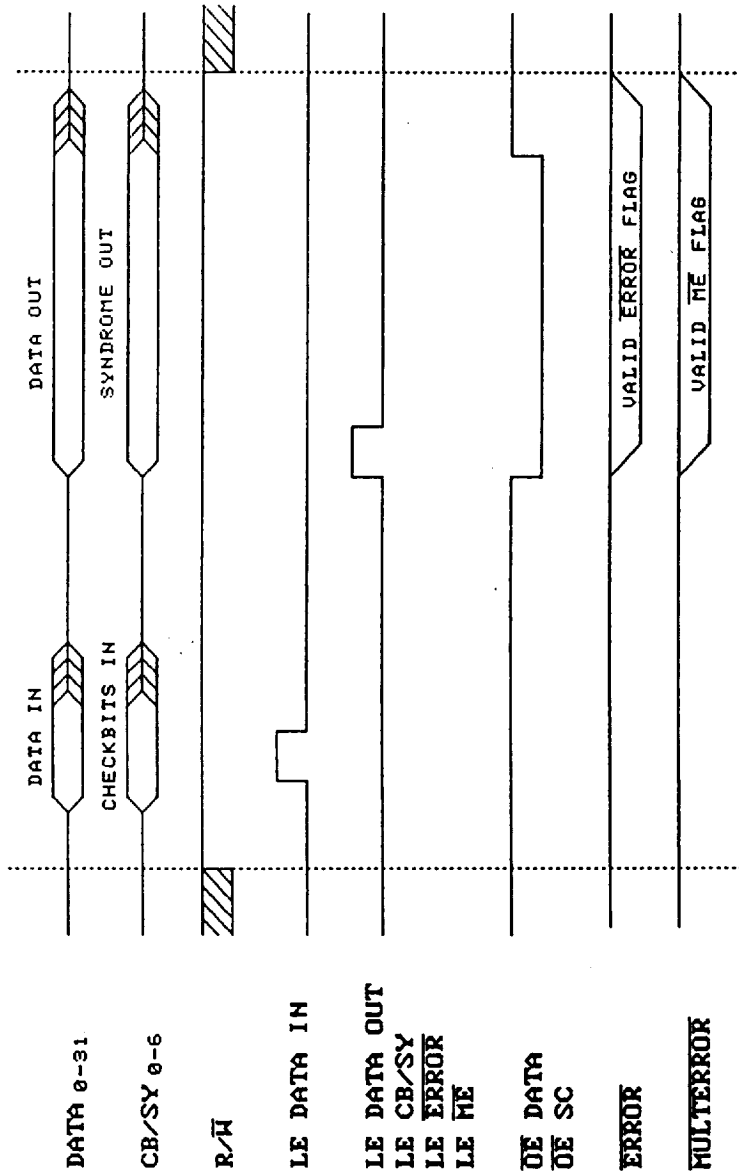
^{**} changing during Self-Test

^{**} depending on R/ \bar{H} signal

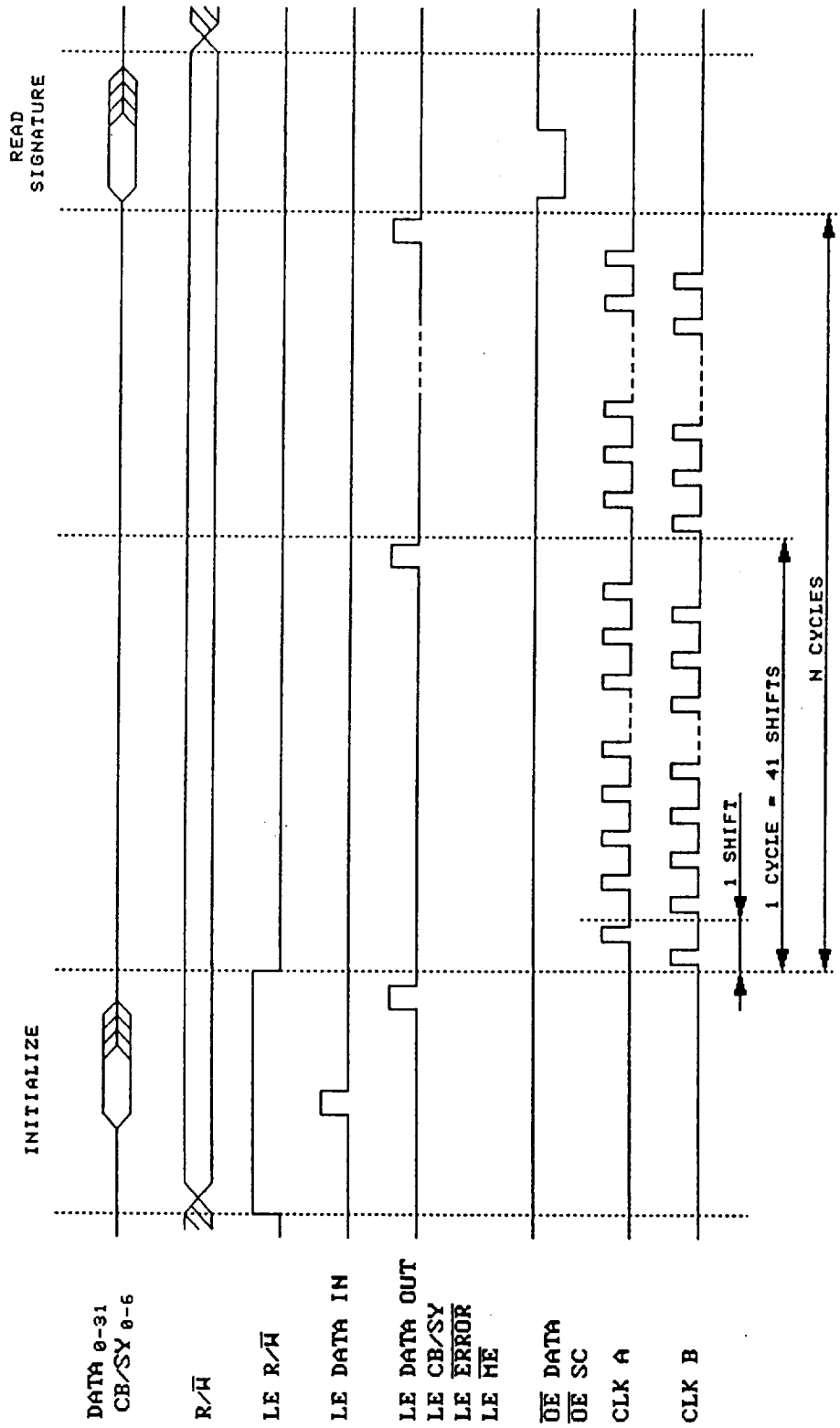
WRITE MODE SWITCHING WAVEFORMS



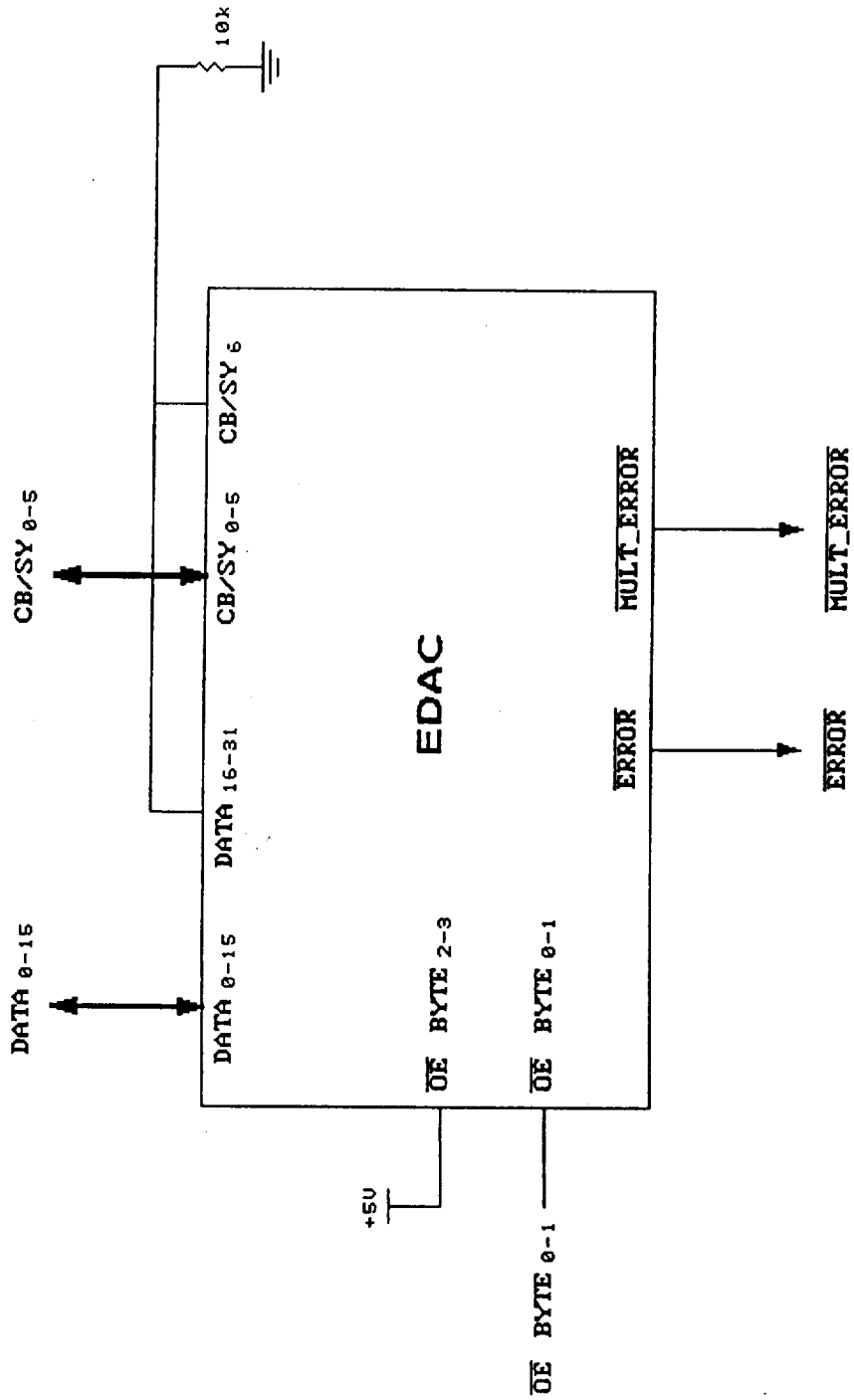
READ MODE SWITCHING WAVEFORMS



SELF TEST MODE SWITCHING WAVEFORMS



16 - BIT CONFIGURATION



APPENDIX C: FAULT SIMULATOR BREADBOARD

C.1 Logic Diagram of Fault Simulator Breadboard

C.1.1 Map of Fault Simulator Breadboard.....C-1

C.1.2 Figure 1.....C-2

C.1.3 Figure 2.....C-3

C.1.4 Figure 3.....C-4

C.1.5 Figure 4.....C-5

C.1.6 Figure 5.....C-6

C.1.7 Figure 6.....C-7

C.1.8 Figure 7.....C-8

C.1.9 Figure 8.....C-9

C.1.10 Figure 9.....C-10

C.1.11 Figure 10.....C-11

C.1.12 Figure 11.....C-12

C.1.13 Figure 12.....C-13

C.1.14 Figure 13.....C-14

C.1.15 Figure 14.....C-15

C.1.16 Figure 15.....C-16

C.1.17 Figure 16.....C-17

C.1.18 Figure 17.....C-18

C.1.19 Figure 18.....C-19

C.1.20 Figure 19.....C-20

C.1.21 Figure 20.....C-21

C.1.22 Figure 21.....C-22

C.1.23 Figure 22.....C-23

C.1.24 Figure 23.....C-24

C.1.25 Figure 24.....C-25

C.1.26	Figure 25.....	C-26
C.1.27	Figure 26.....	C-27
C.1.28	Figure 27.....	C-28
C.1.29	Figure 28.....	C-29
C.1.30	Figure 29.....	C-30
C.1.31	Figure 30.....	C-31
C.1.32	Figure 31.....	C-32
C.1.33	Figure 32.....	C-33
C.1.34	Figure 33.....	C-34
C.1.35	Figure 34.....	C-35
C.1.36	Figure 35.....	C-36
C.1.37	Figure 36.....	C-37
C.1.38	Figure 37.....	C-38
C.1.39	Figure 38.....	C-39
C.1.40	Figure 39.....	C-40
C.1.41	Figure 40.....	C-41
C.1.42	Figure 41.....	C-42
C.1.43	Figure 42.....	C-43
C.1.44	Figure 43.....	C-44
C.1.45	Figure 44.....	C-45
C.1.46	Figure 45.....	C-46
C.1.47	Figure 46.....	C-47
C.1.48	Figure 47.....	C-48
C.1.49	Figure 48.....	C-49
C.1.50	Figure 49.....	C-50
C.1.51	Figure 50.....	C-51
C.1.52	Figure 51.....	C-52

C.1.53	Figure 52.....	C-53
C.1.54	Figure 53.....	C-54
C.1.55	Figure 54.....	C-55
C.1.56	Figure 55.....	C-56
C.1.57	Figure 56.....	C-57
C.1.58	Figure 57.....	C-58
C.1.59	Figure 58.....	C-59
C.1.60	Figure 59.....	C-60
C.1.61	Figure 60.....	C-61
C.2	Detailed Diagram of XOR/XNOR Gates.....	C-62
C.2.1	Position of XOR Gates.....	C-63
C.2.2	Position of XNOR Gates.....	C-67
C.3	Detailed Diagram of Scannable Latches.....	C-68
C.3.1	Position of Scannable Latches.....	C-69
C.4	Connector Assignment.....	C-72
C.5	Pin Assignment of CMOS chips.....	C-73
C.6	Compatibility	
C.6.1	Integrated Devices Technology (IDT 49C460) Chip	
C.6.1.1	Map of IDT 49C460 Adapter.....	C-74
C.6.1.2	Figure 1.....	C-75
C.6.1.3	Figure 2.....	C-76
C.6.2	Texas Instruments (SN 74ALS632) Chip	
C.6.2.1	Map of SN 74ALS632 Adapter.....	C-77
C.6.2.2	Figure 1.....	C-78
C.6.2.3	Figure 2.....	C-79
C.7	Results of Design Verification	
C.7.1	Parity Tree Network.....	C-80
C.7.2	Error Detector.....	C-81

C.7.3	Error Locator.....	C-82
C.7.4	Random Test.....	C-83

Map of EDAC Fault Simulator Breadboard

Fig. 1	Fig. 2	Fig. 3	Fig. 4	Fig. 5	Fig. 6	Fig. 7	Fig. 8	Fig. 9	Fig. 10	Fig. 11	Fig. 12
Fig. 13	Fig. 14	Fig. 15	Fig. 16	Fig. 17	Fig. 18	Fig. 19	Fig. 20	Fig. 21	Fig. 22	Fig. 23	Fig. 24
Fig. 25	Fig. 26	Fig. 27	Fig. 28	Fig. 29	Fig. 30	Fig. 31	Fig. 32	Fig. 33	Fig. 34	Fig. 35	Fig. 36
Fig. 37	Fig. 38	Fig. 39	Fig. 40	Fig. 41	Fig. 42	Fig. 43	Fig. 44	Fig. 45	Fig. 46	Fig. 47	Fig. 48
Fig. 49	Fig. 50	Fig. 51	Fig. 52	Fig. 53	Fig. 54	Fig. 55	Fig. 56	Fig. 57	Fig. 58	Fig. 59	Fig. 60

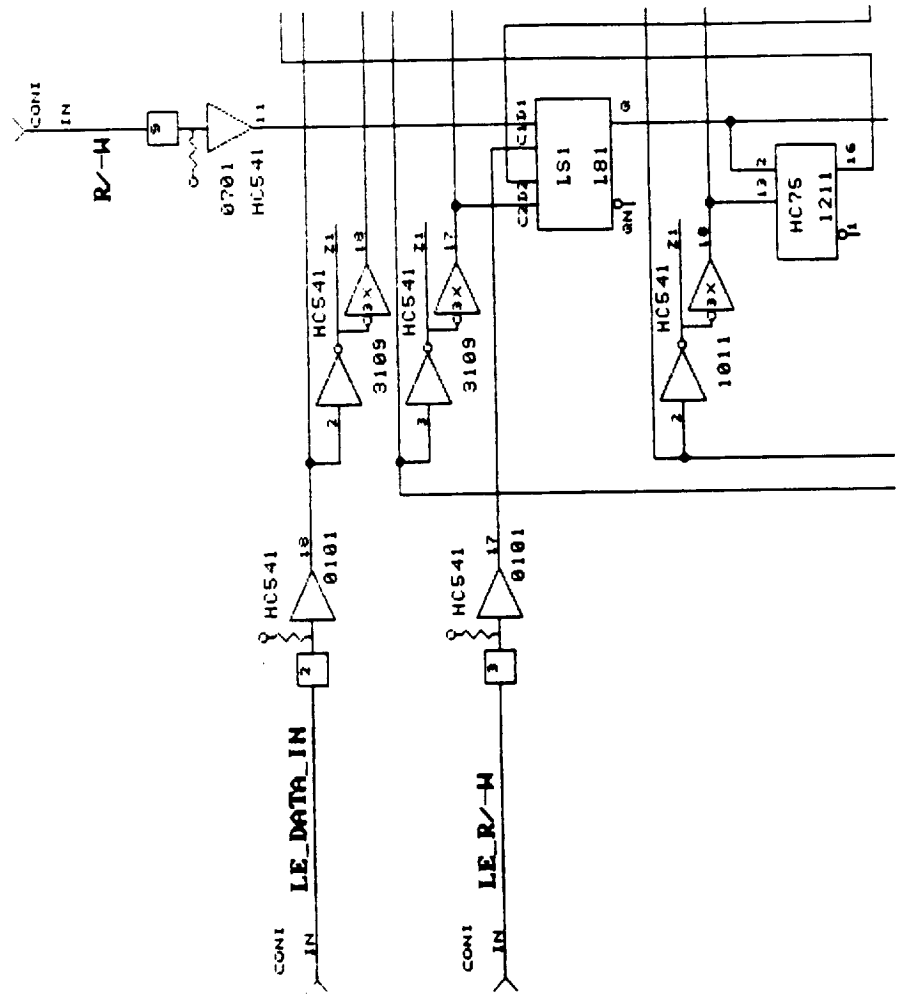


Figure 1

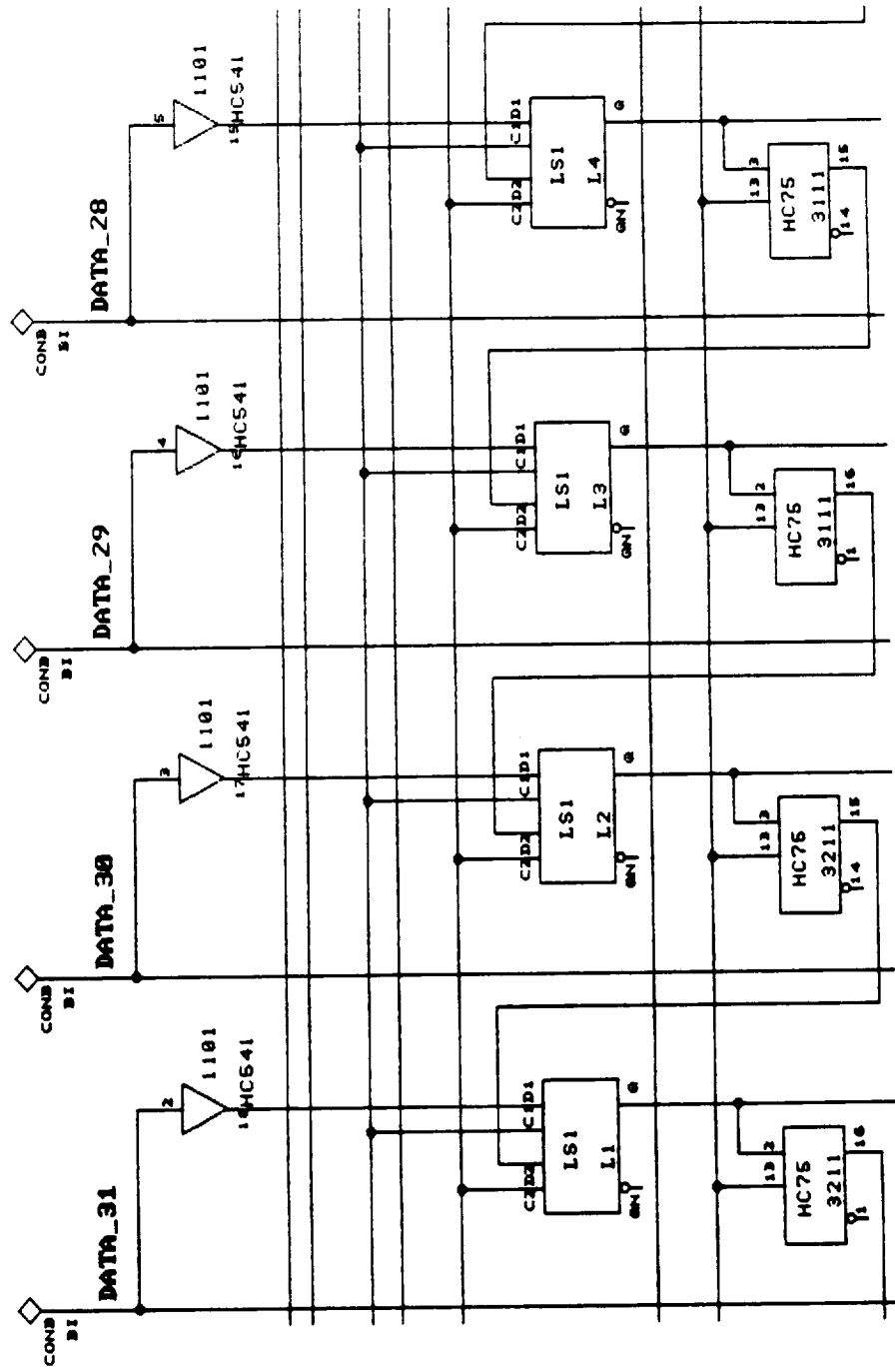


Figure 2

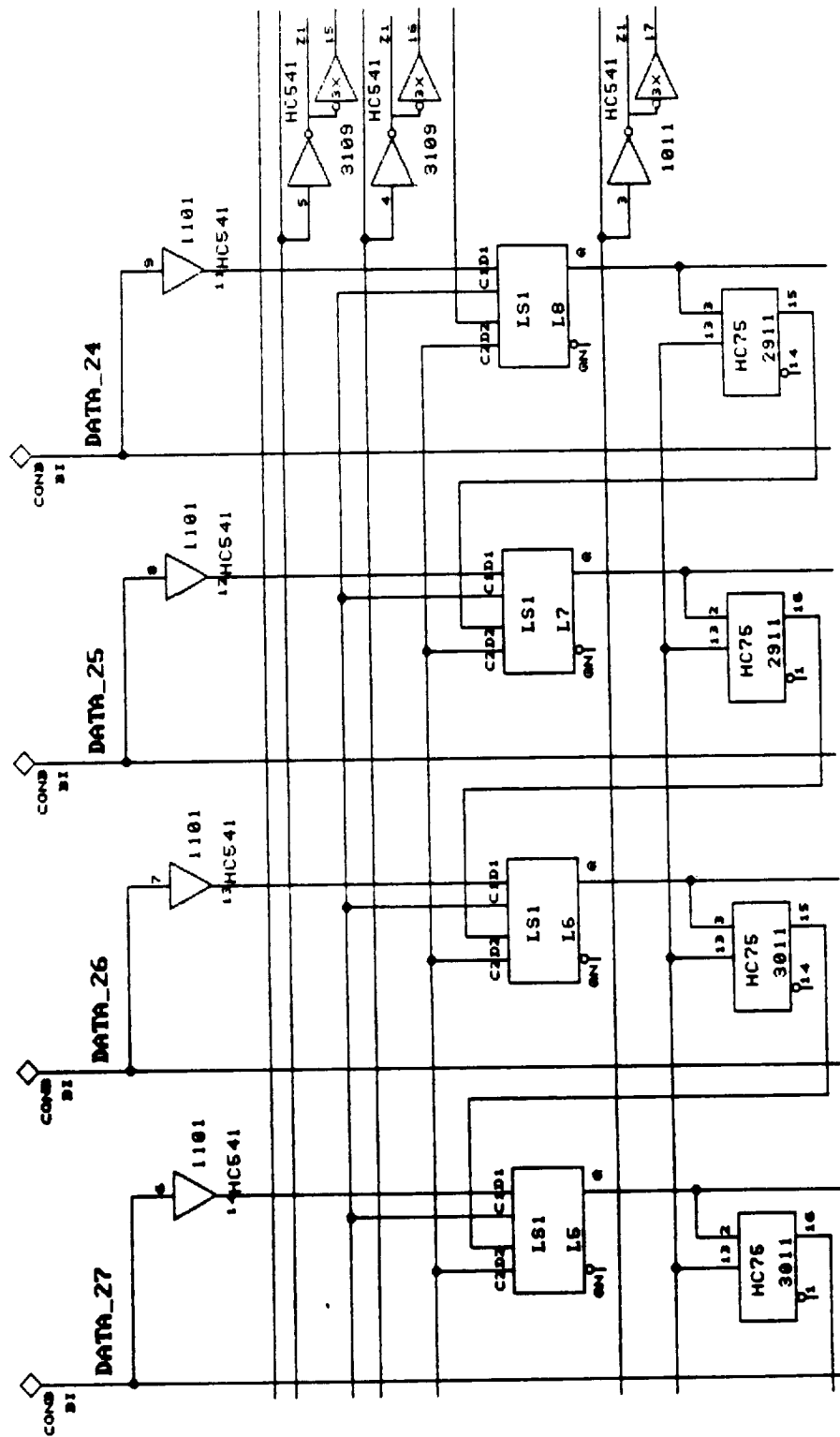
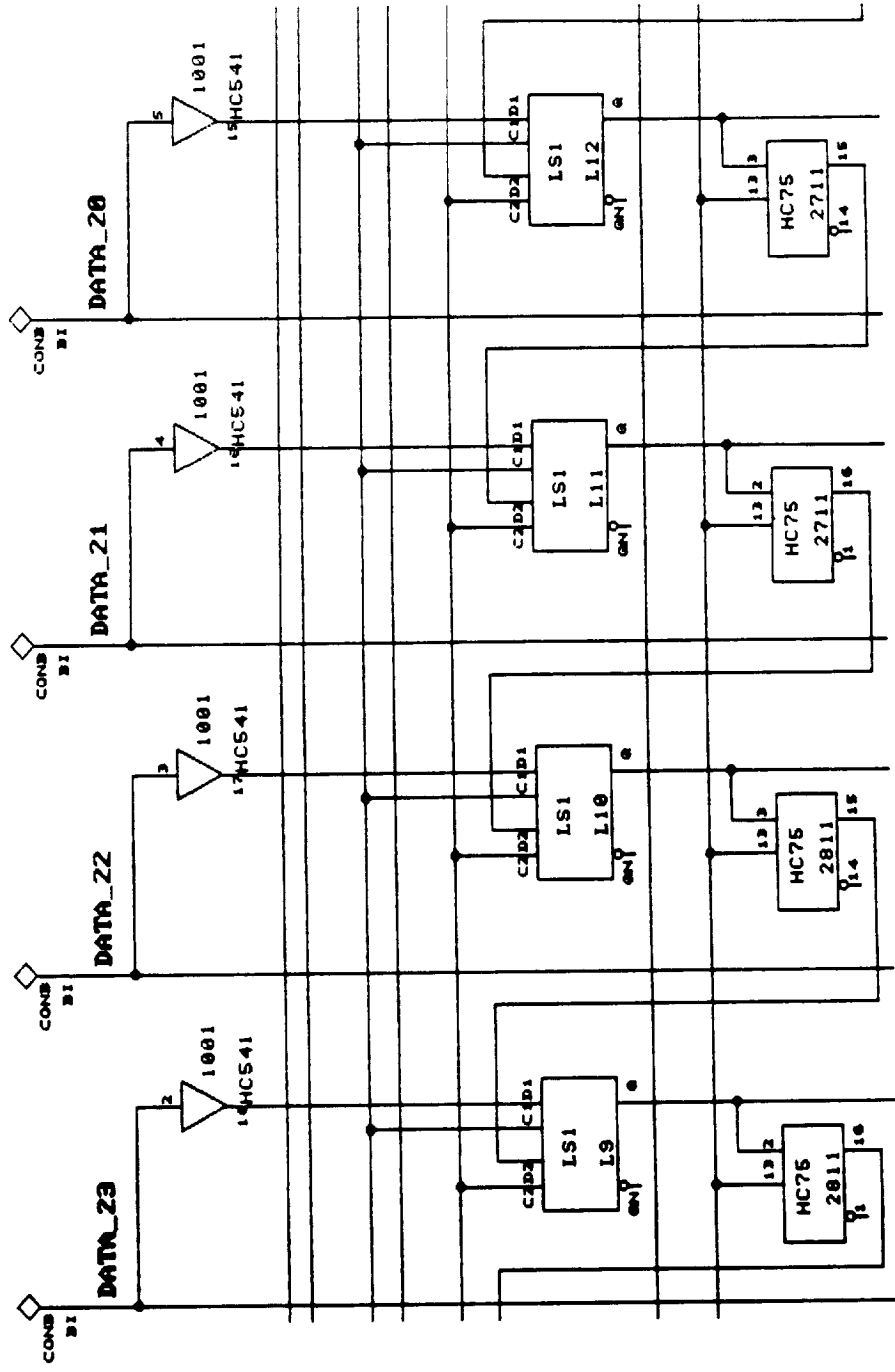


Figure 3



C-5
64

Figure 4

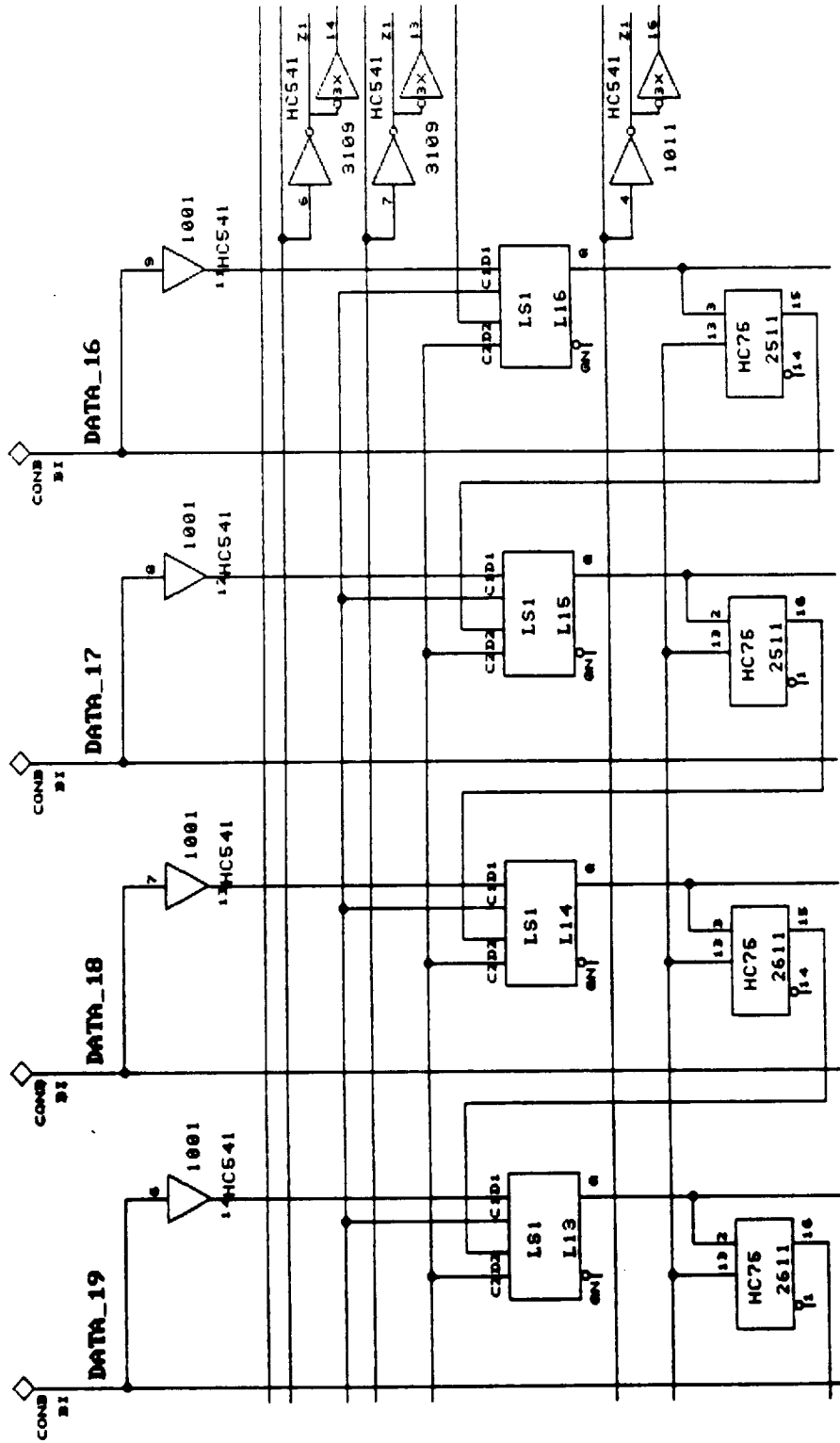


Figure 5

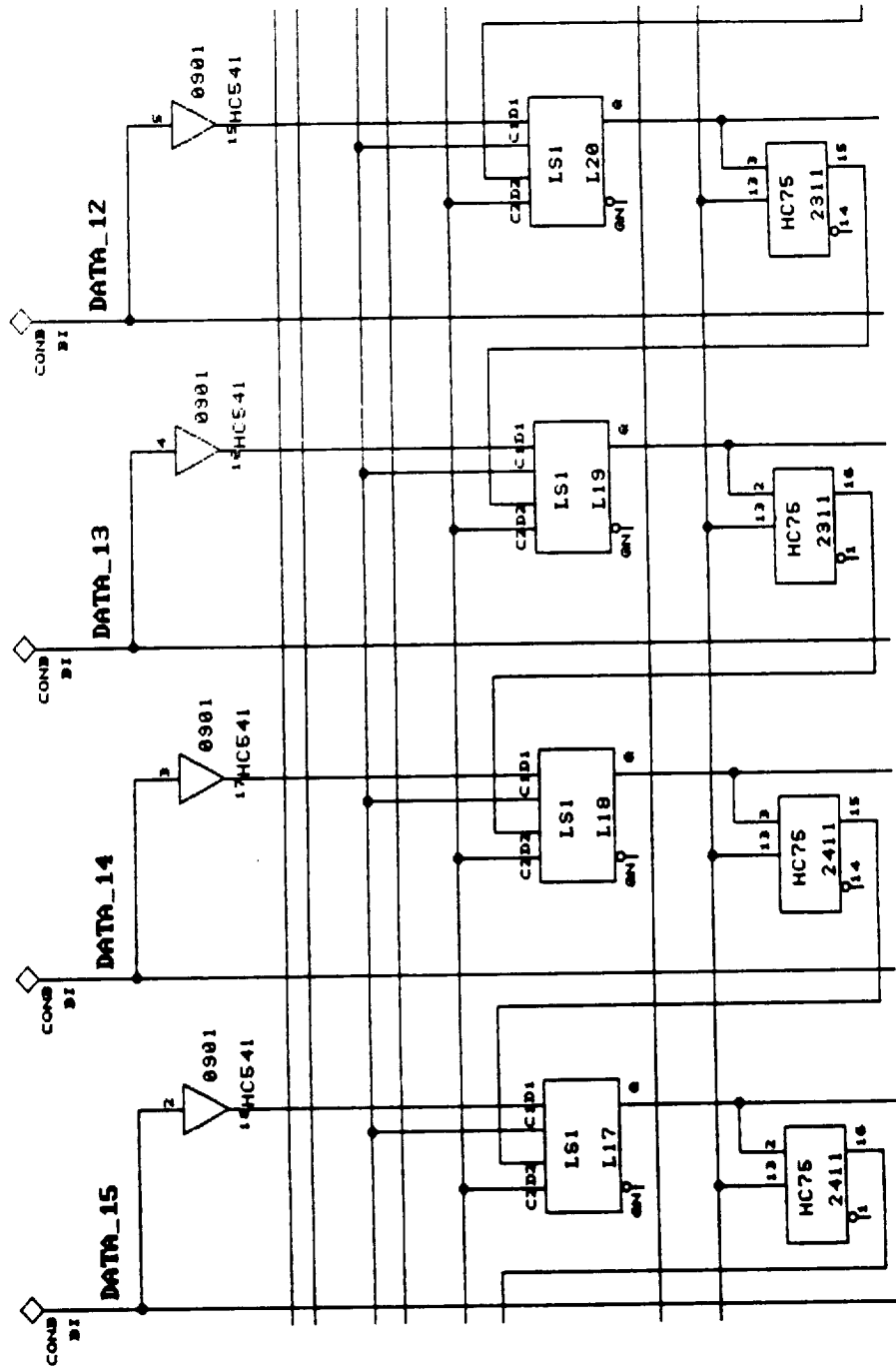


Figure 6

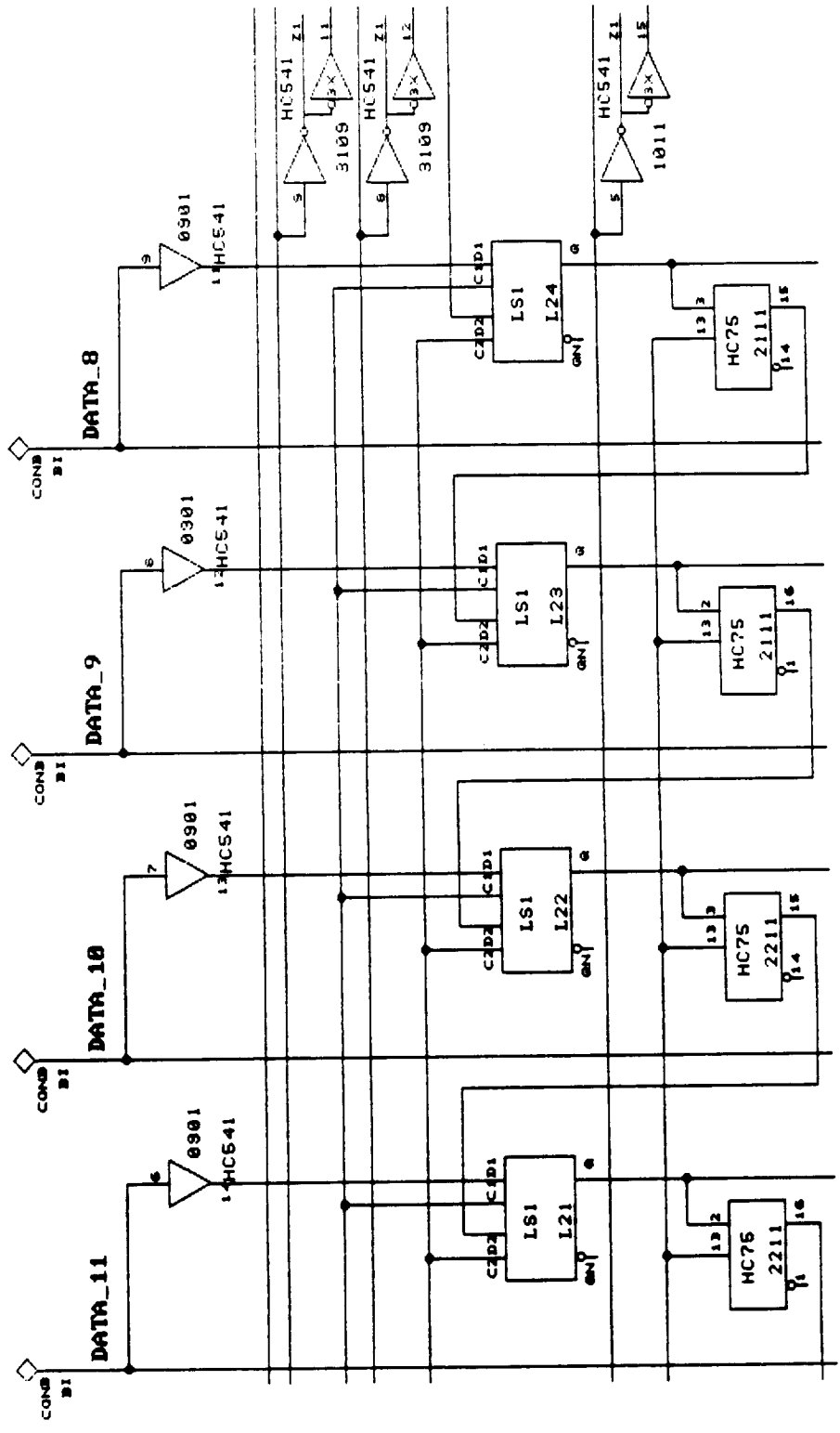


Figure 7

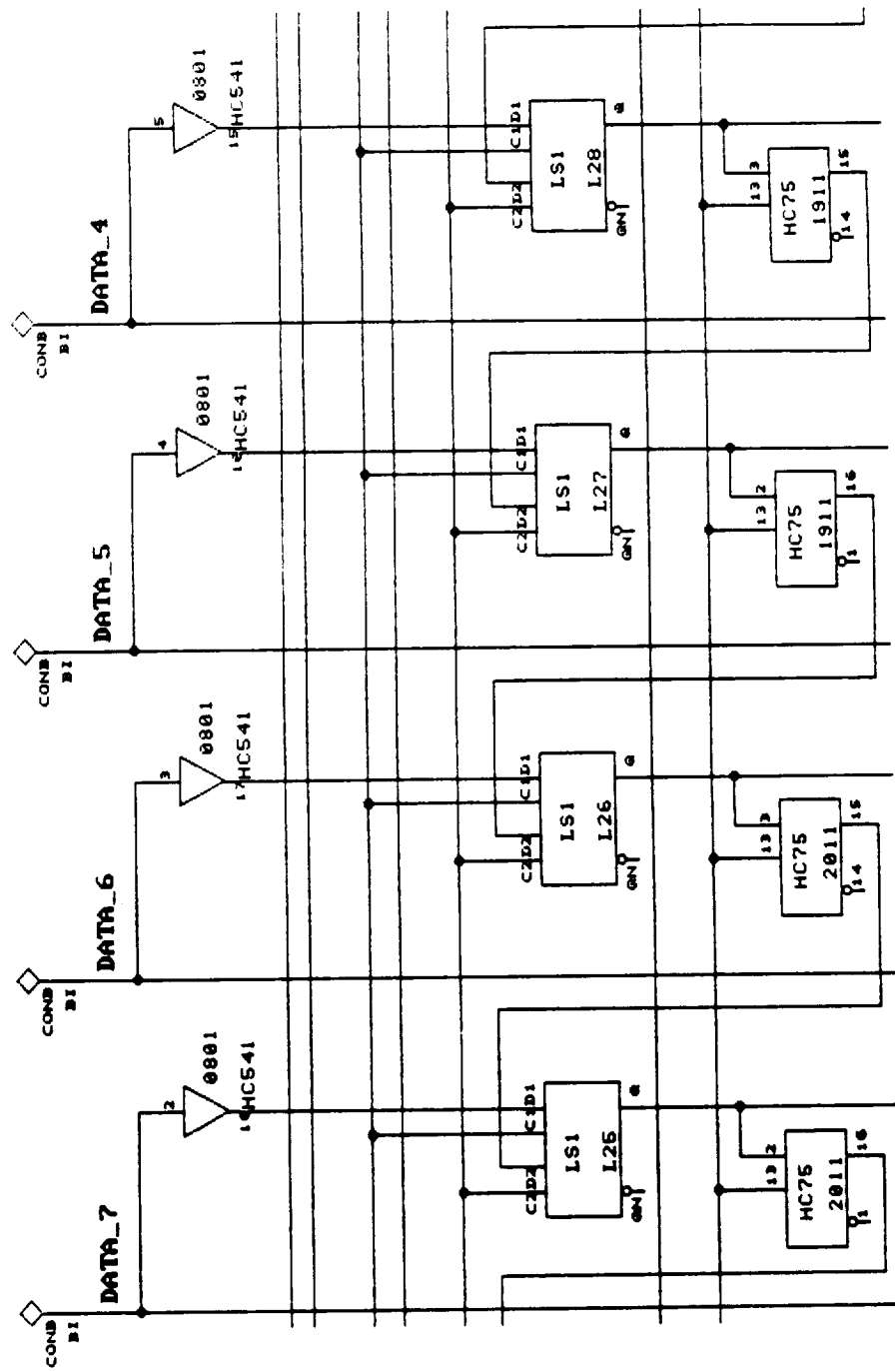


Figure 8

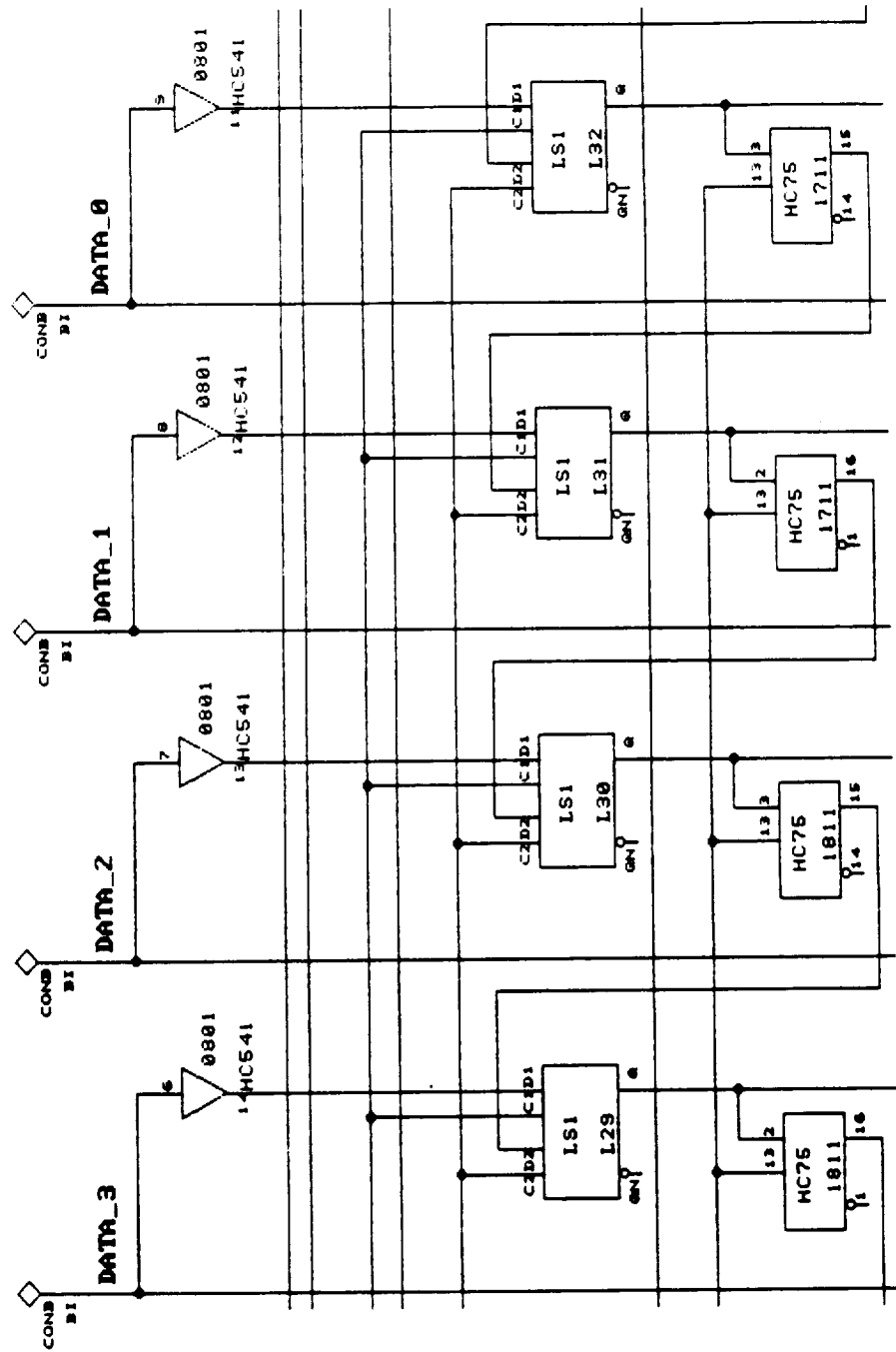


Figure 9

ORIGINAL PAGE IS
OF POOR QUALITY

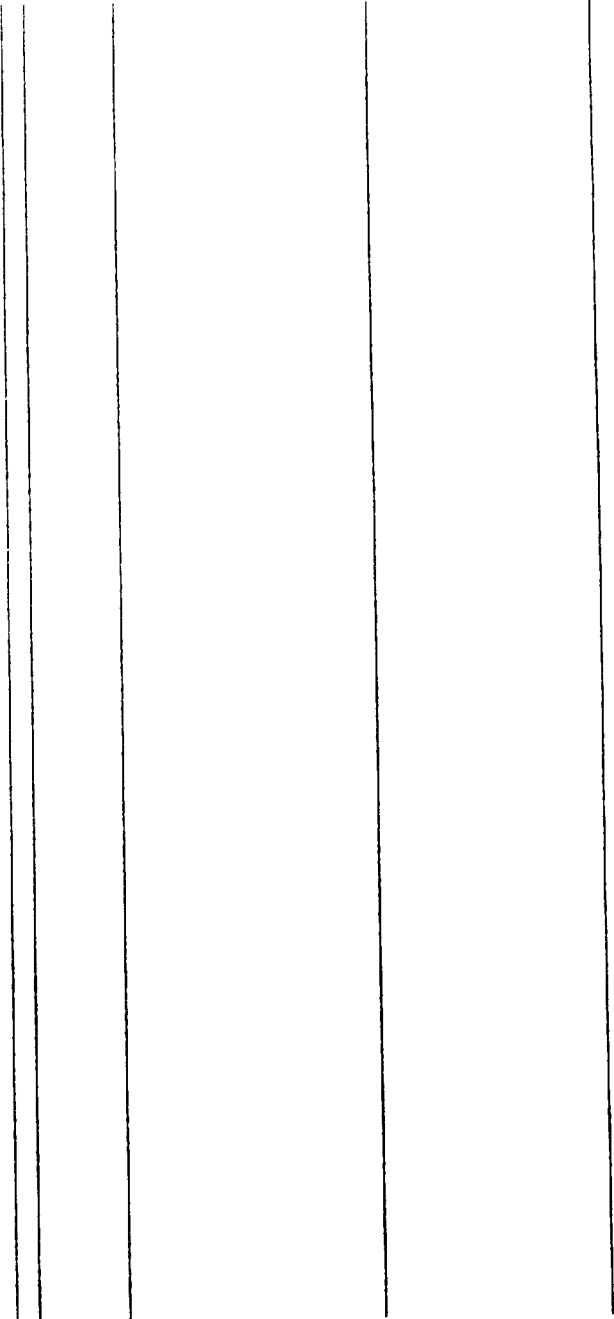


Figure 10

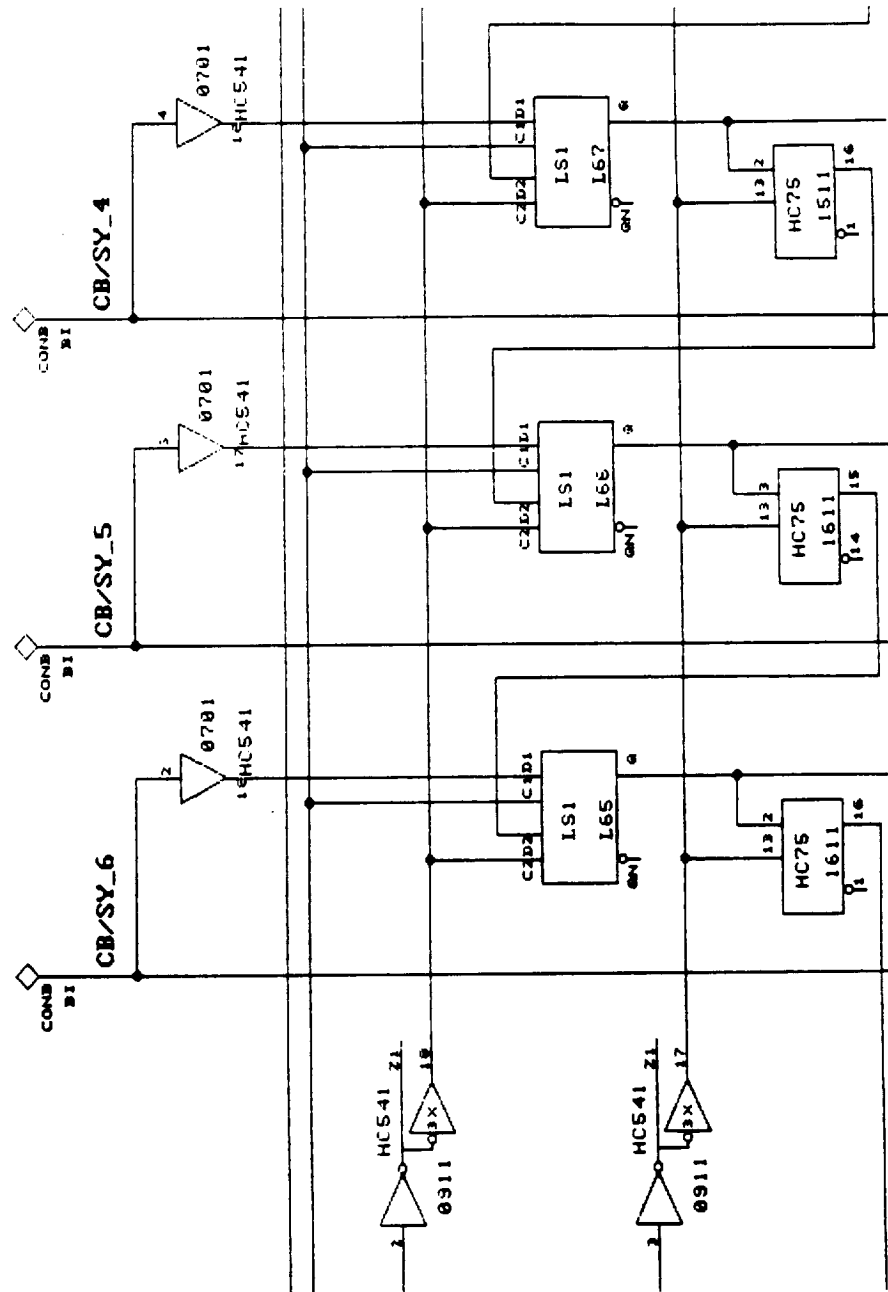


Figure 11

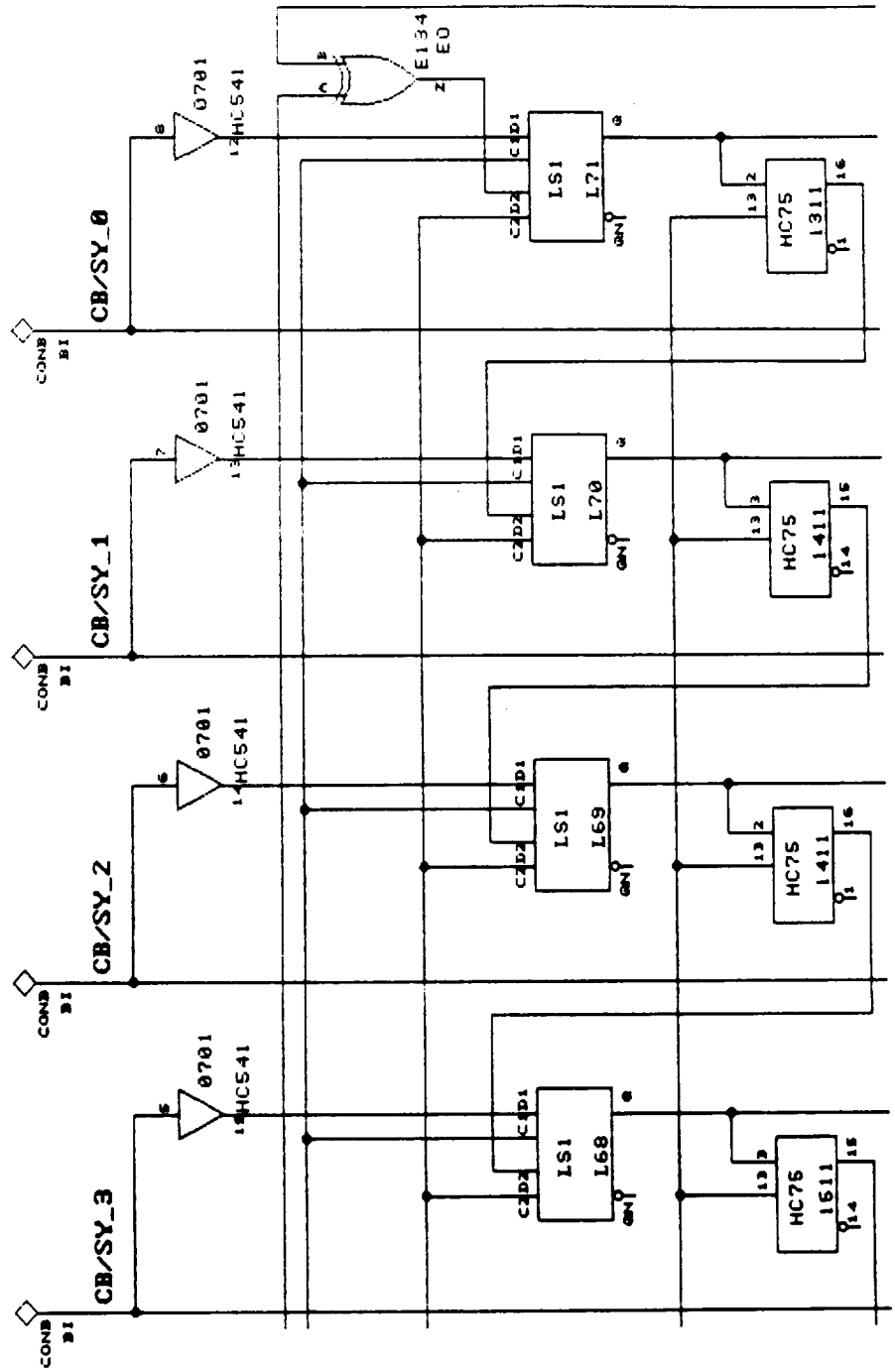


Figure 12

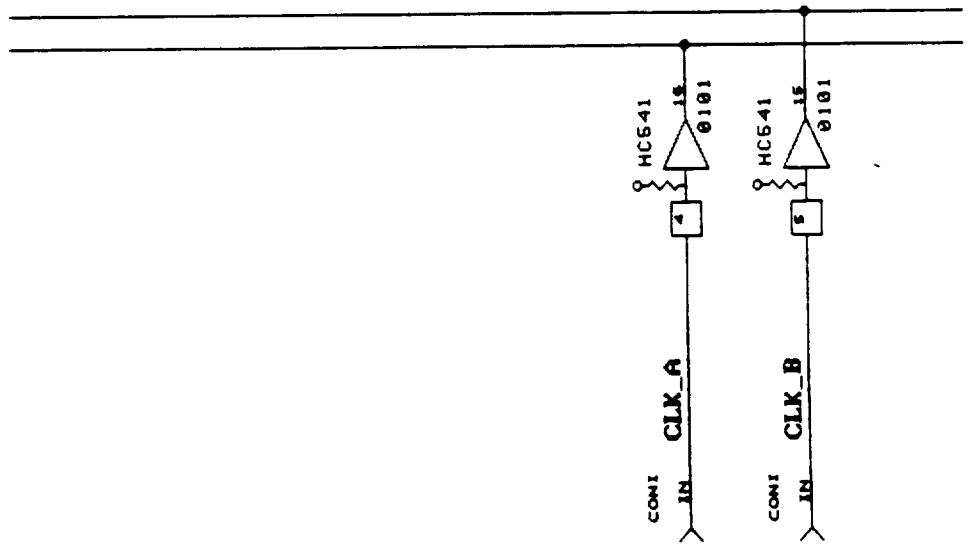
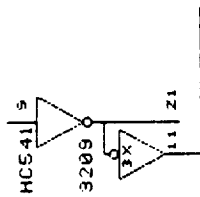


Figure 13

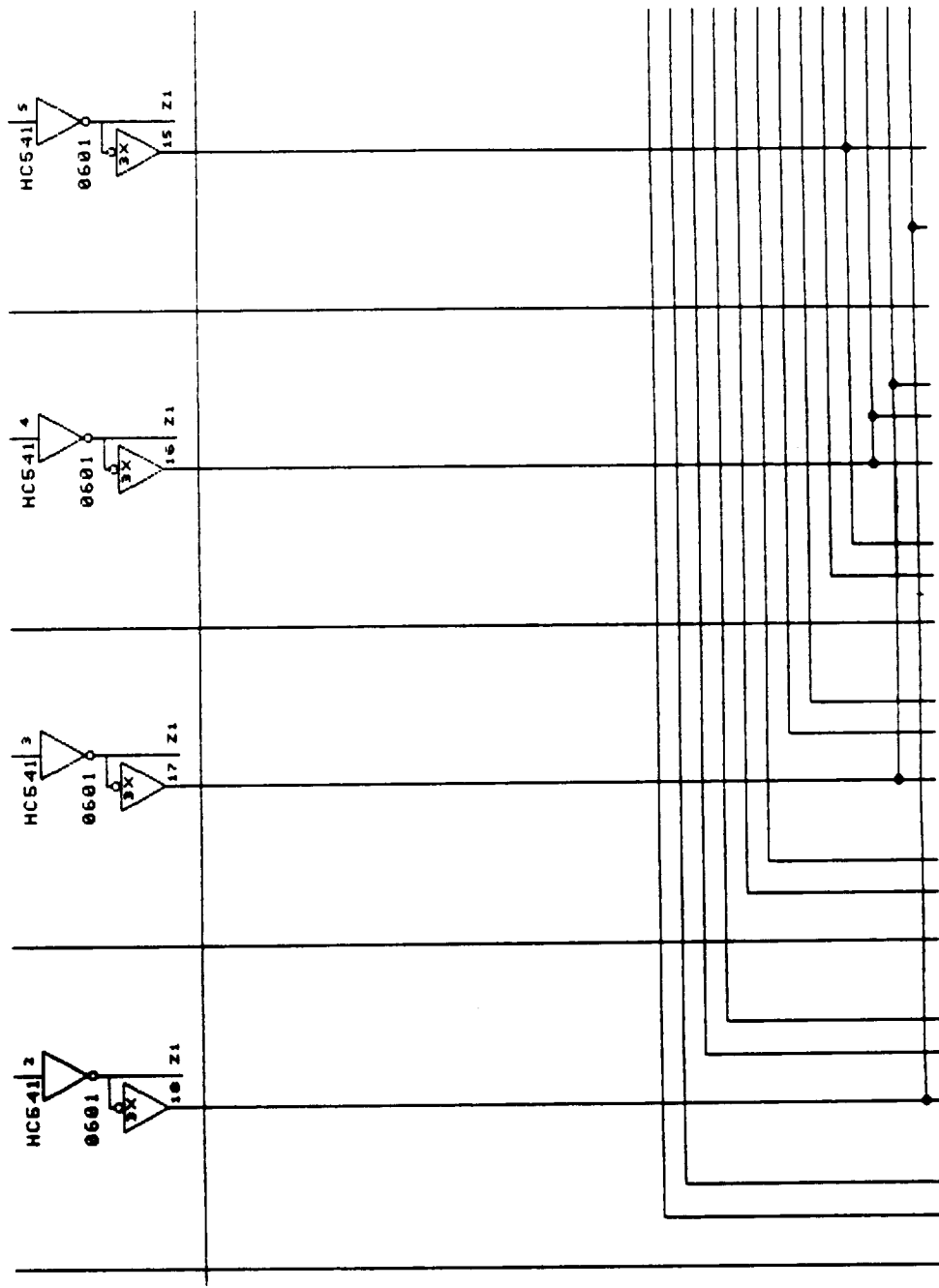


Figure 14

60 3 7
0 0 0 1 1

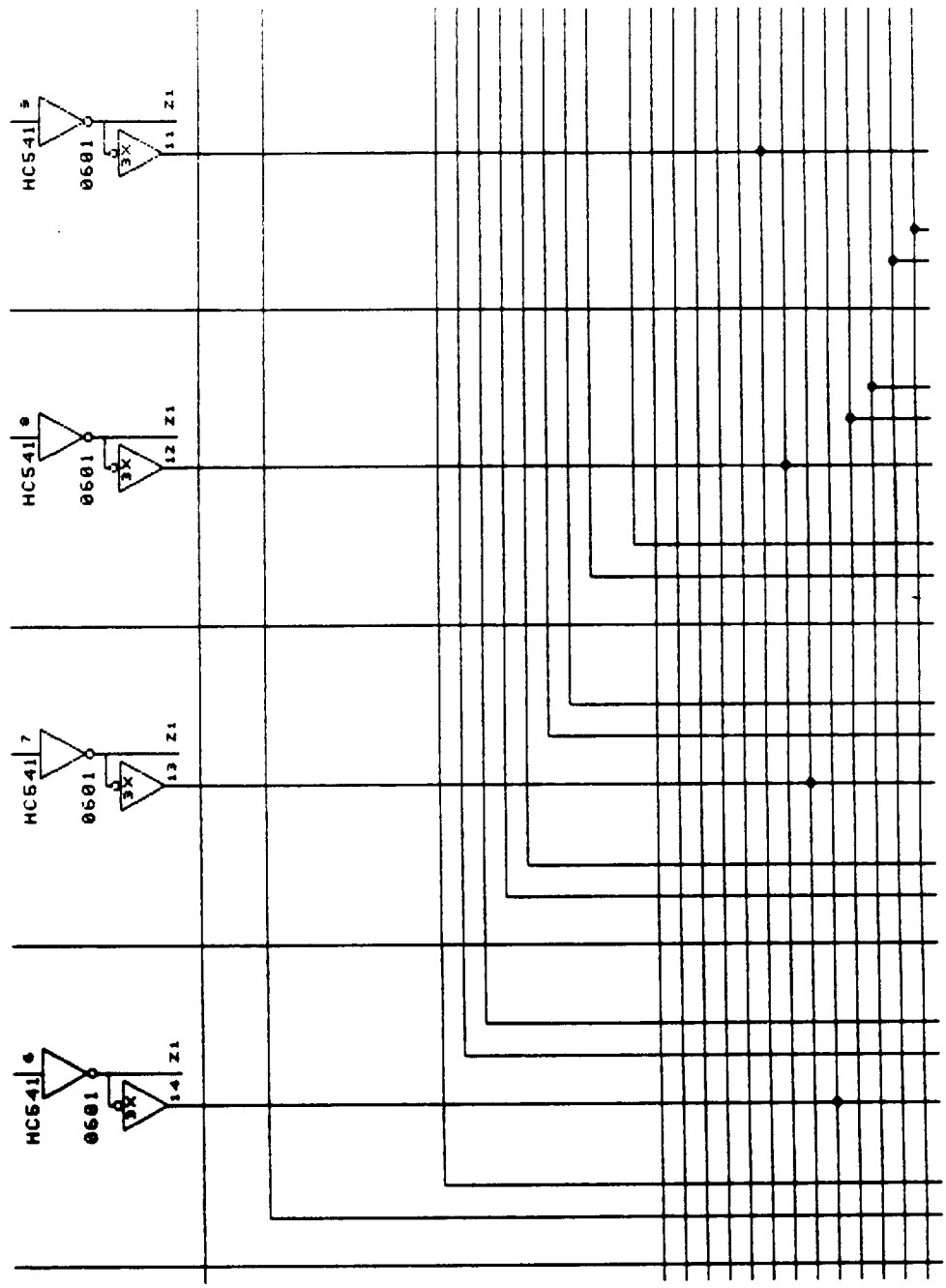


Figure 15

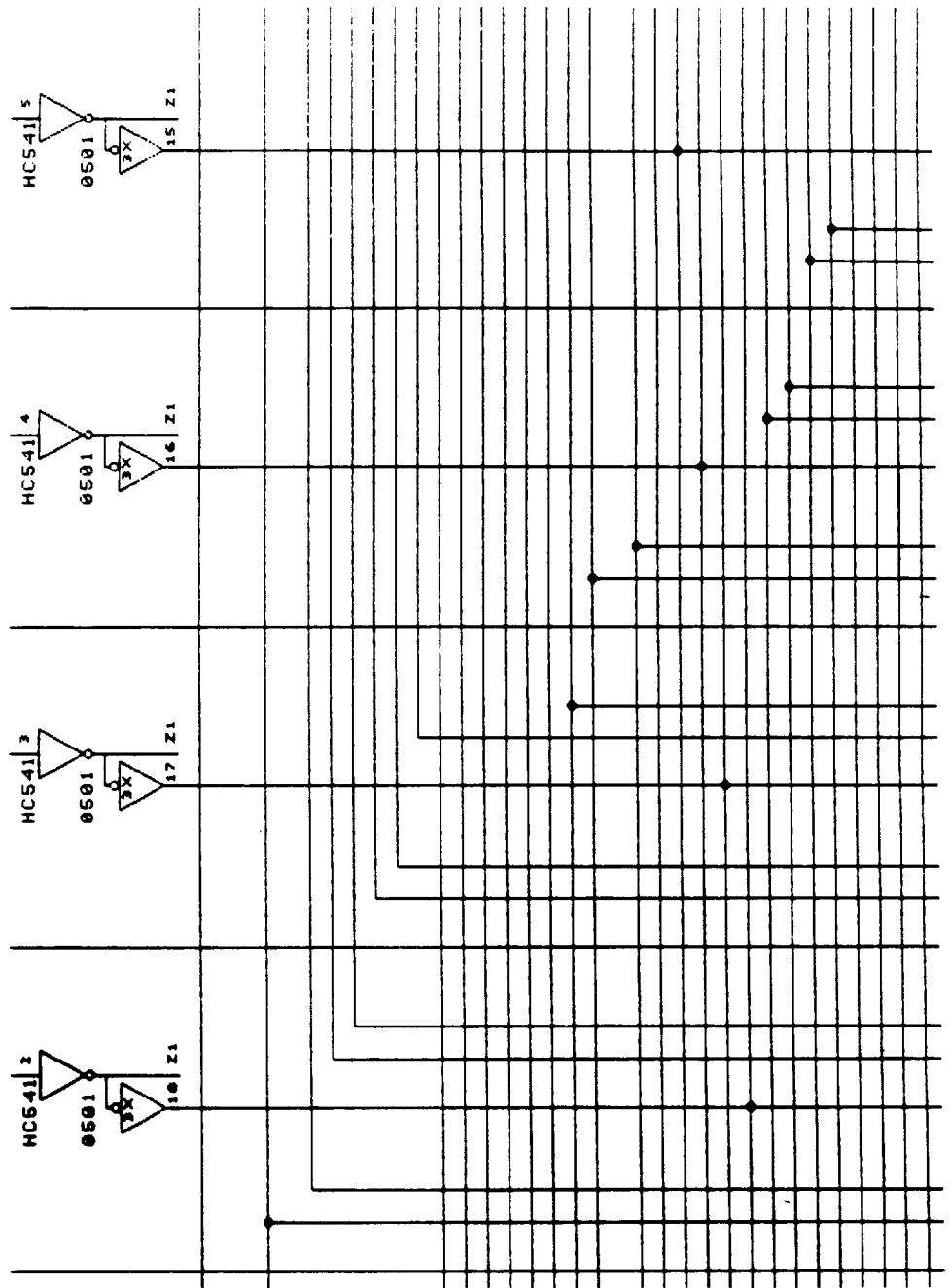


Figure 16

8163

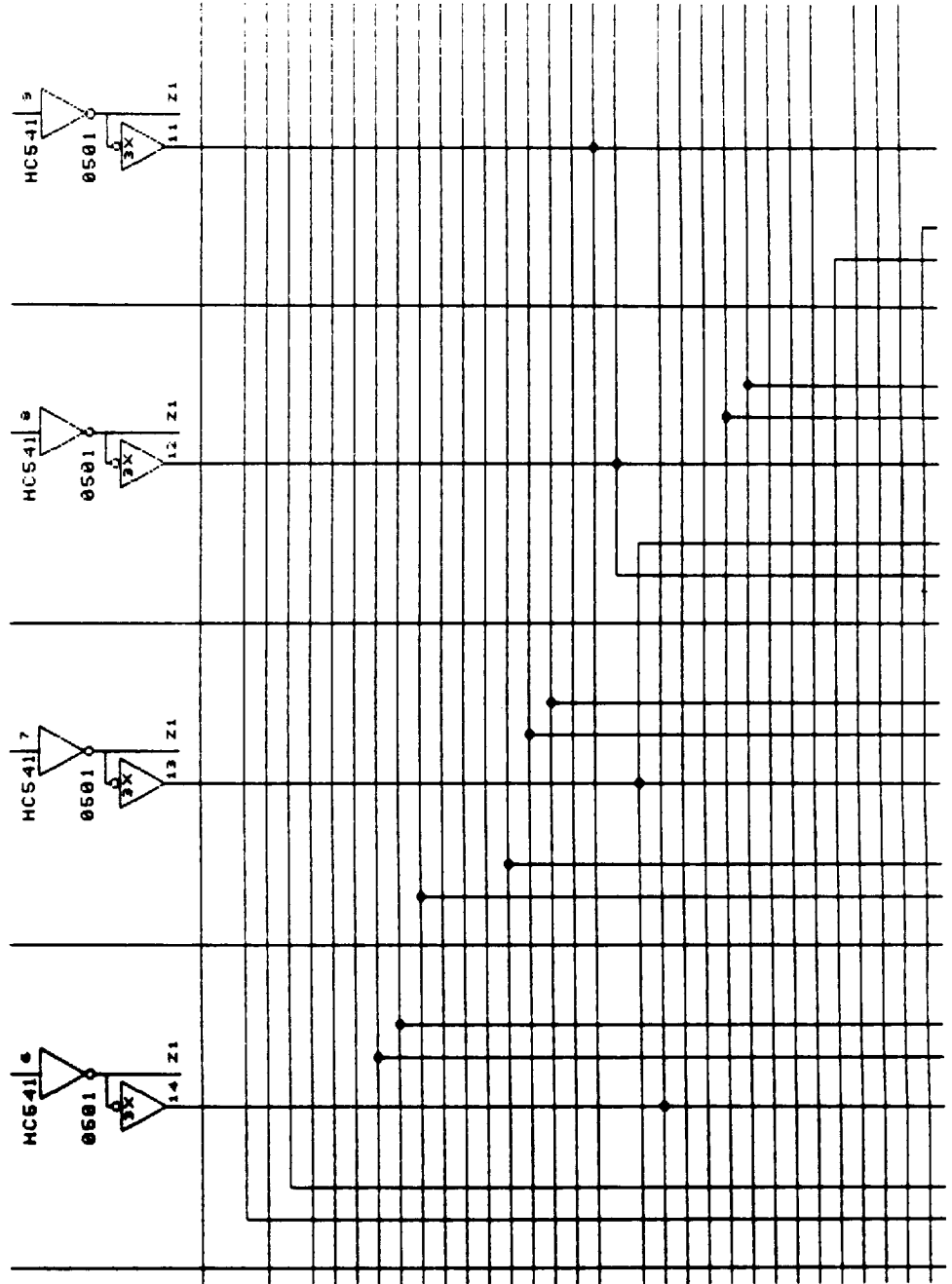


Figure 17

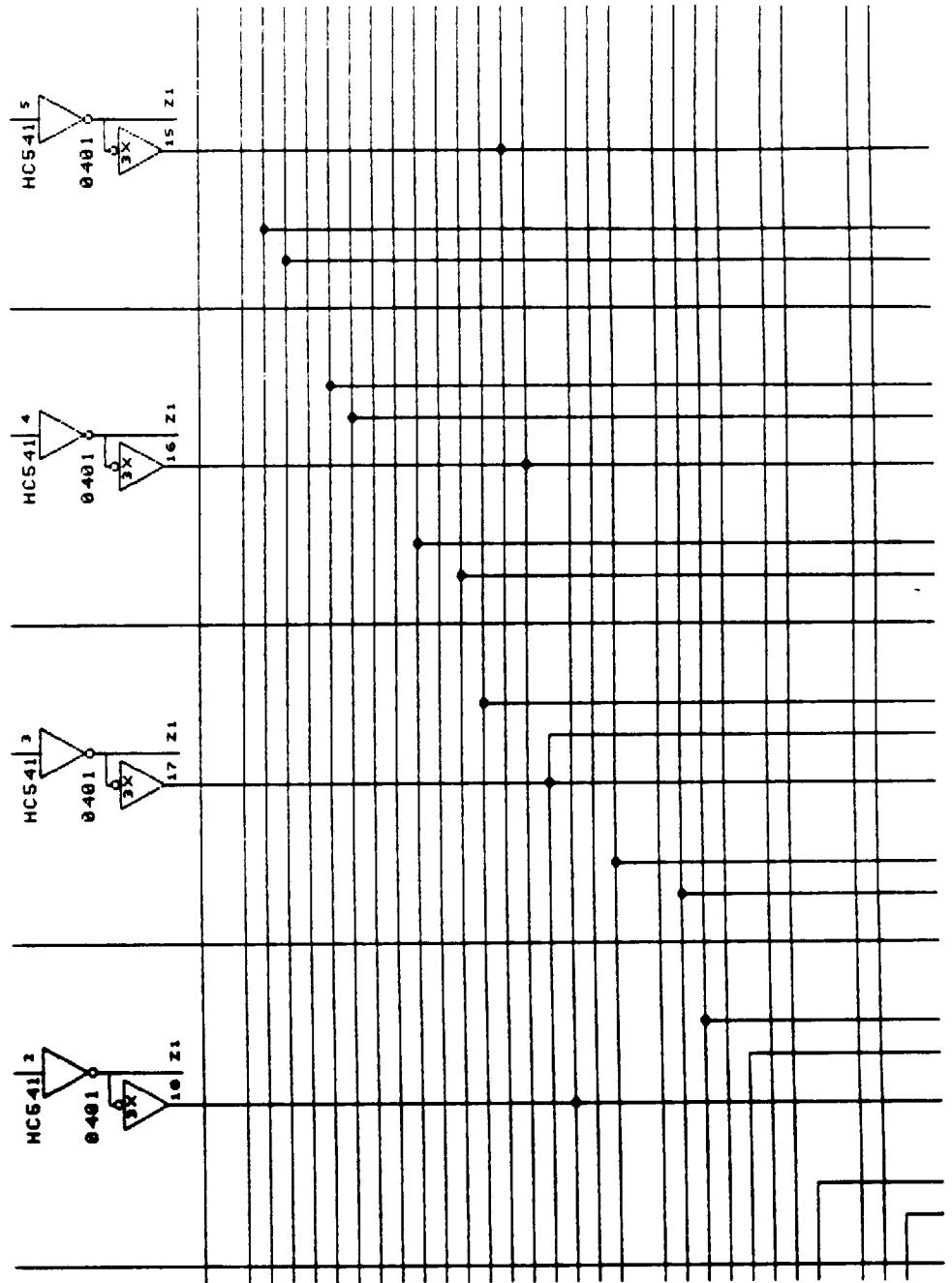


Figure 18

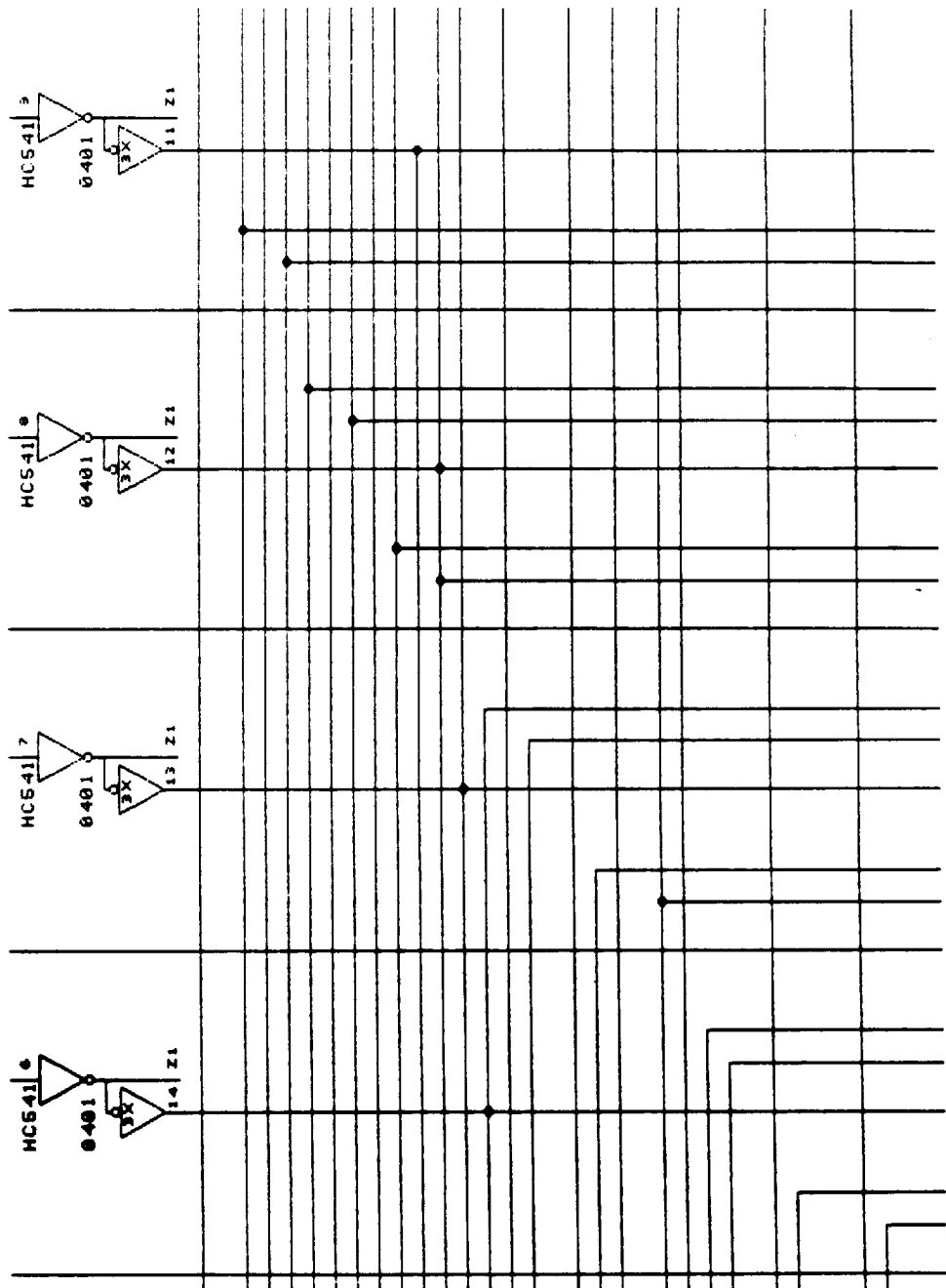


Figure 19

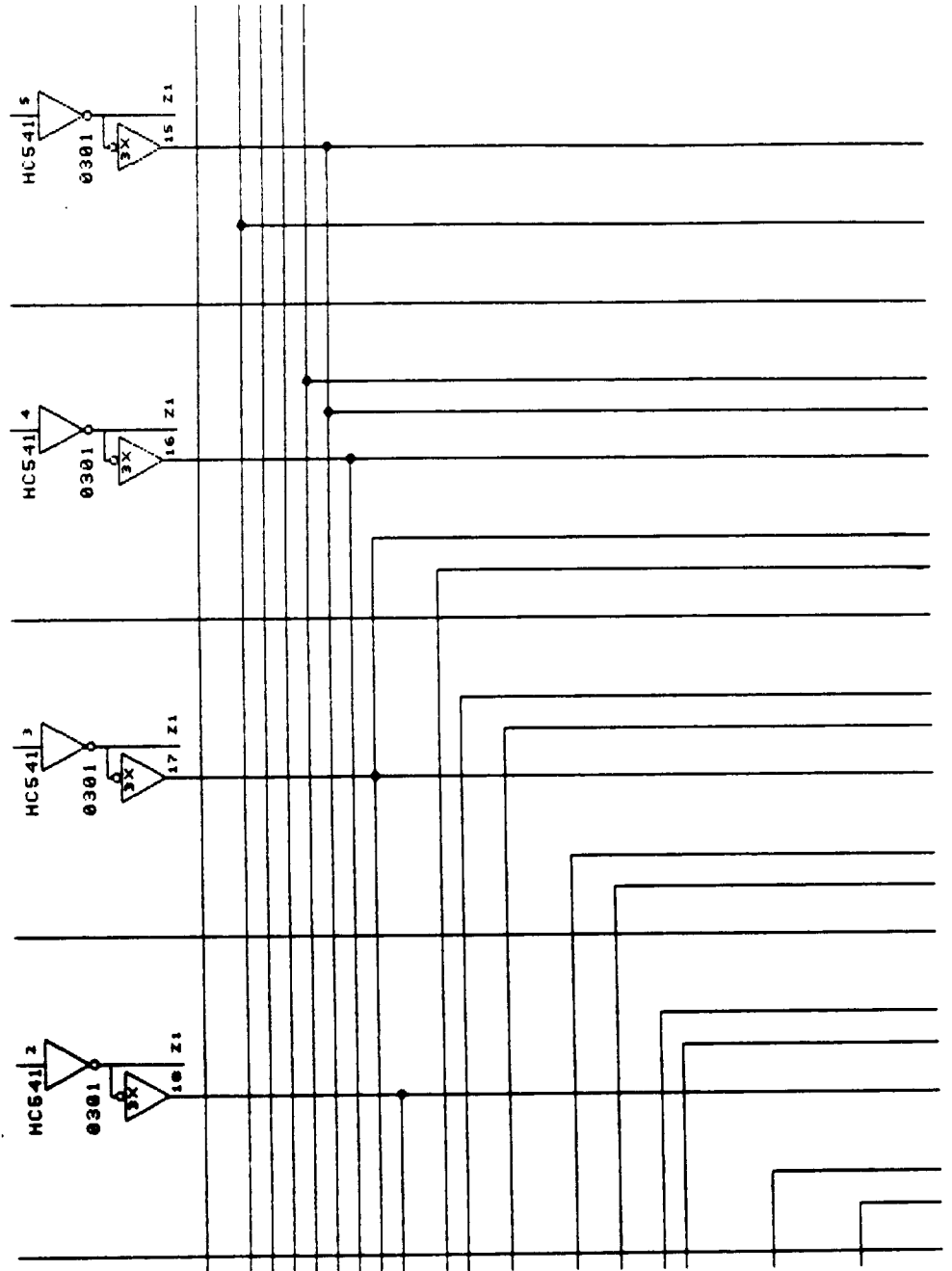


Figure 20

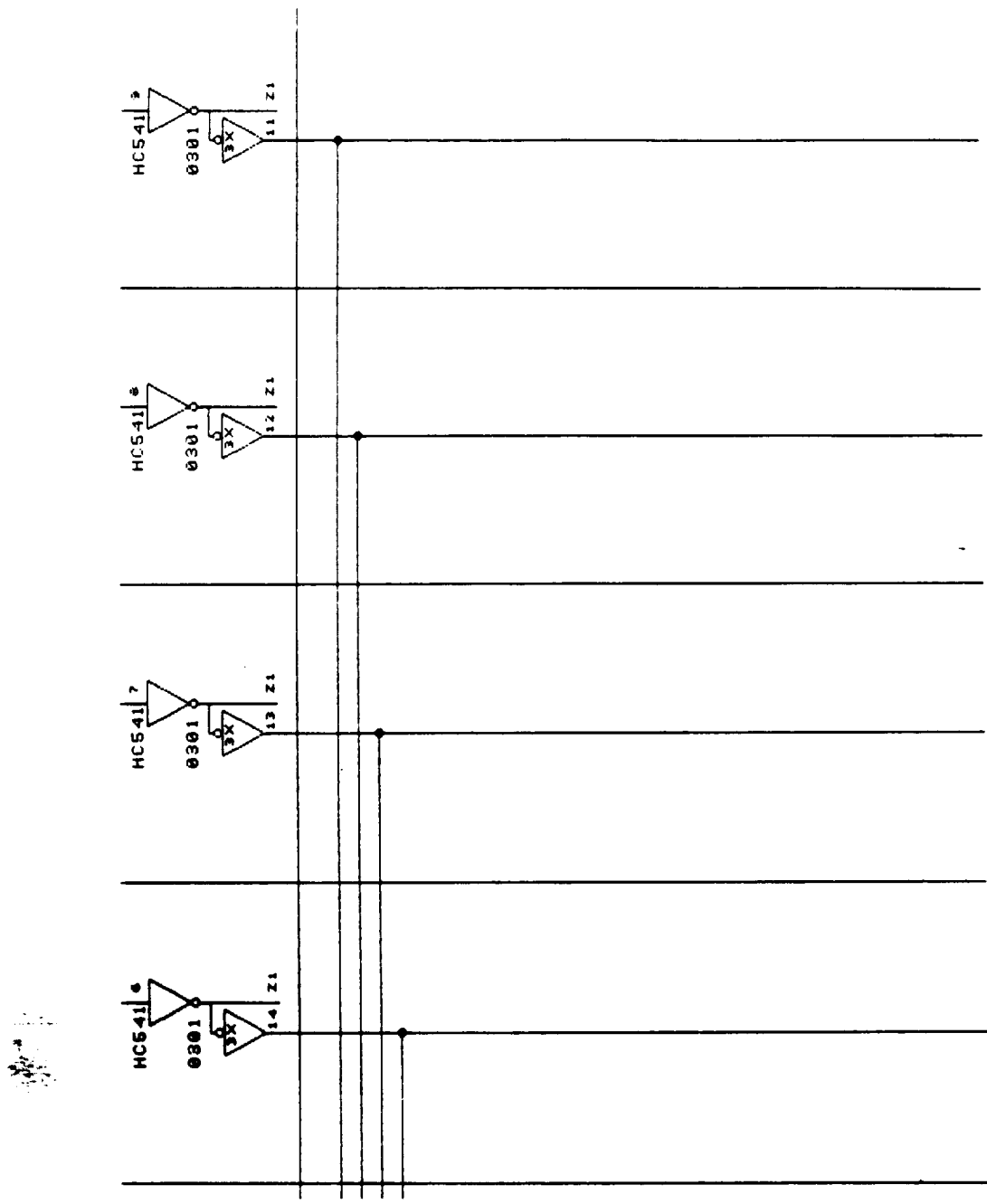


Figure 21

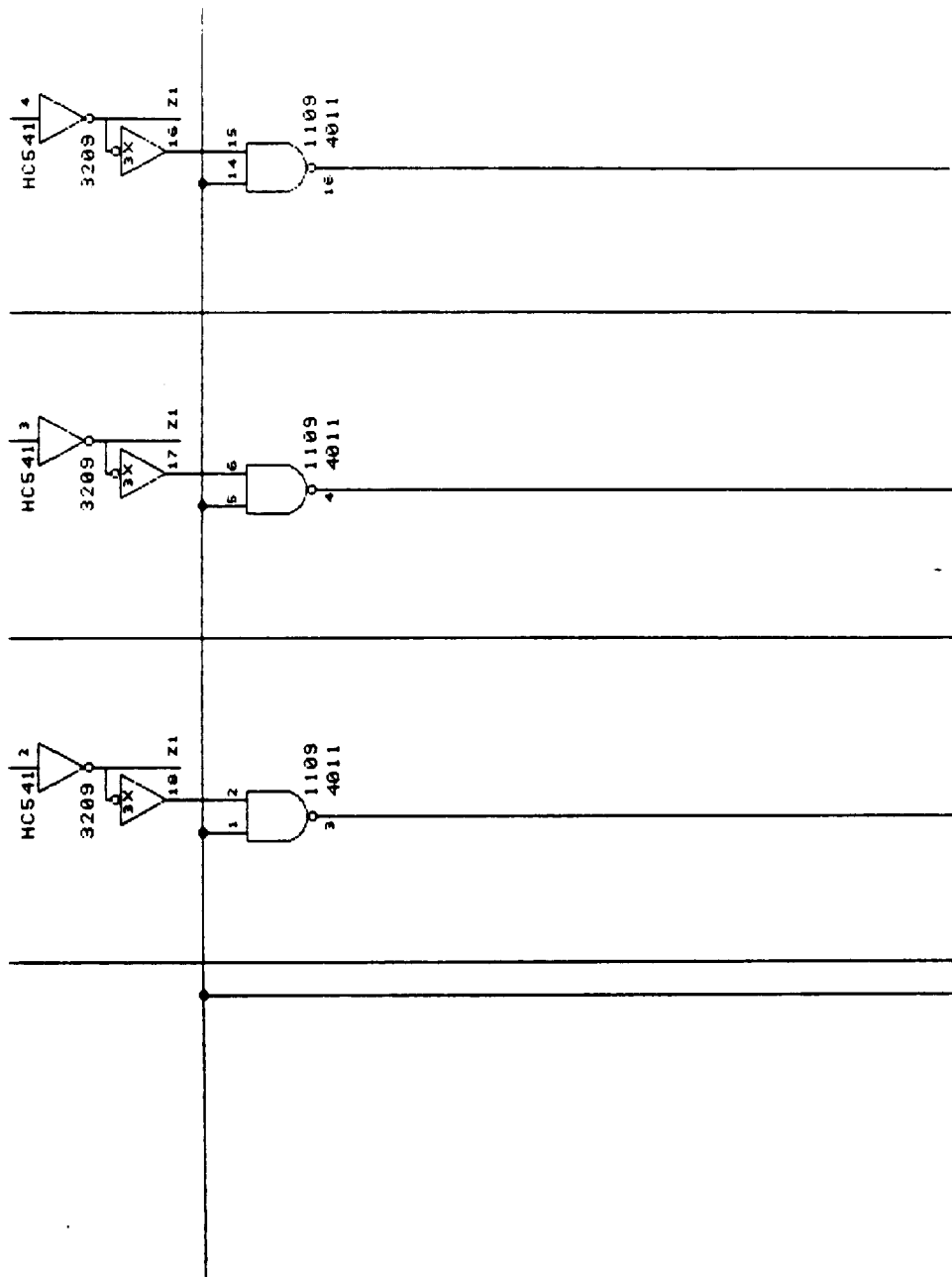


Figure 23

84

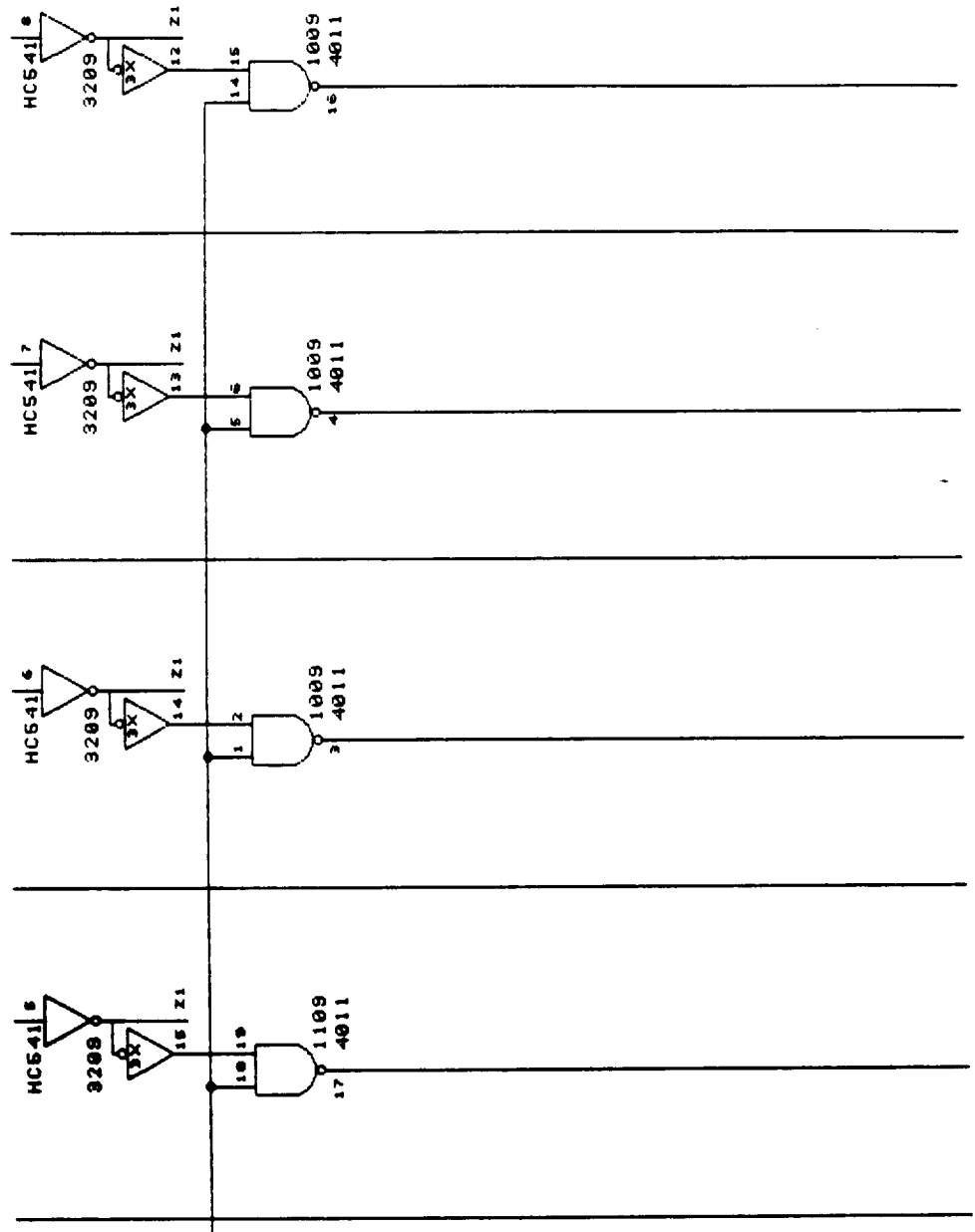


Figure 24

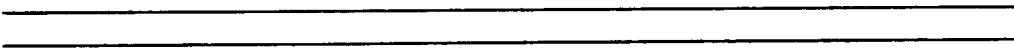


Figure 25

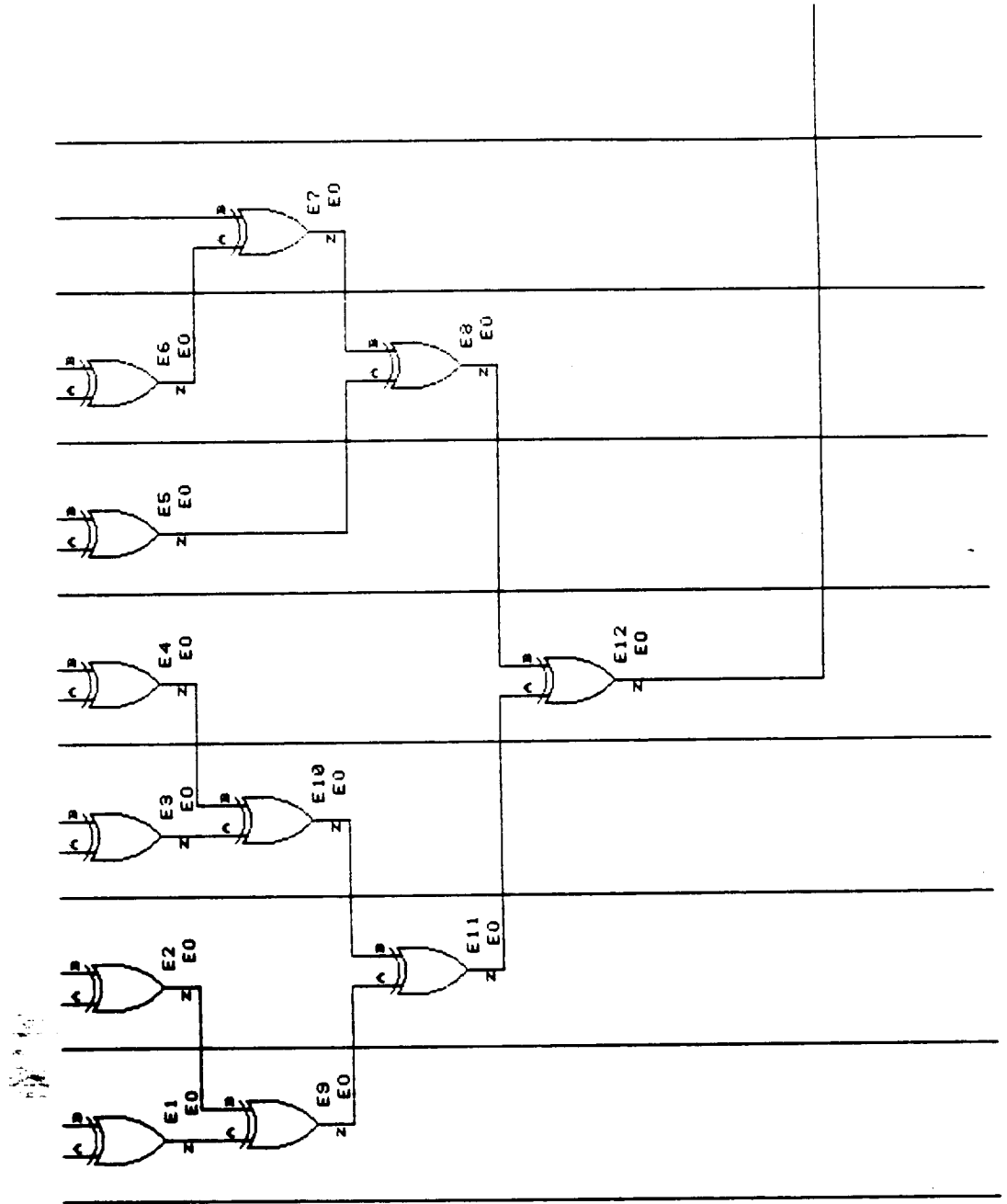


Figure 26

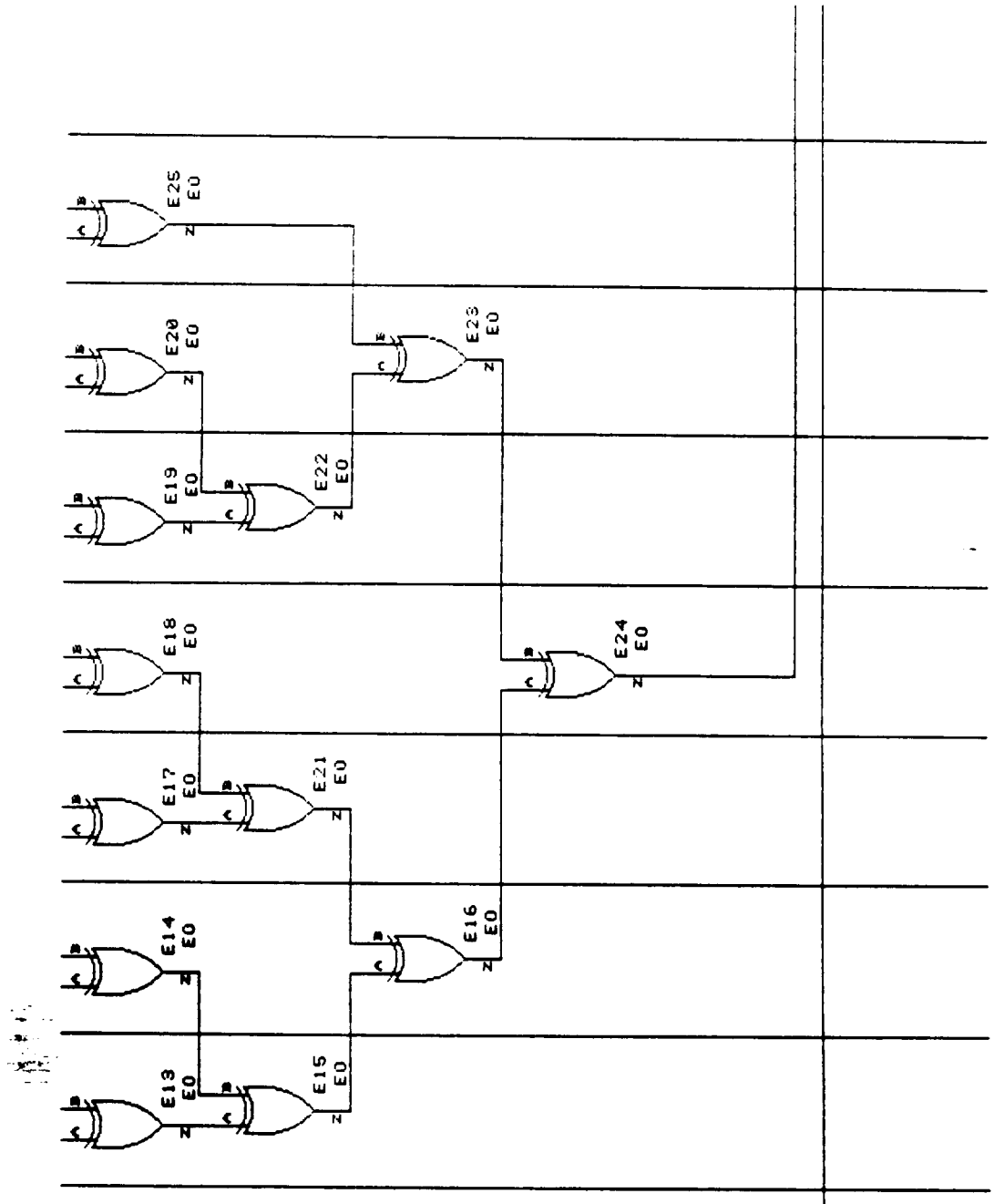


Figure 27

ERROR DETECTION AND CORRECTION UNIT WITH BUILT-IN SELF
TEST CAPABILITY FOR SPACECRAFT APPLICATIONS

NAS7-1028

PART 2

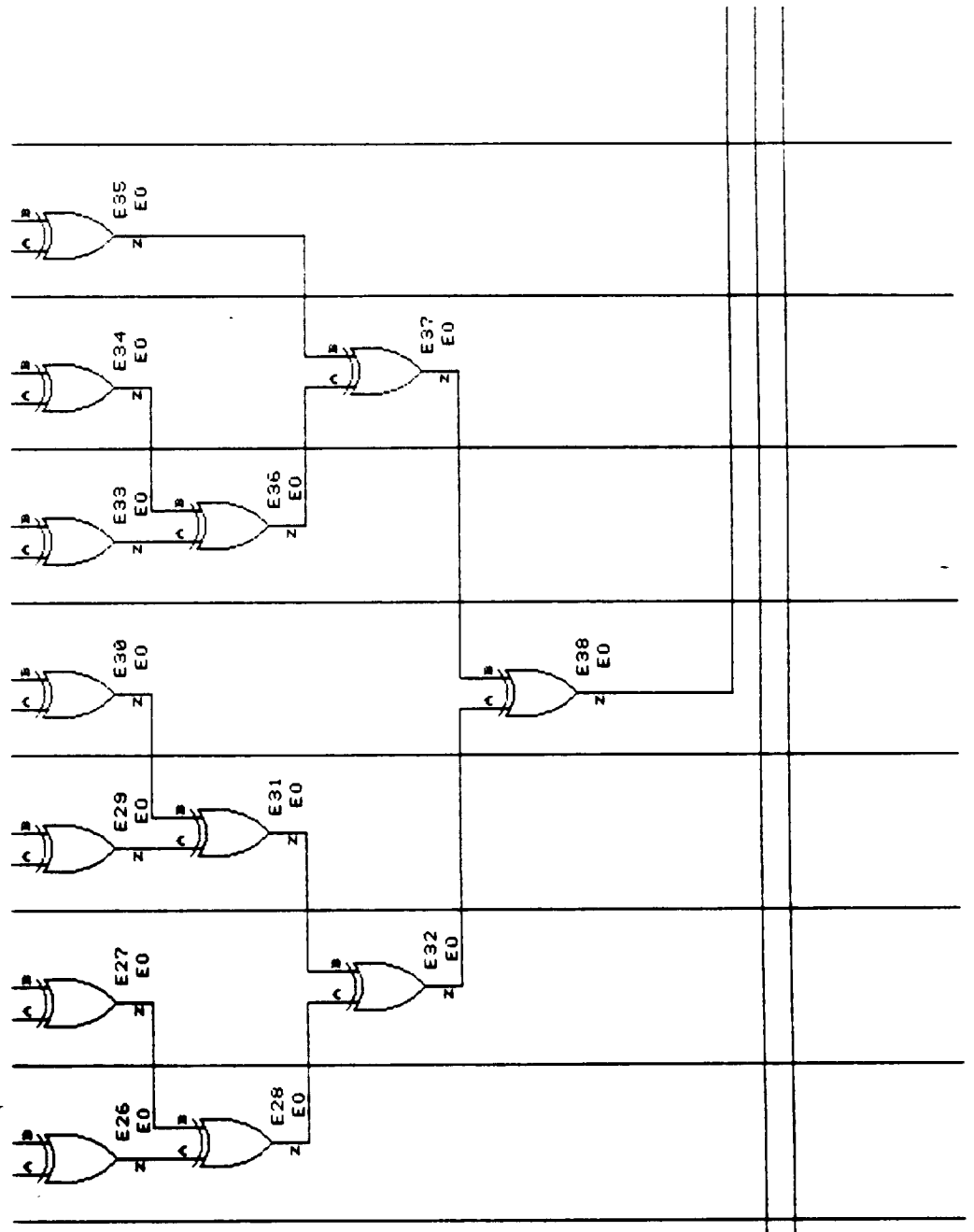


Figure 78

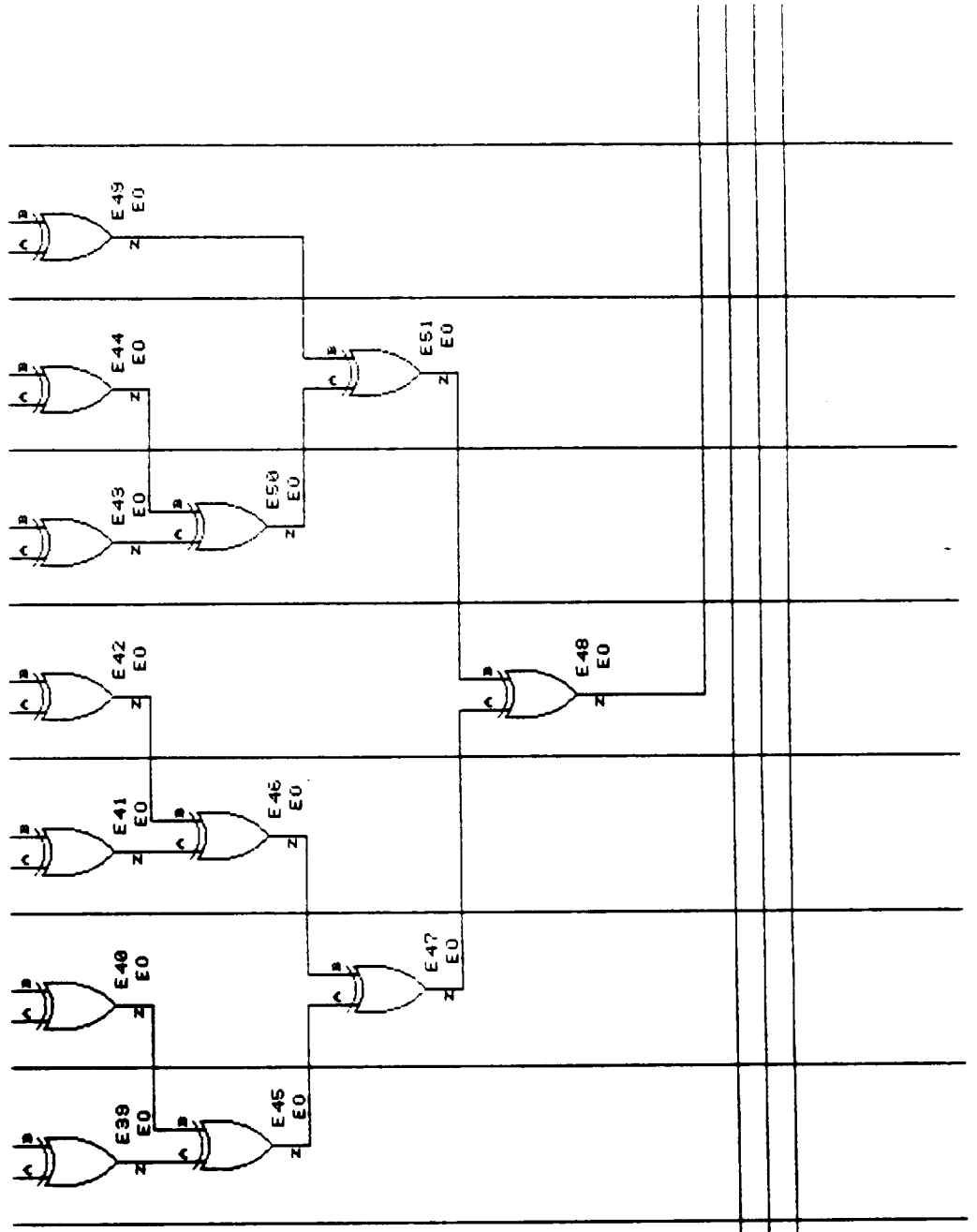


Figure 29

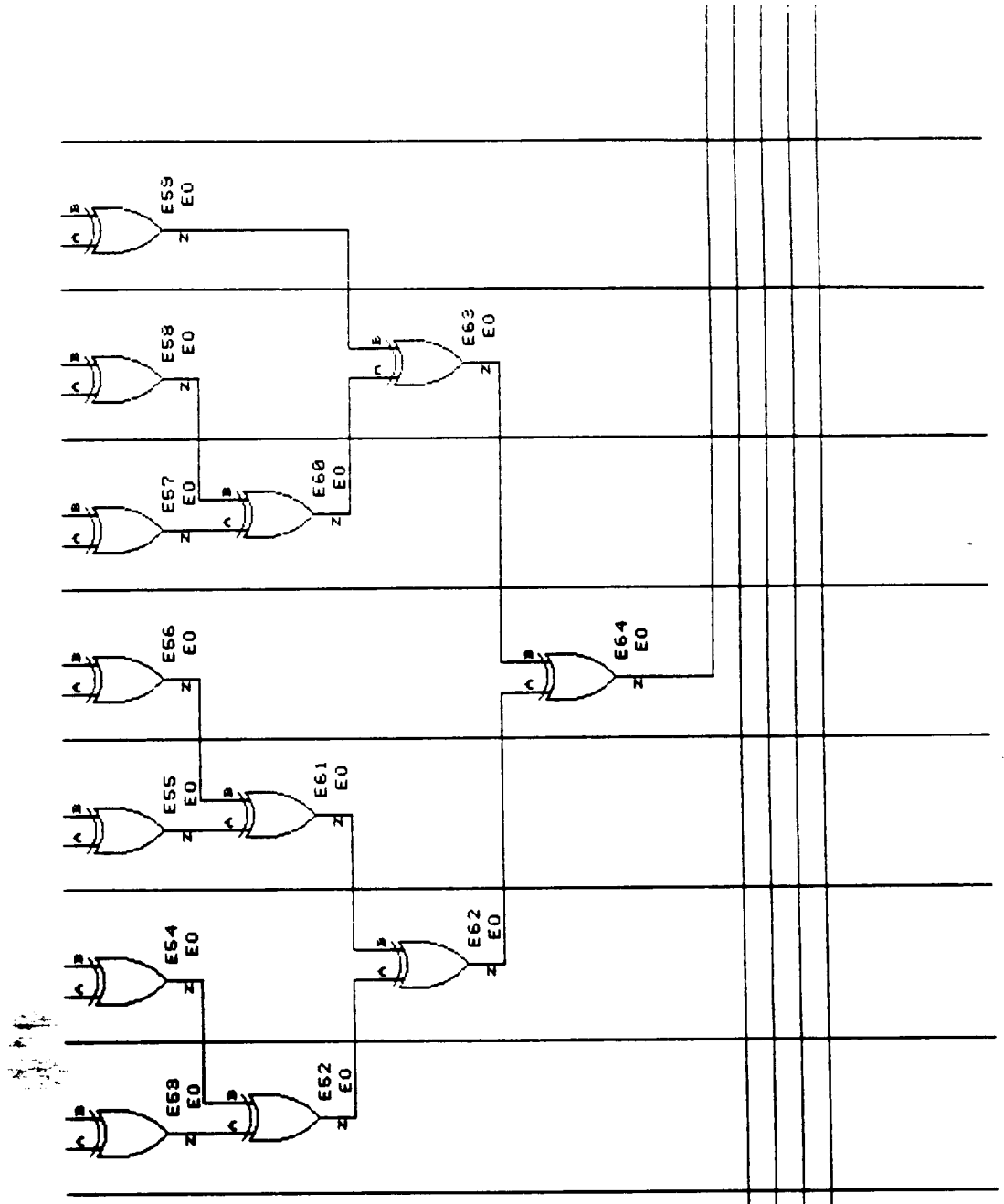


Figure 30

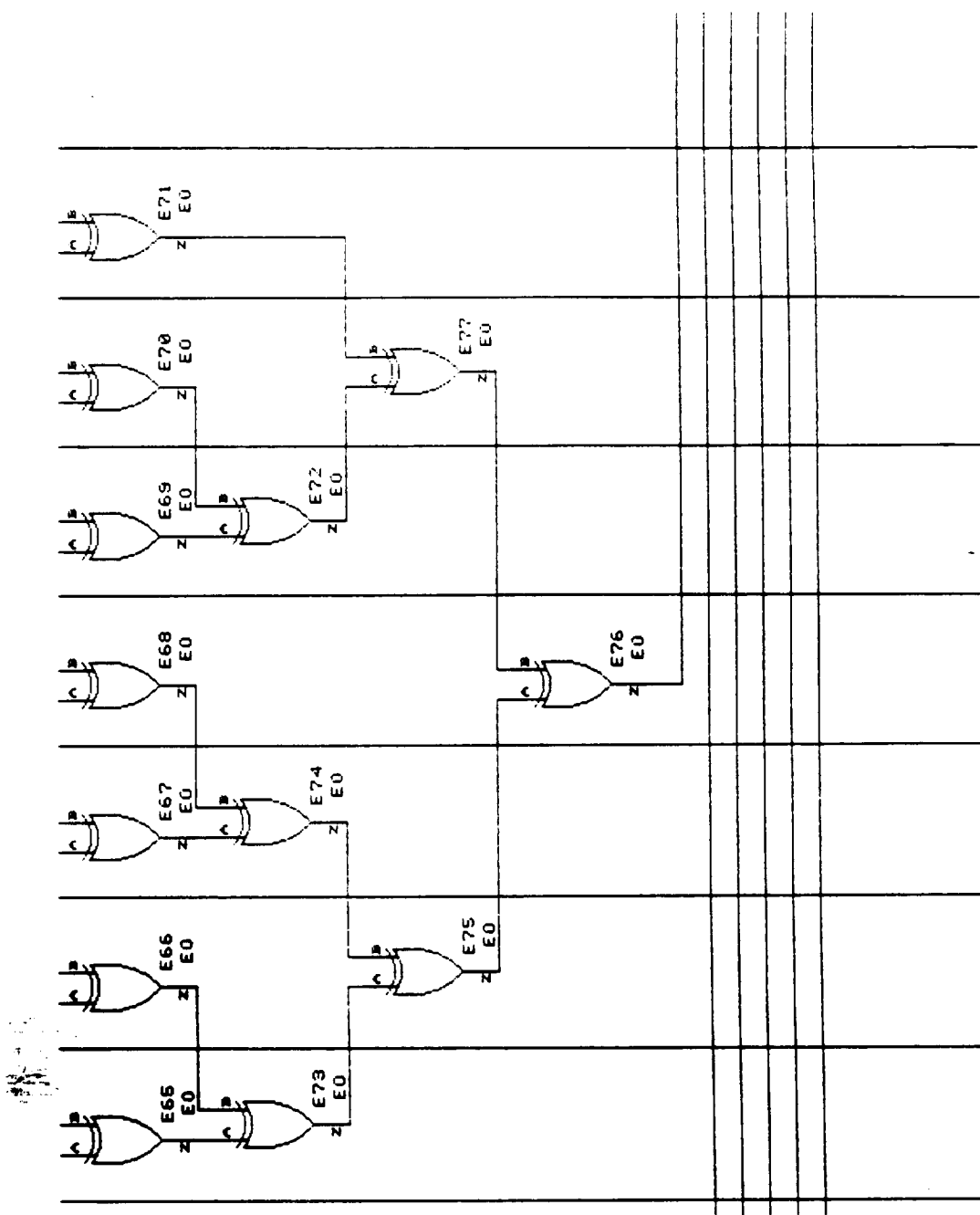


Figure 31

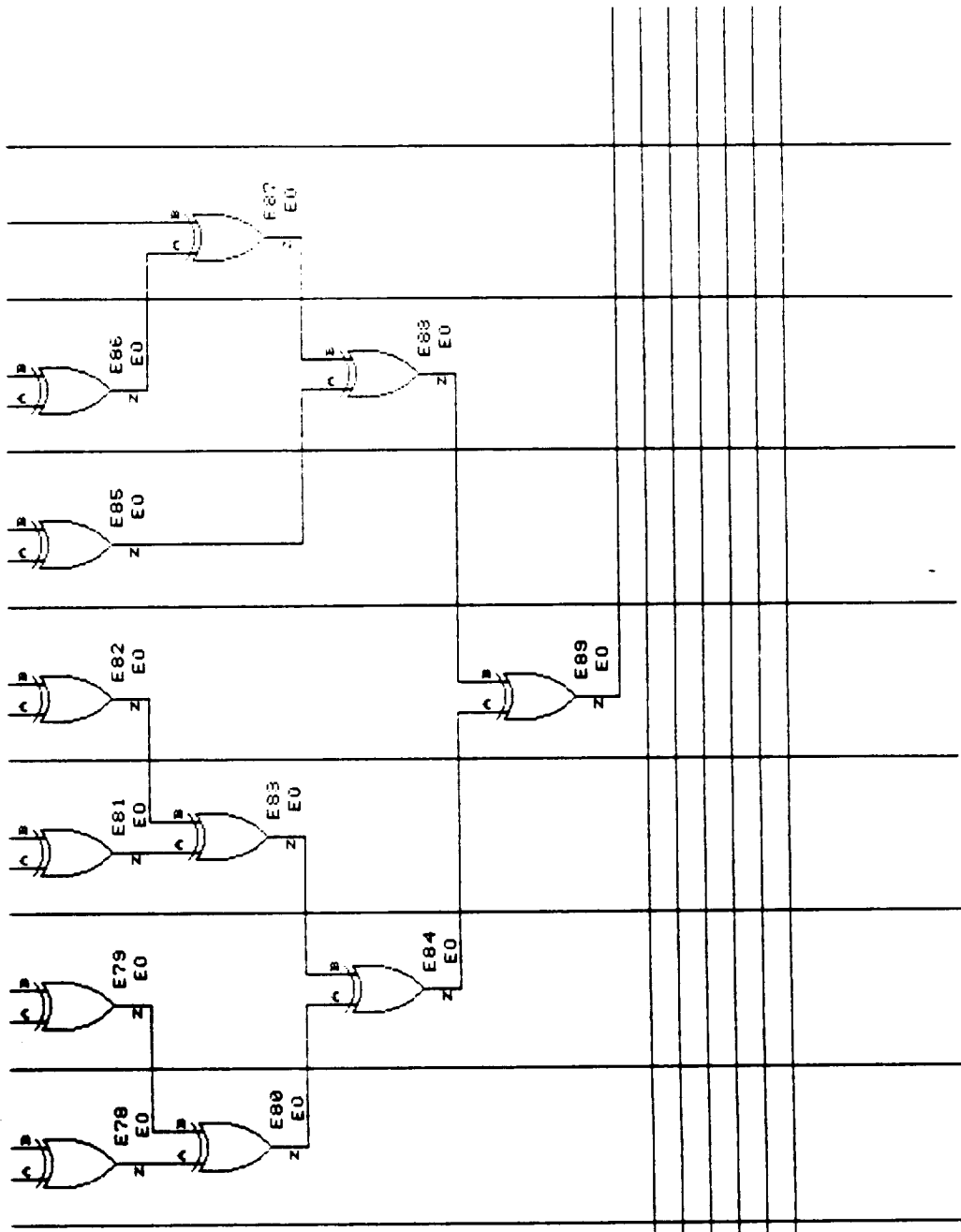


Figure 32

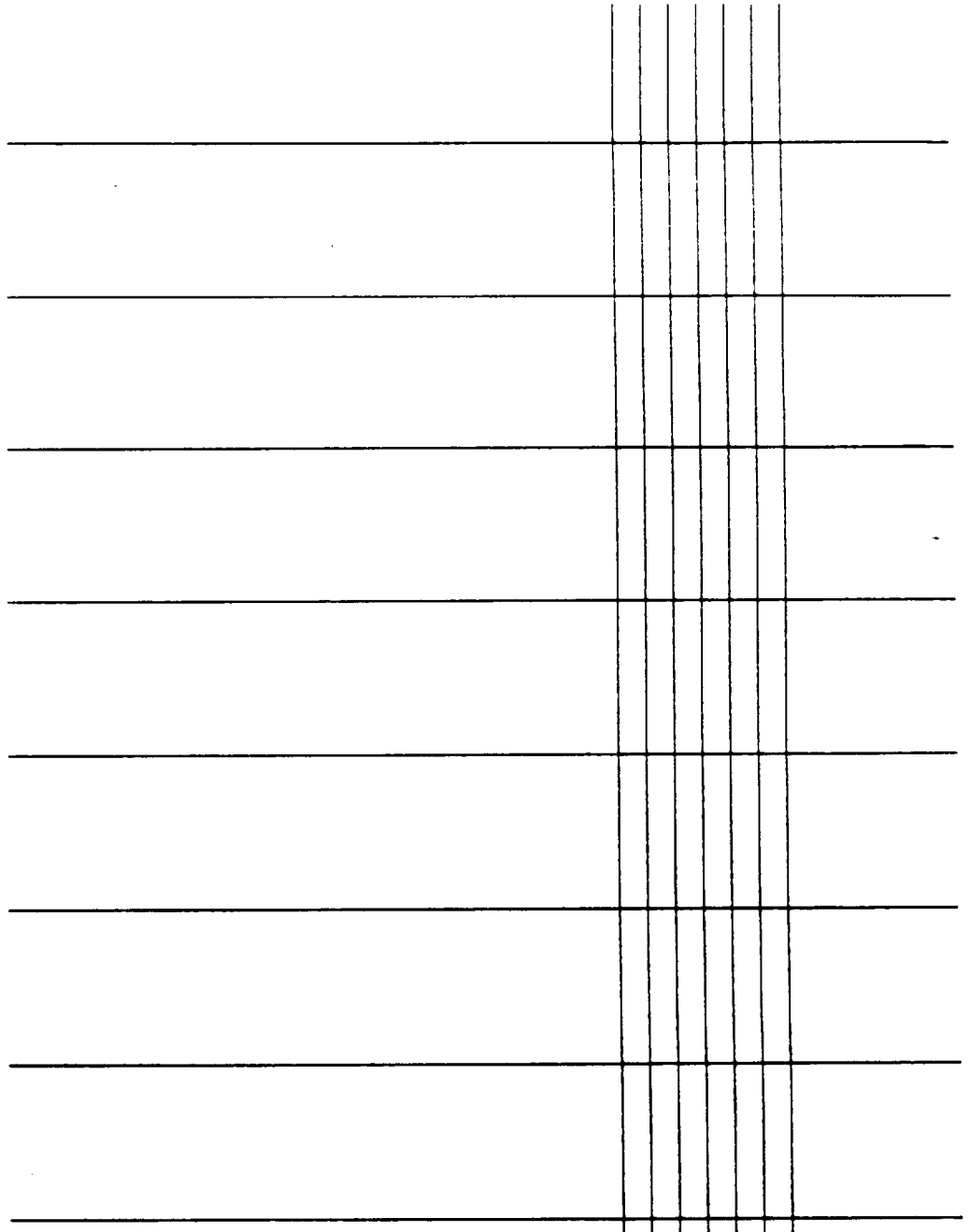


Figure 33

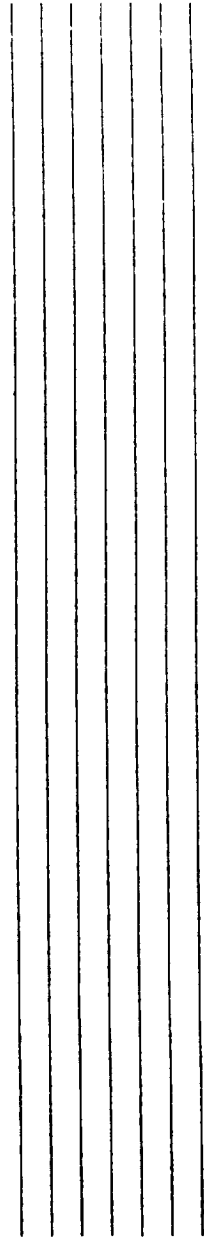


Figure 34

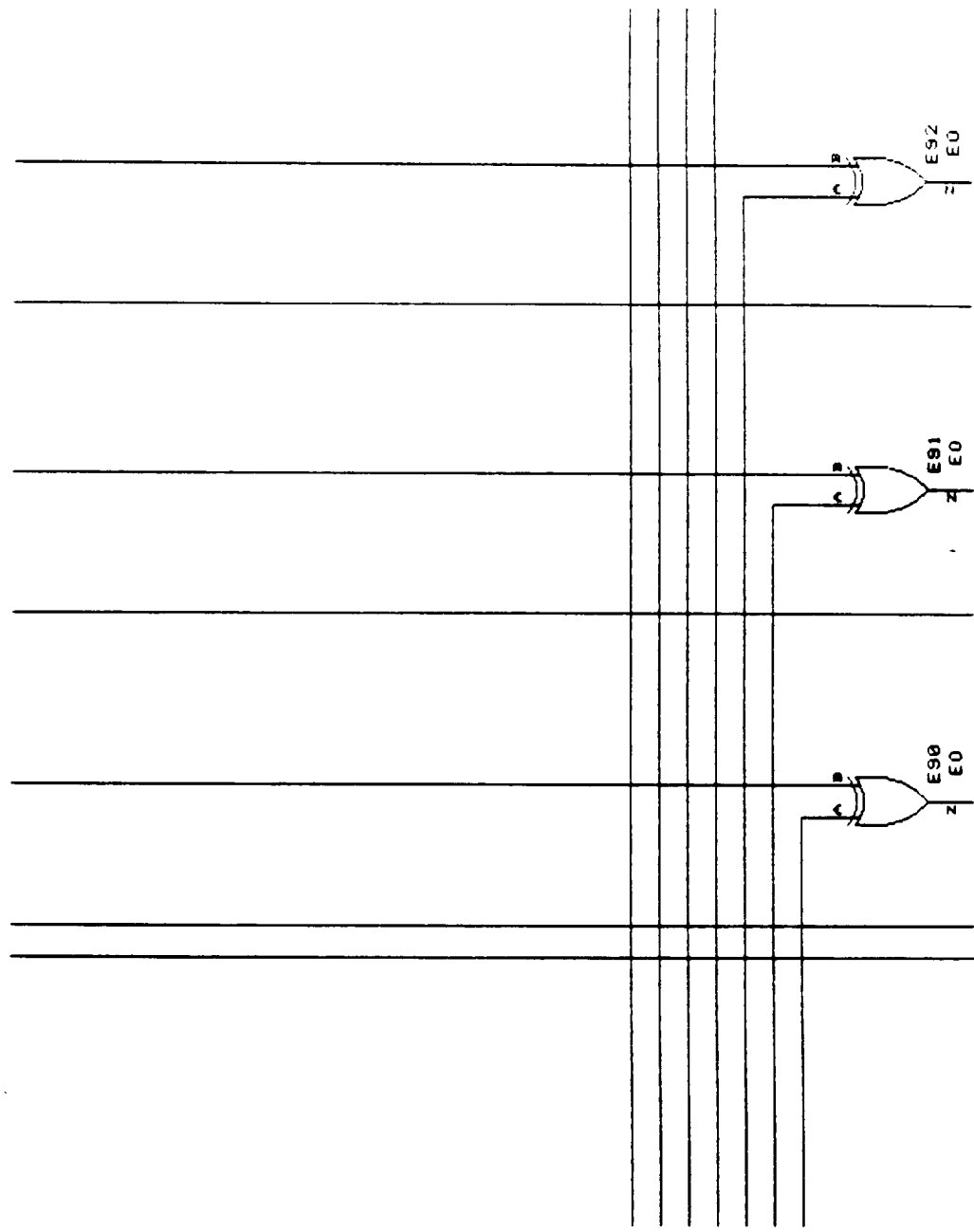


Figure 35

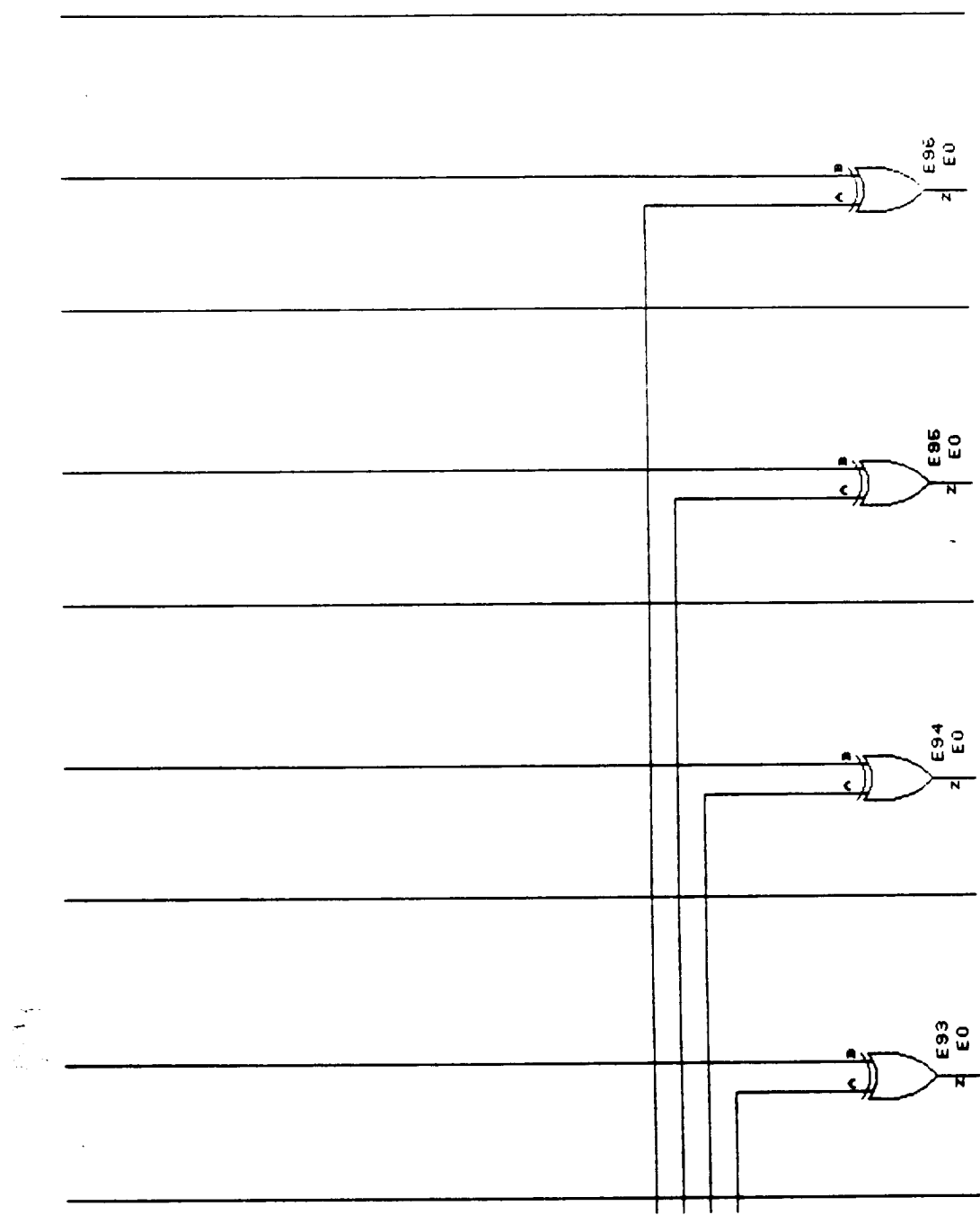


Figure 36

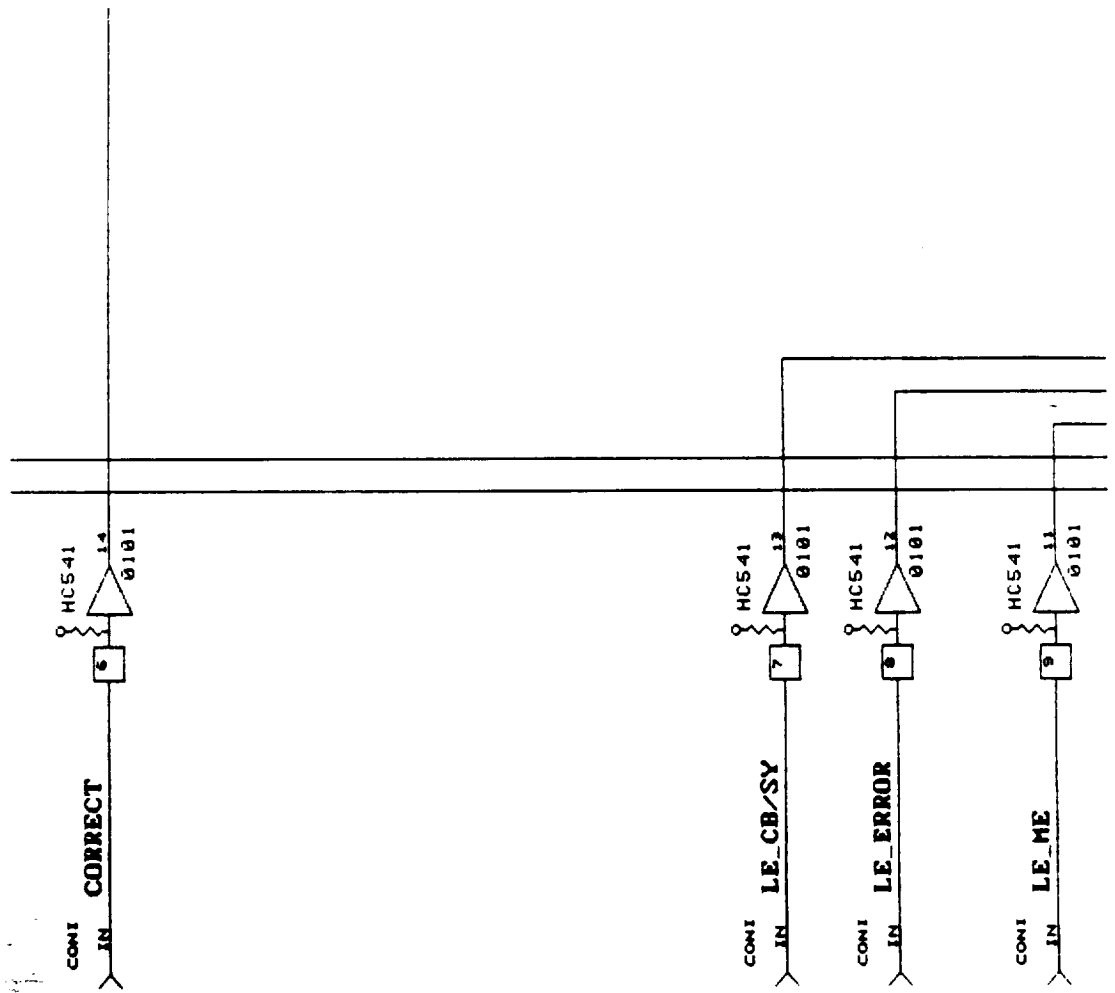


Figure 37

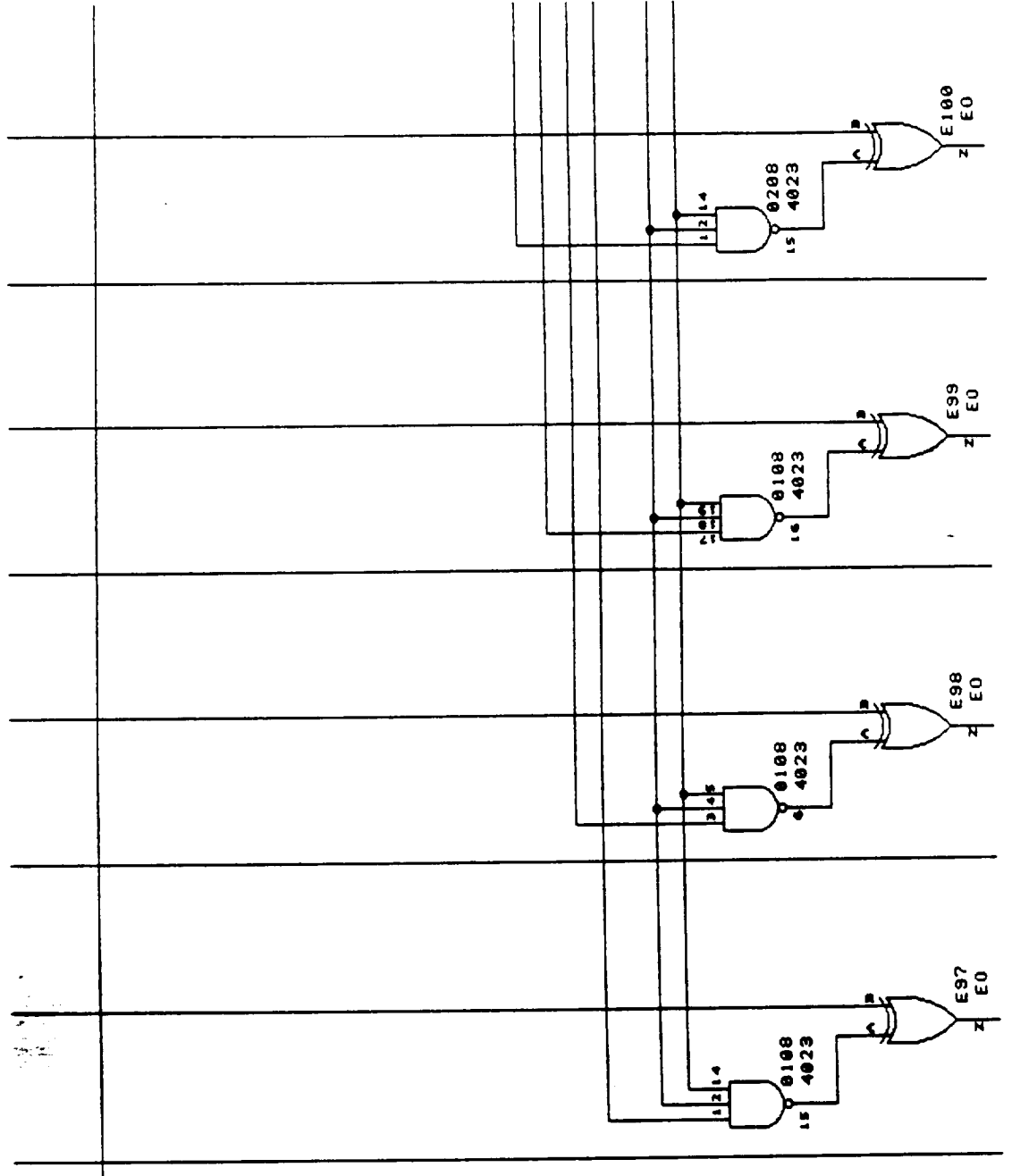


Figure 3B

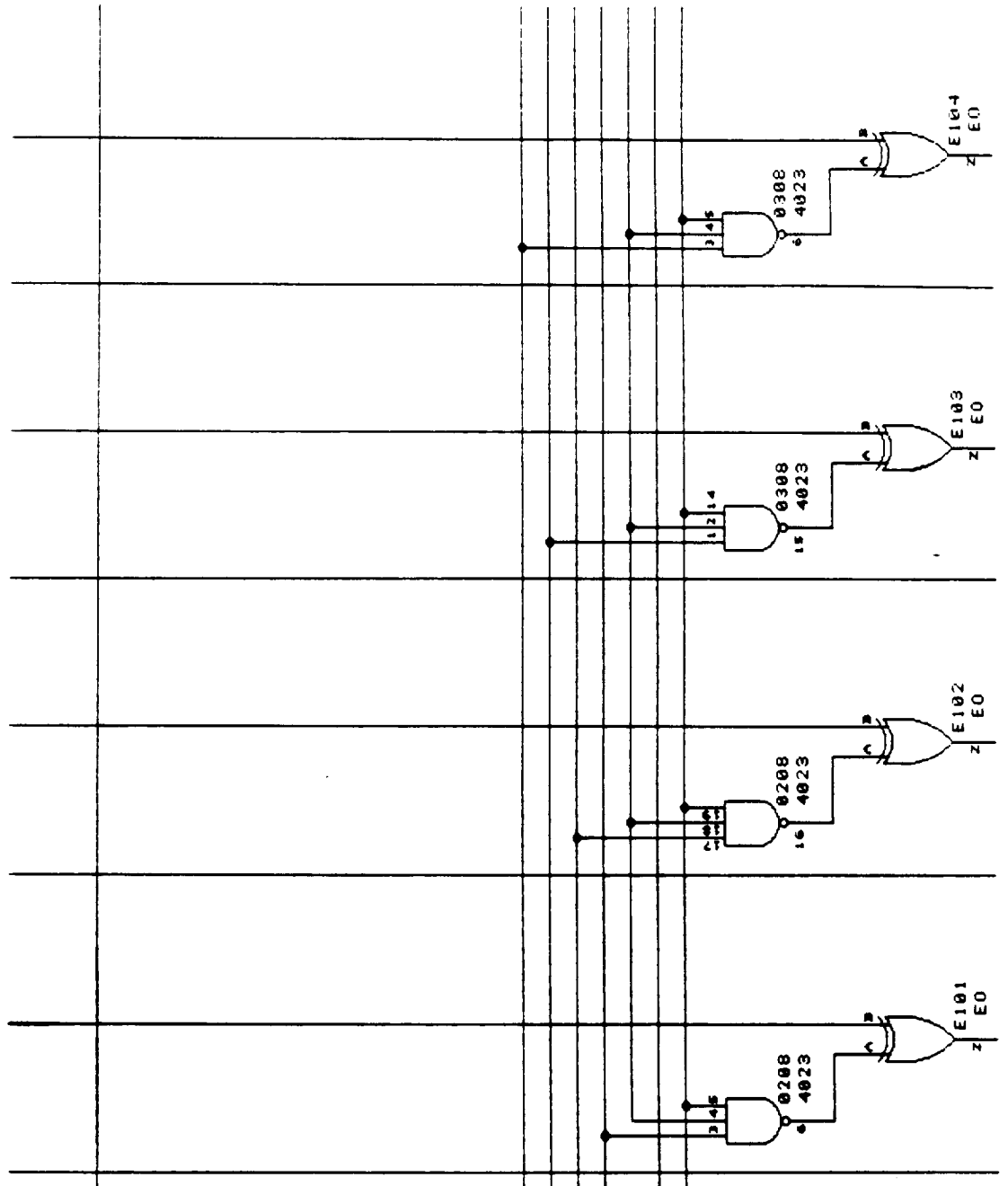


Figure 39

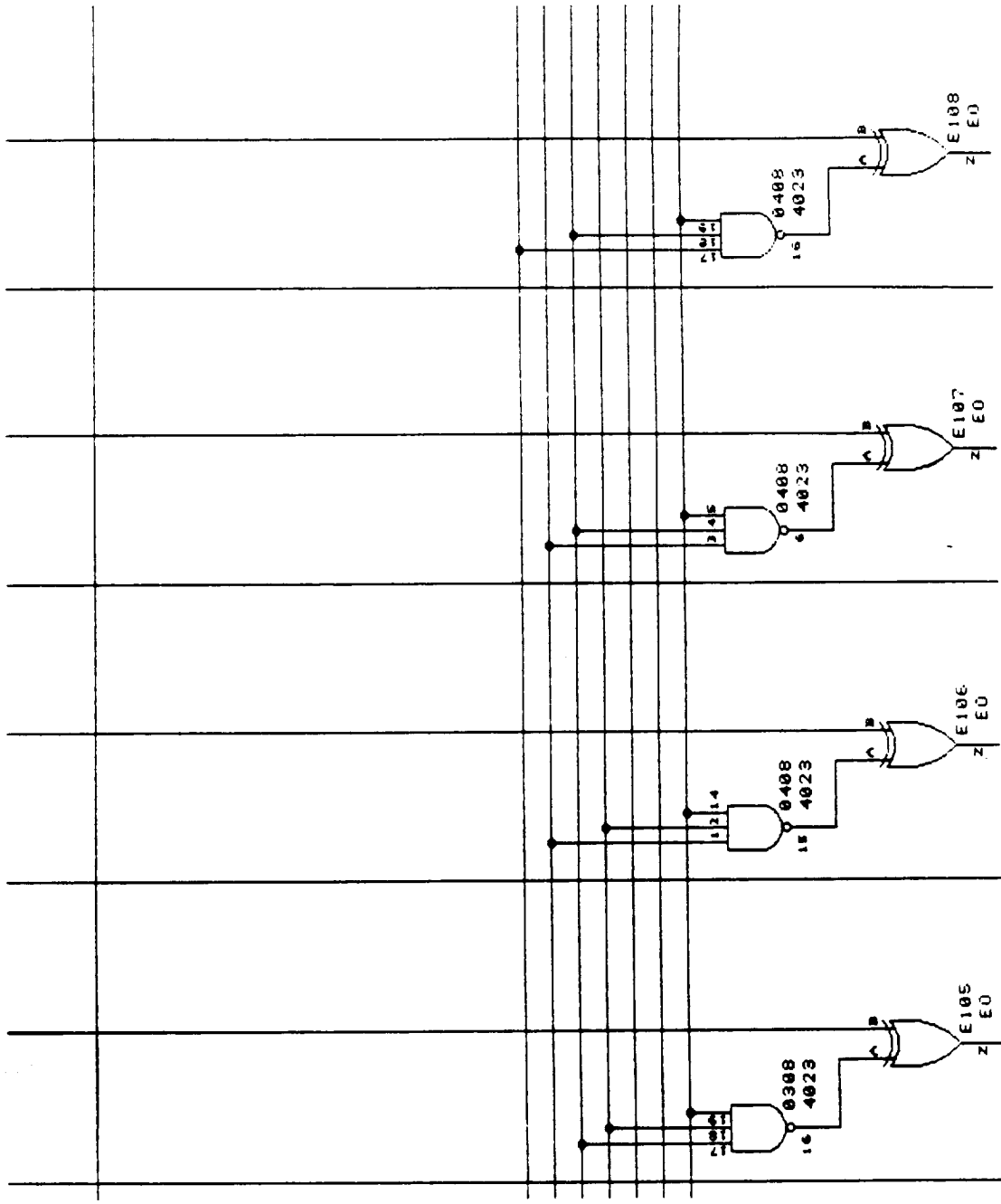


Figure 40

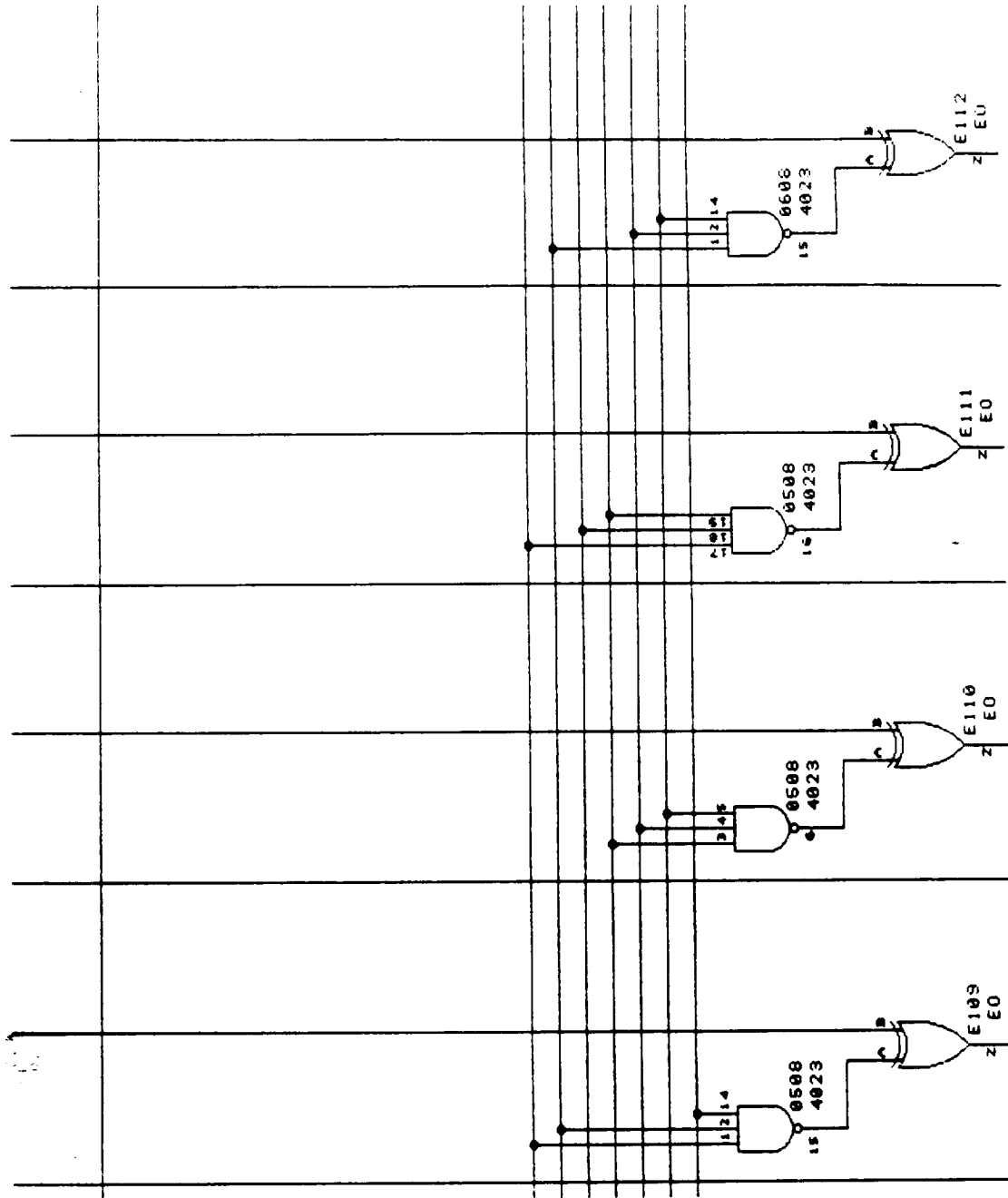


Figure 41

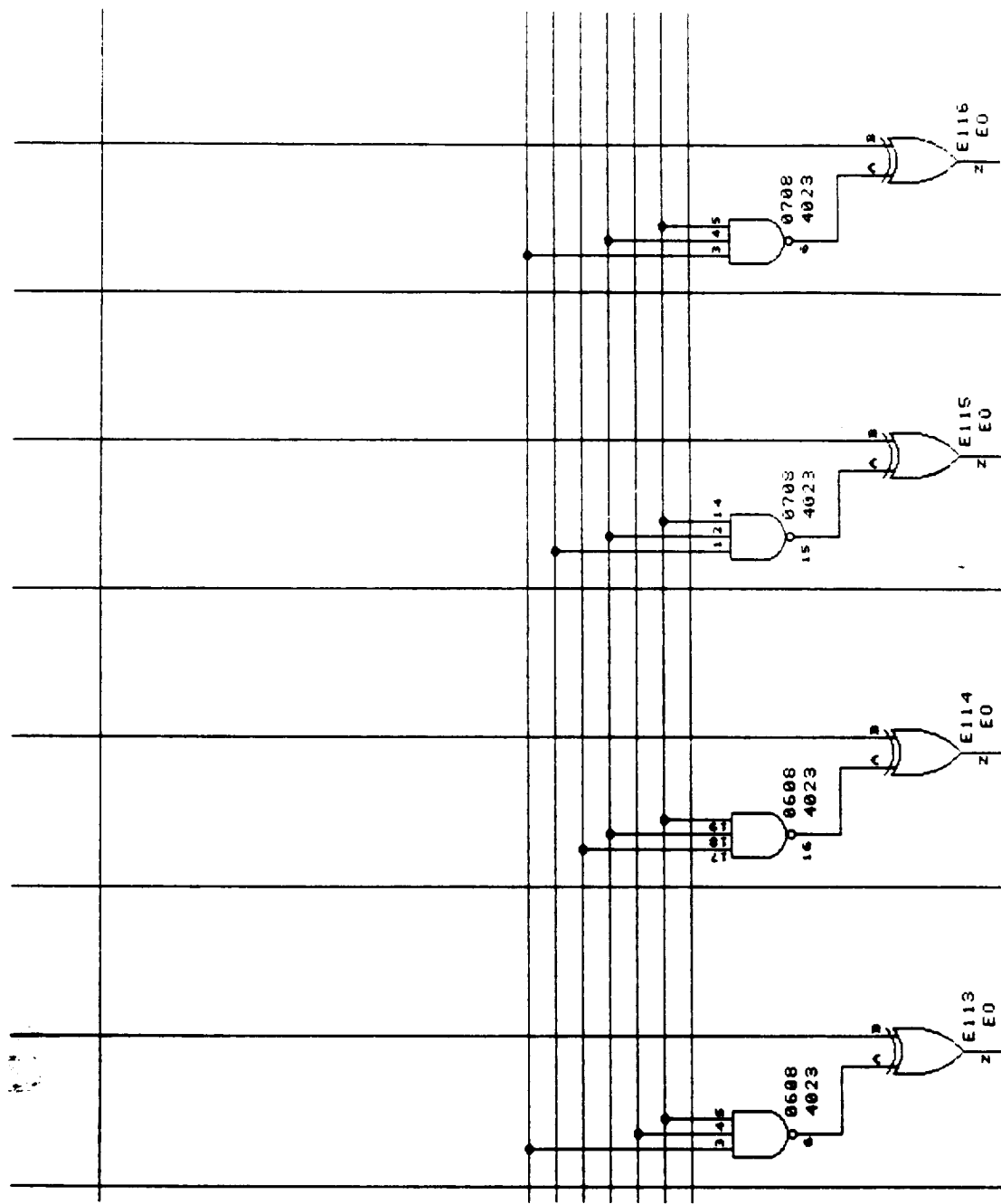


Figure 47

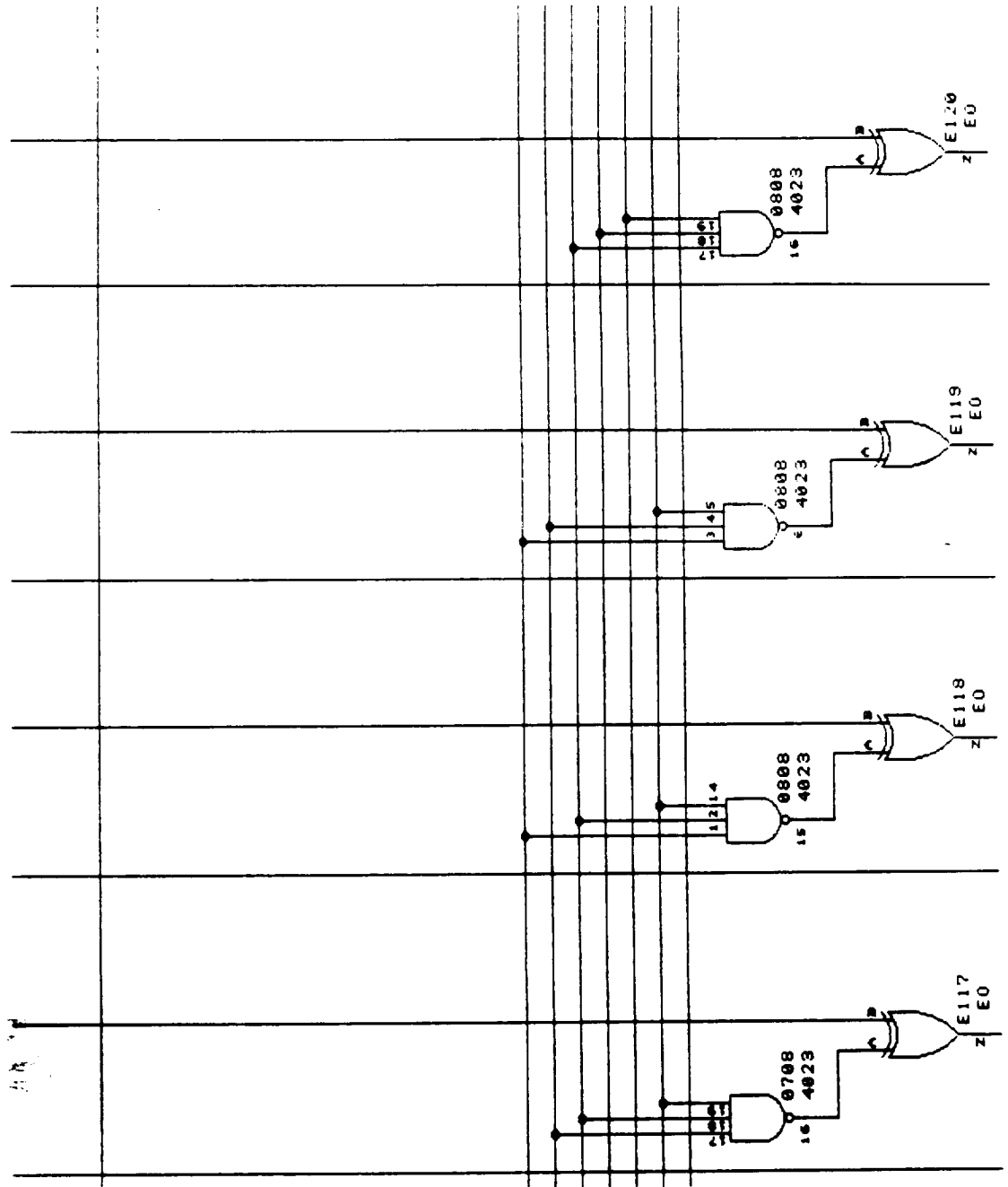


Figure 43

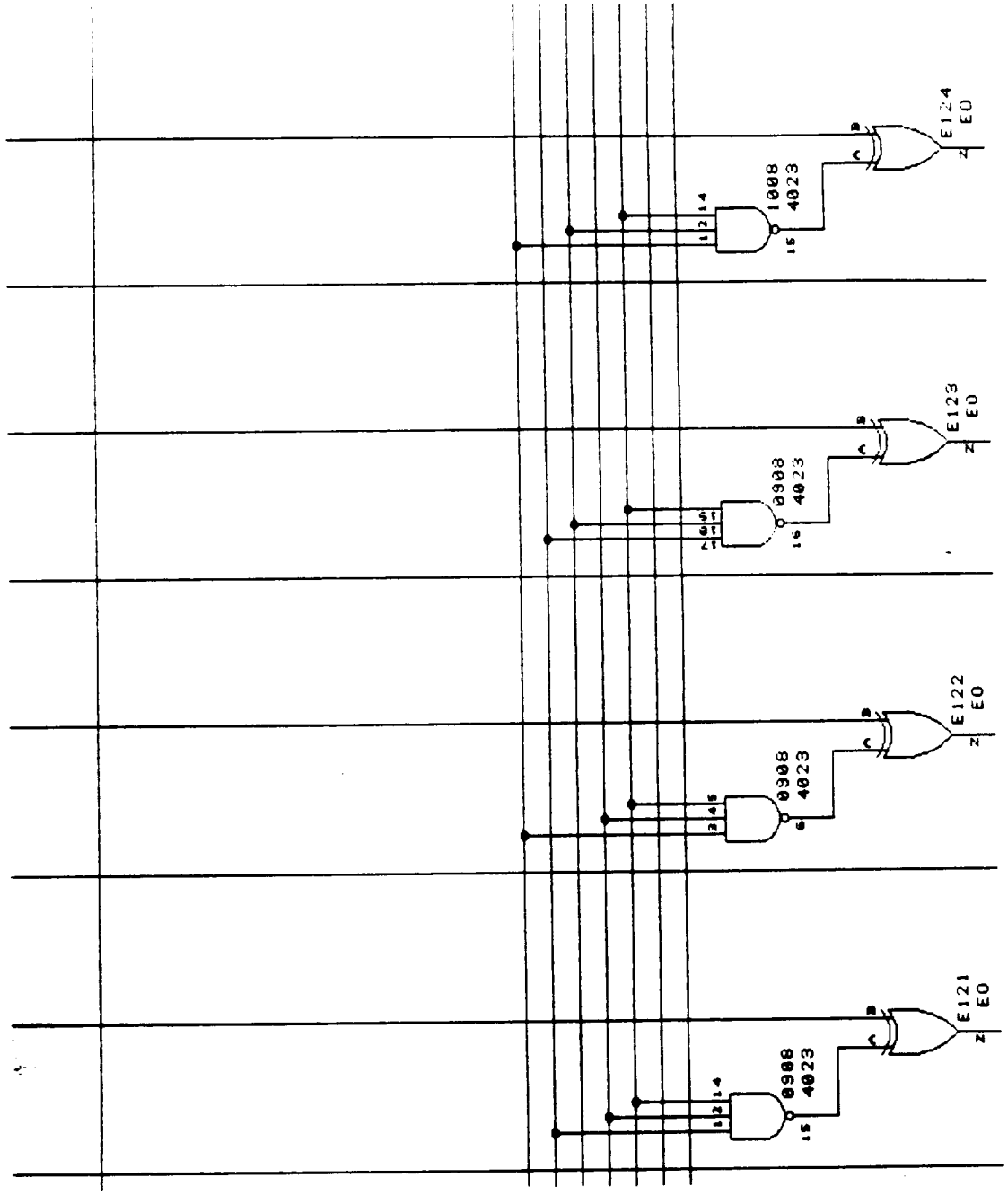


Figure 44

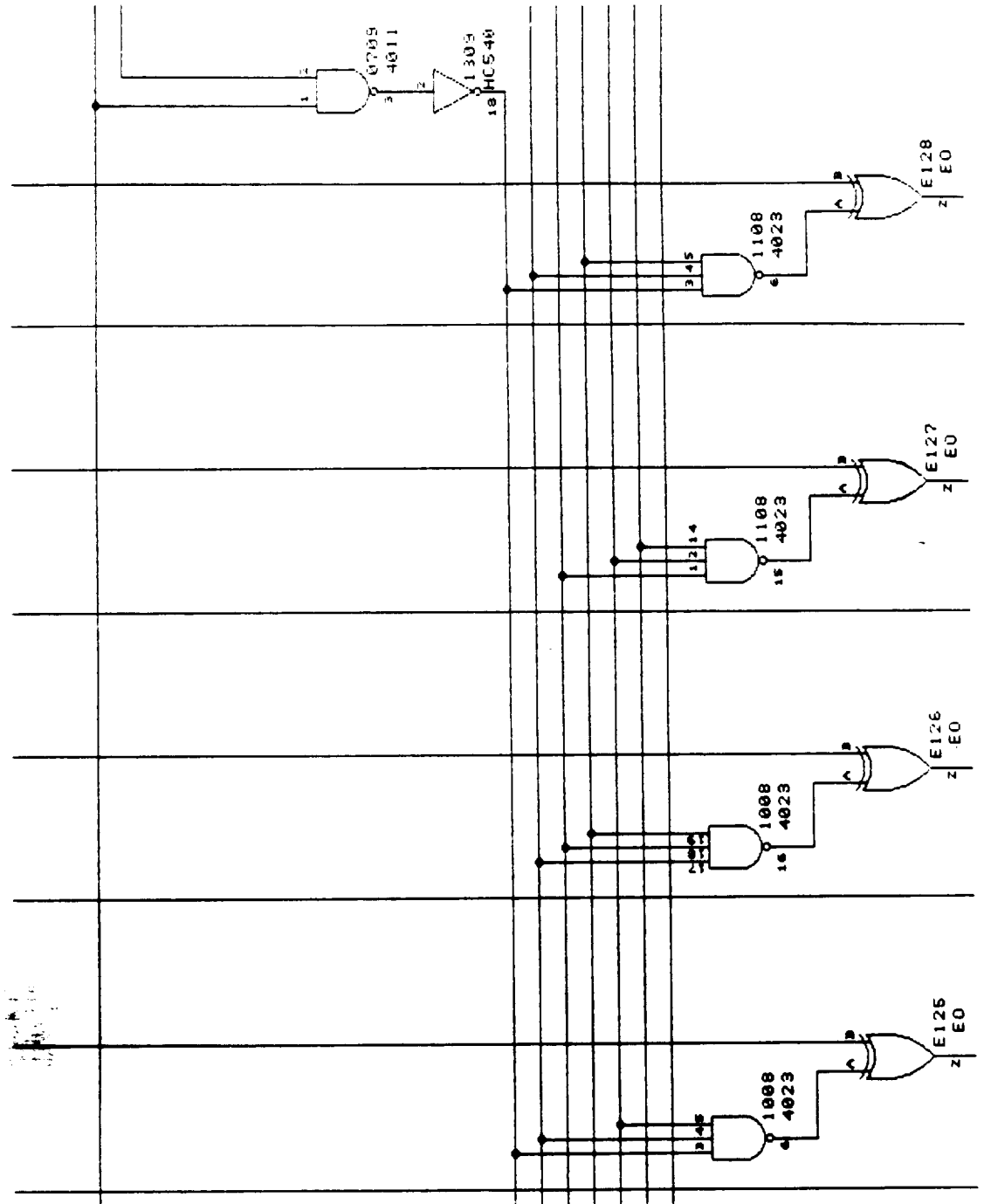


Figure 45

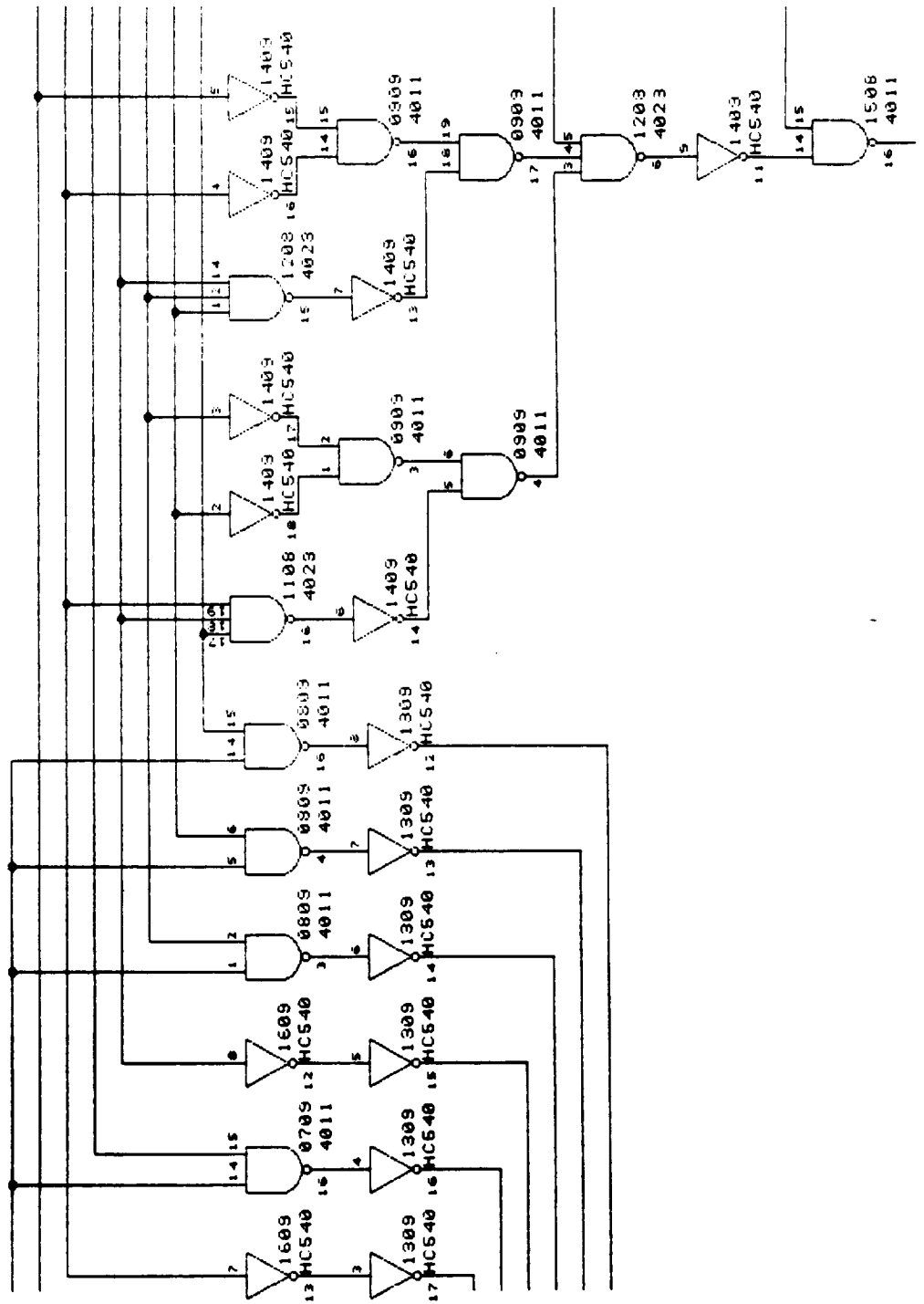


Figure 46

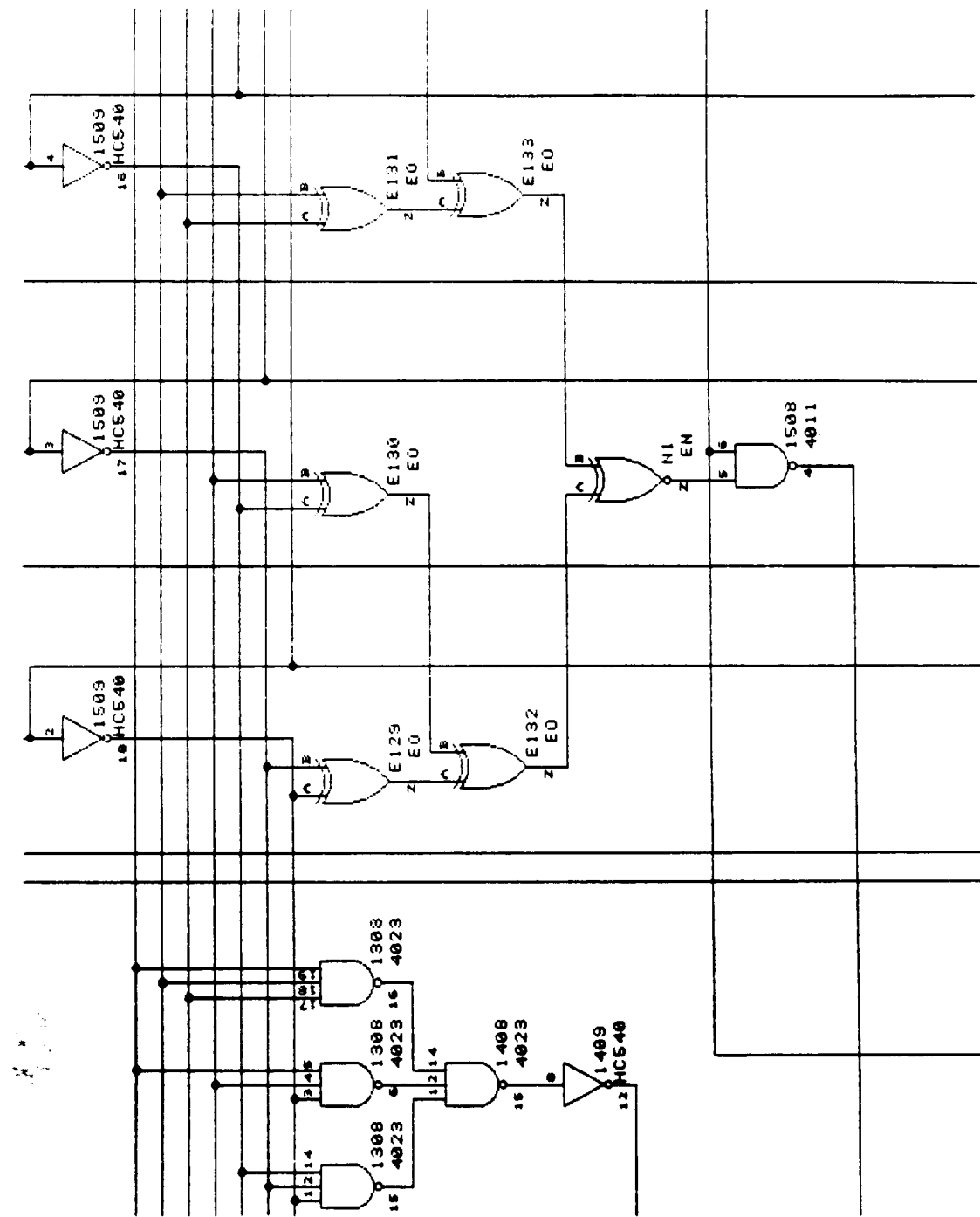


Figure 47

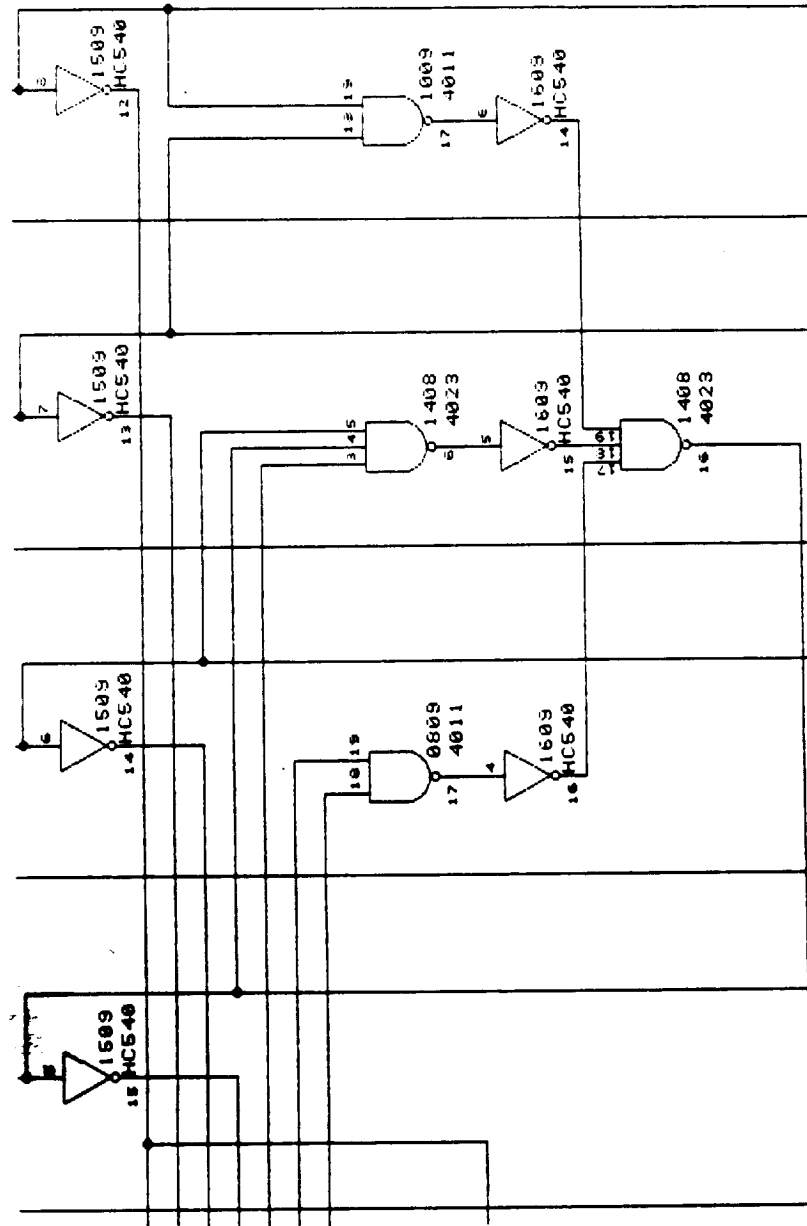


Figure 48

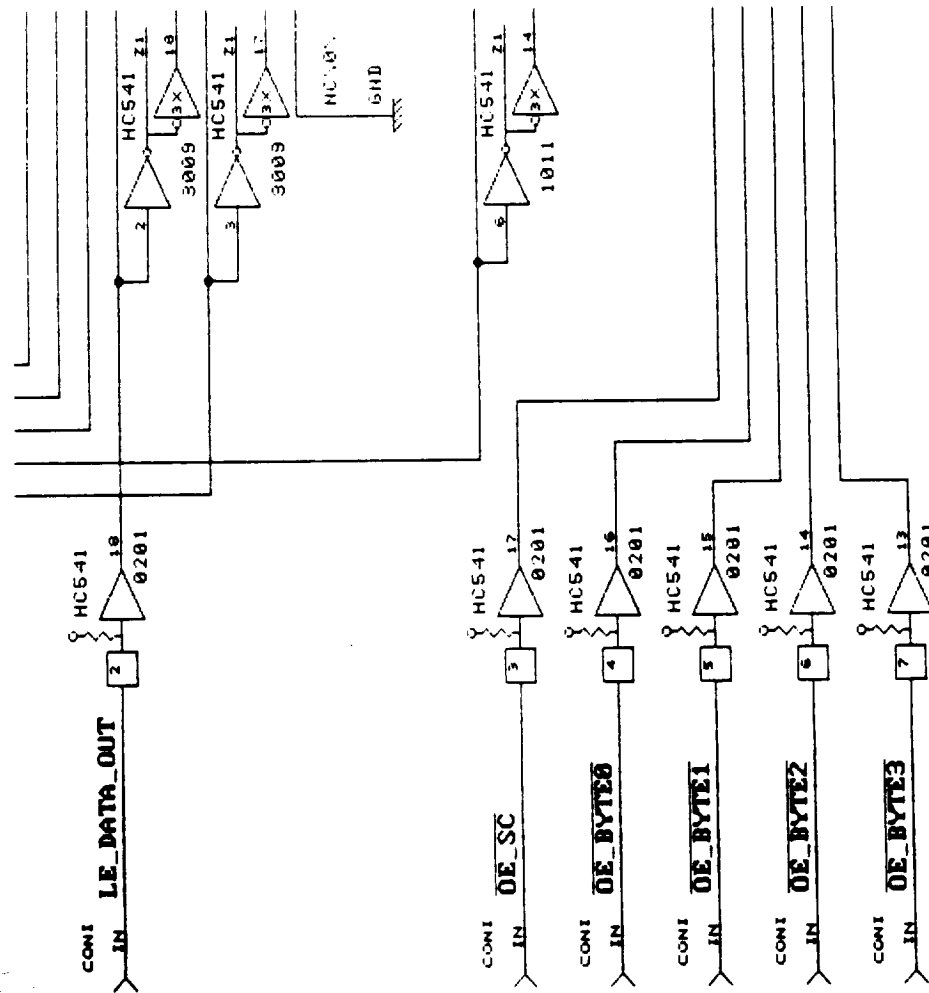


Figure 49

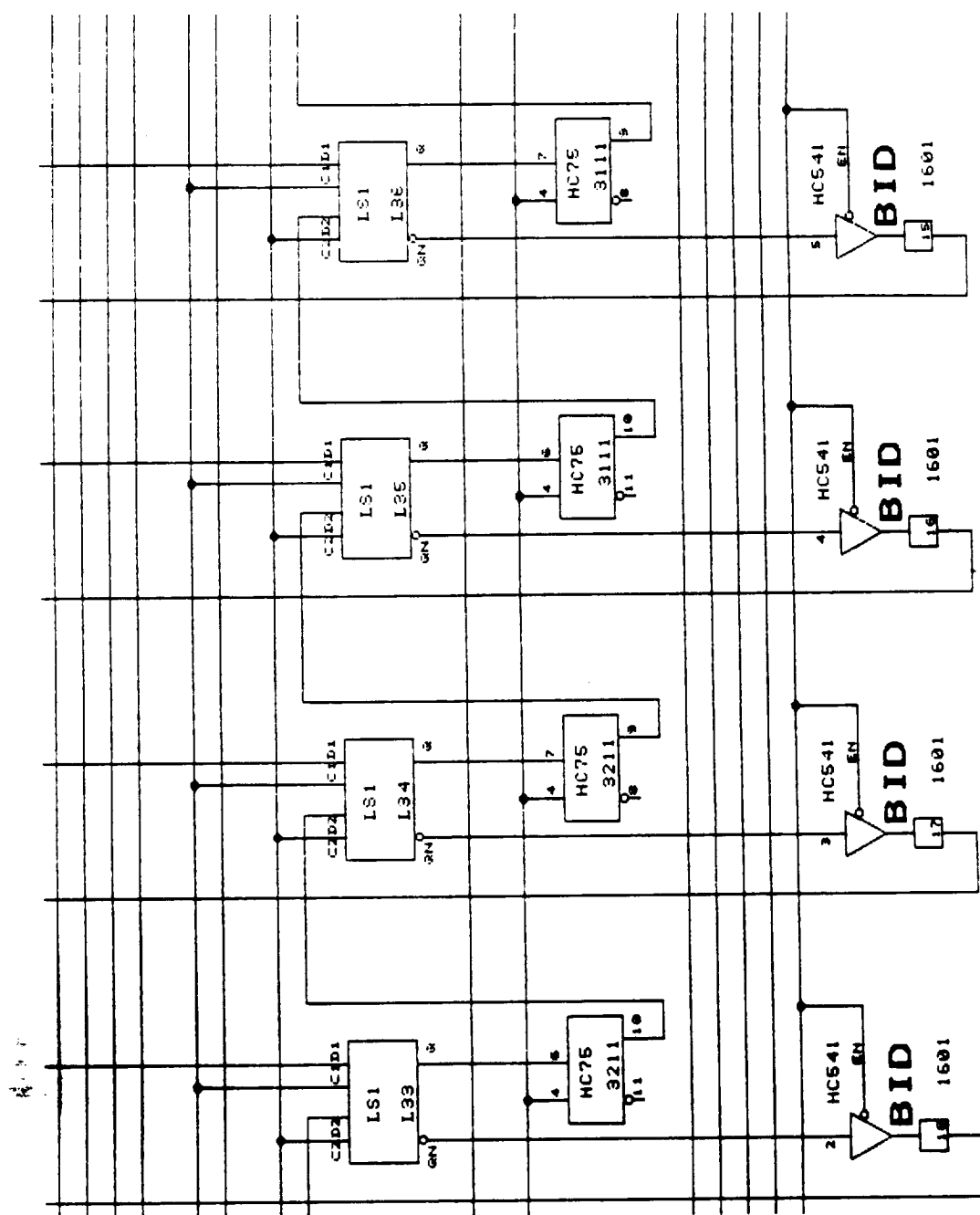


Figure 50

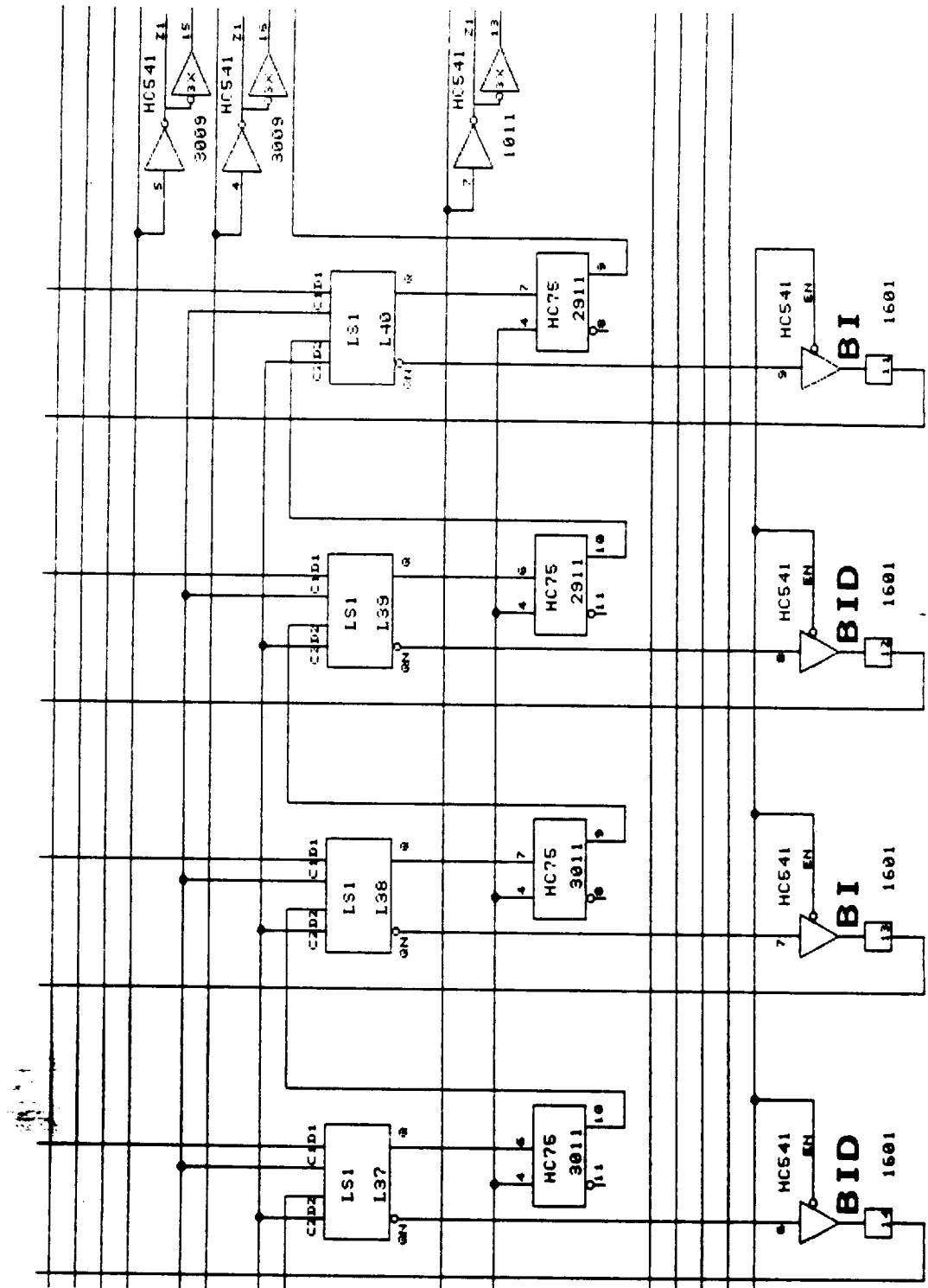


Figure 5.1

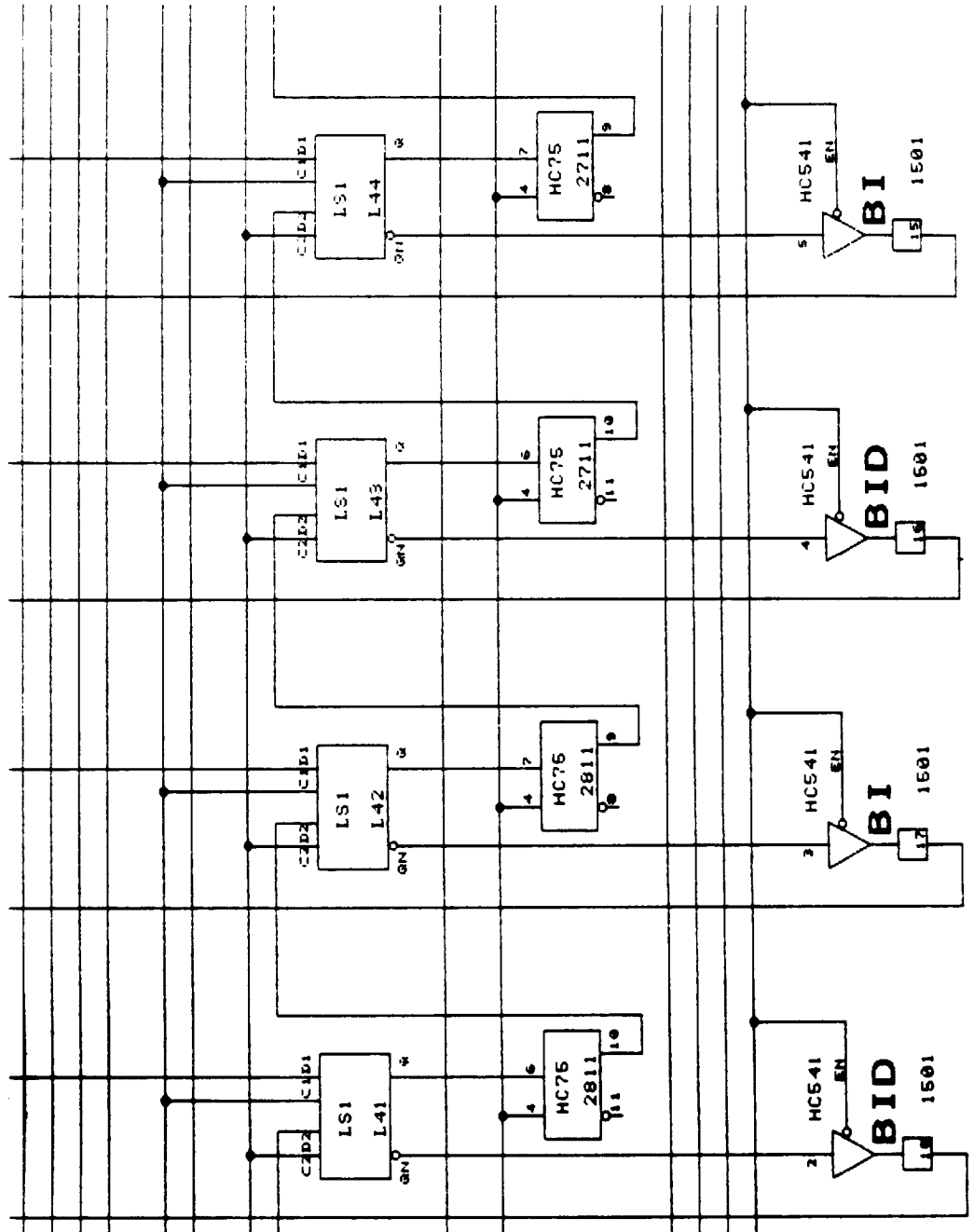


Figure 52

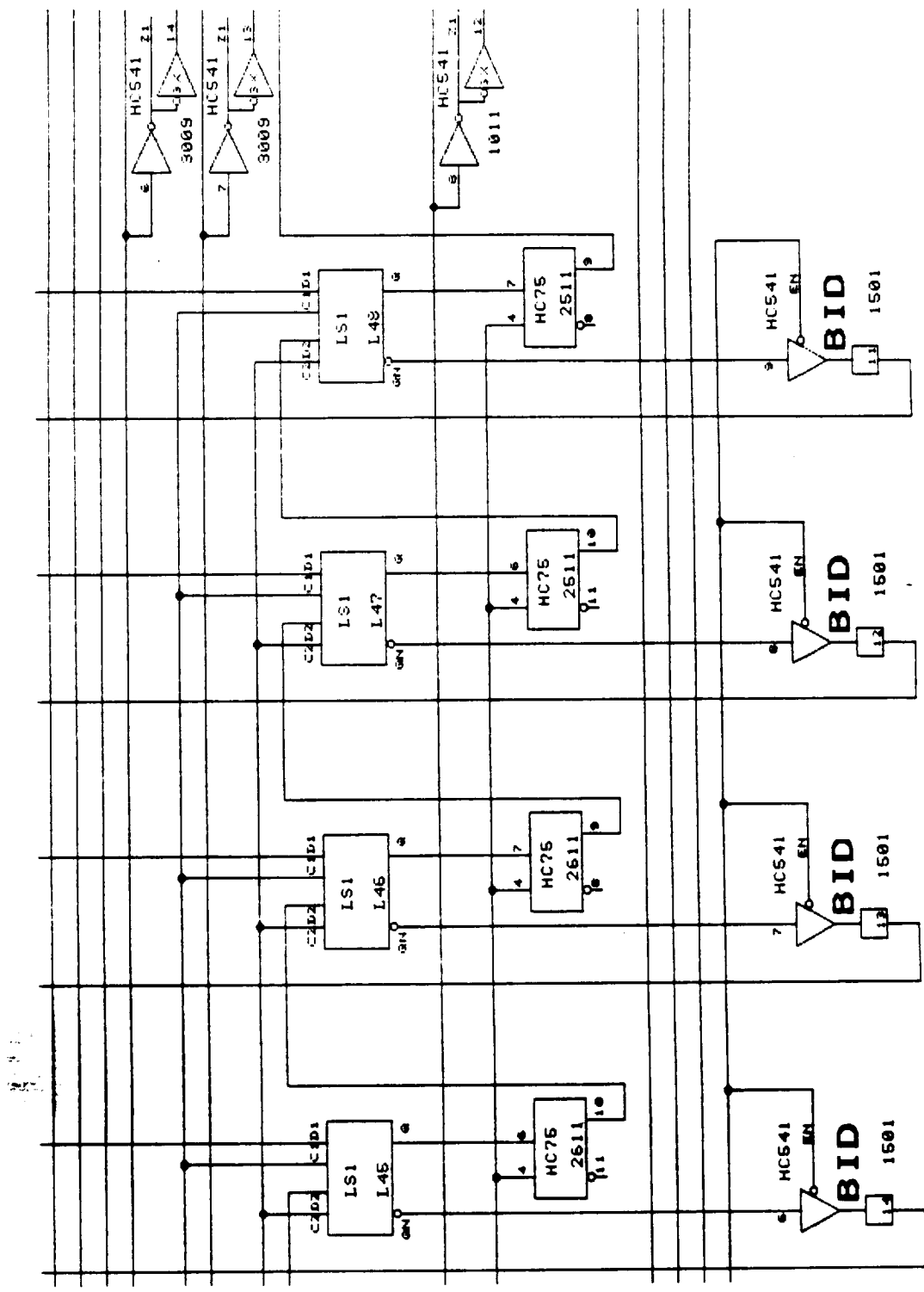


Figure 53

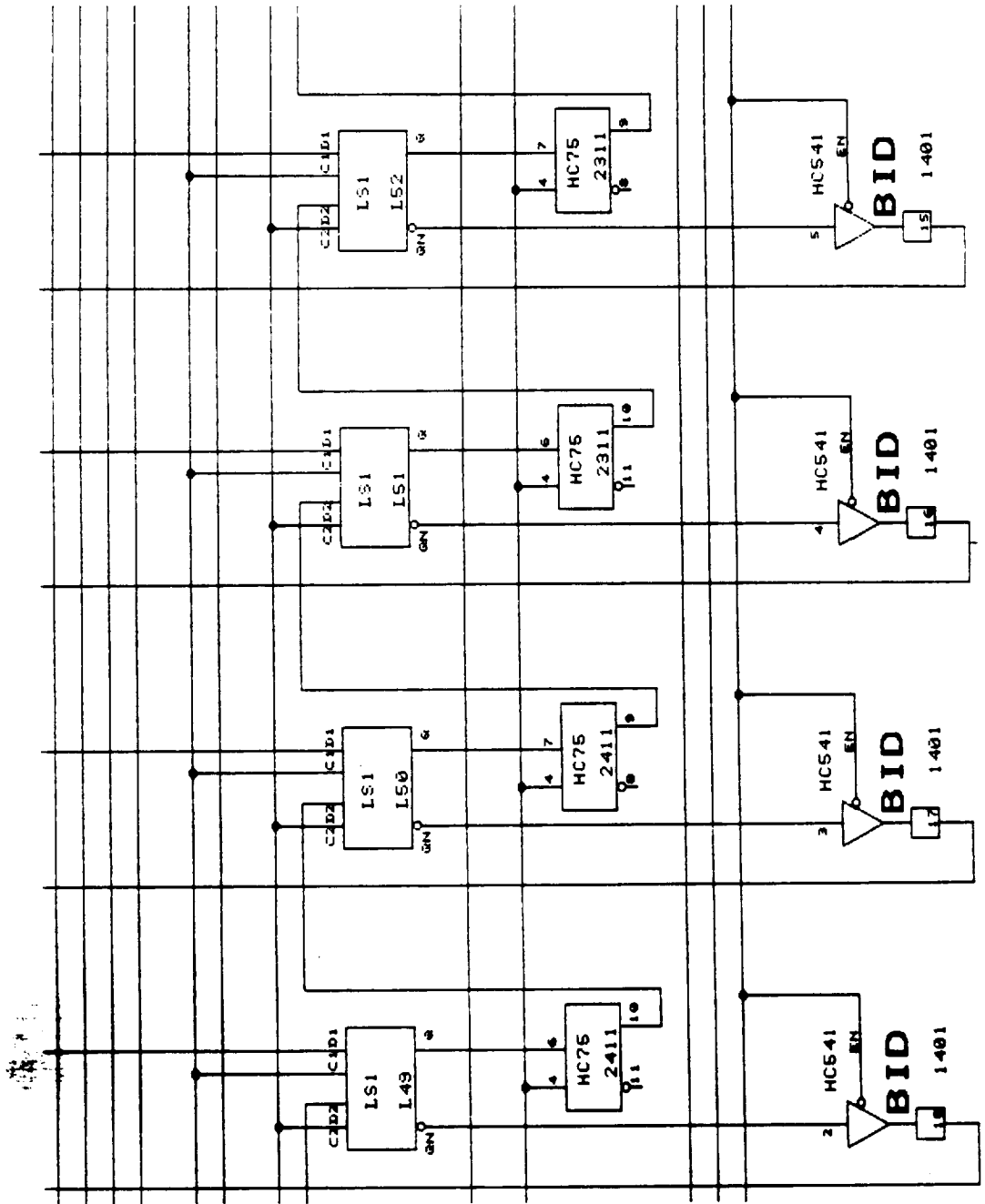


Figure 54

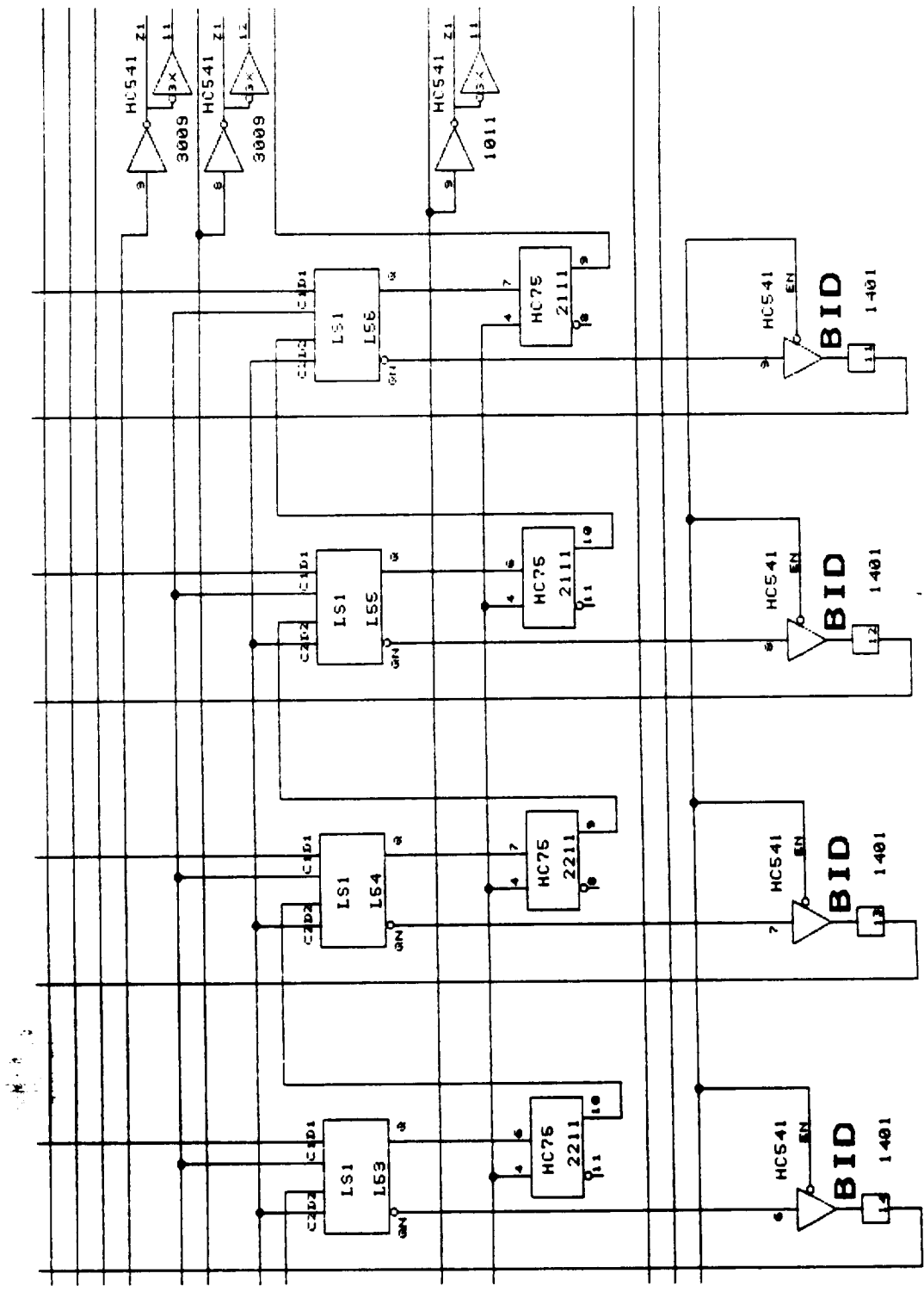


Figure 55

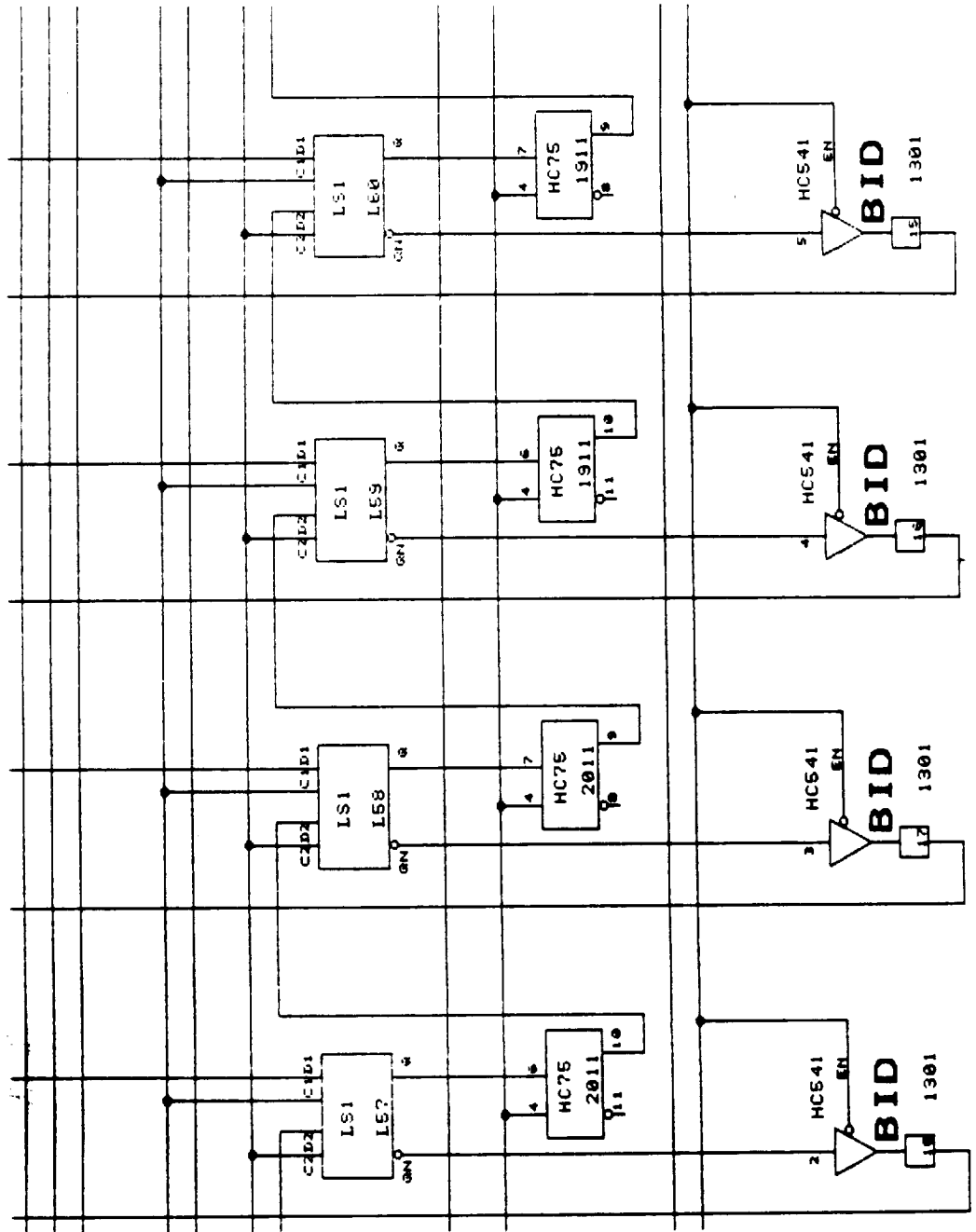


Figure 56

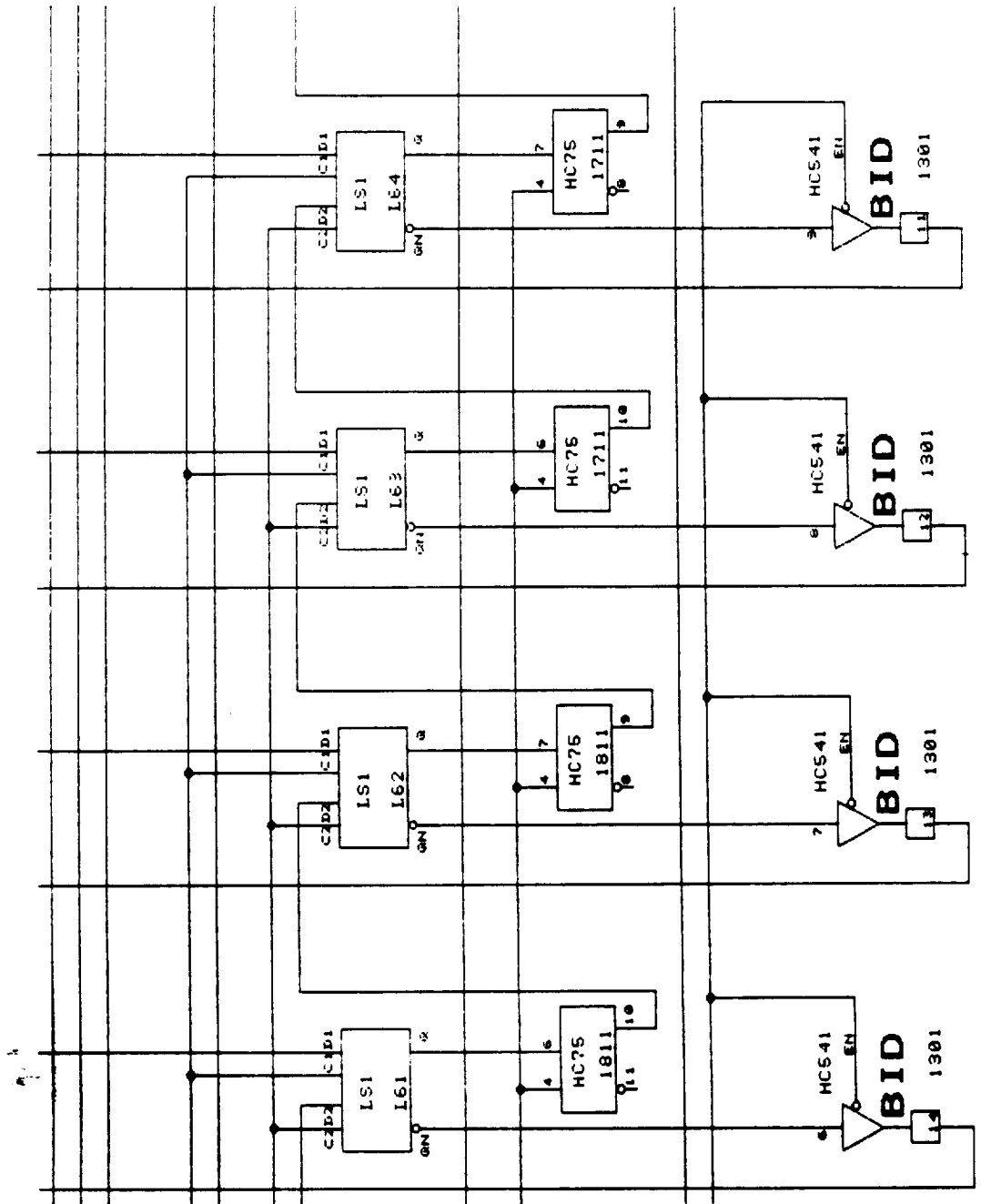


Figure 57

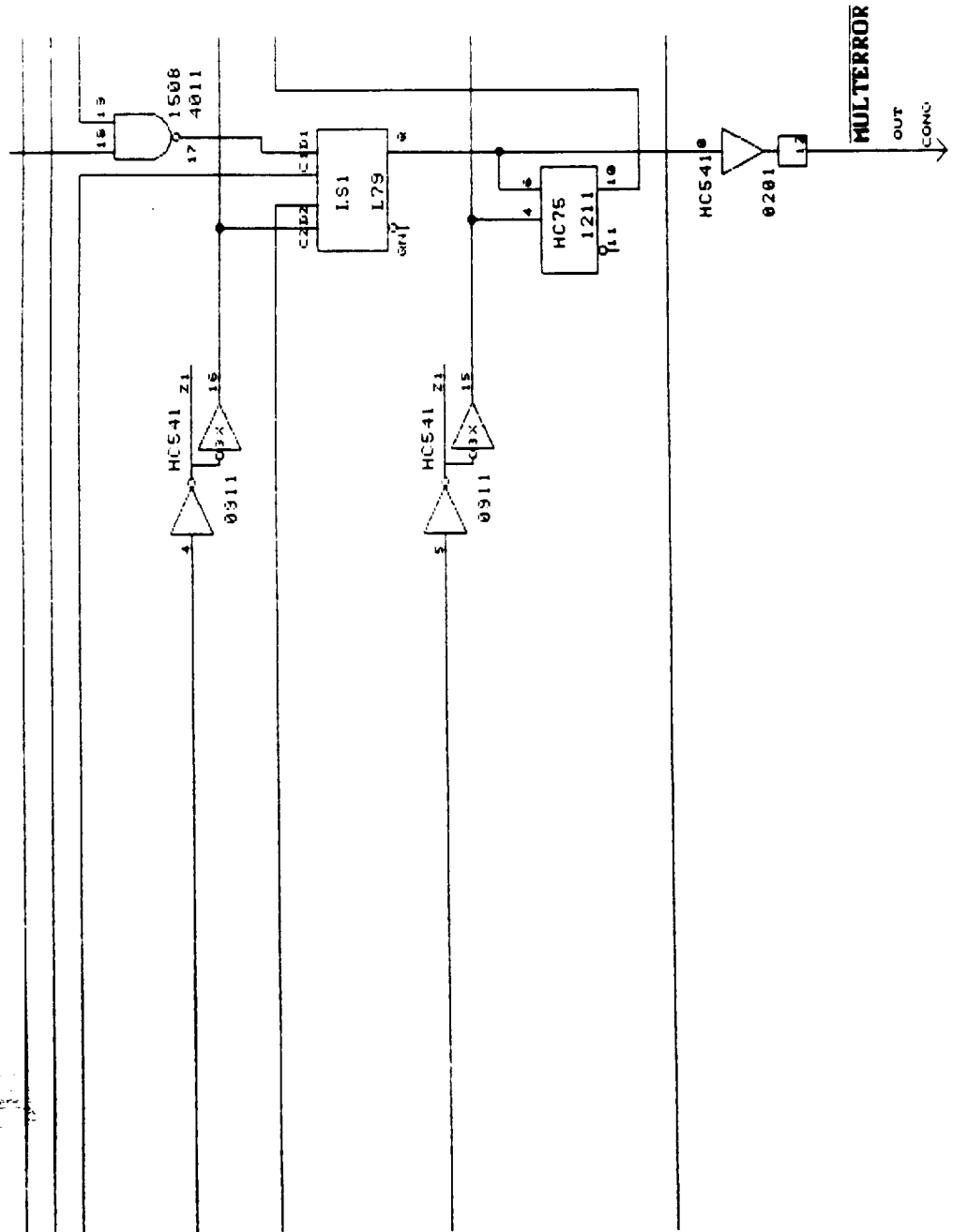


Figure 58

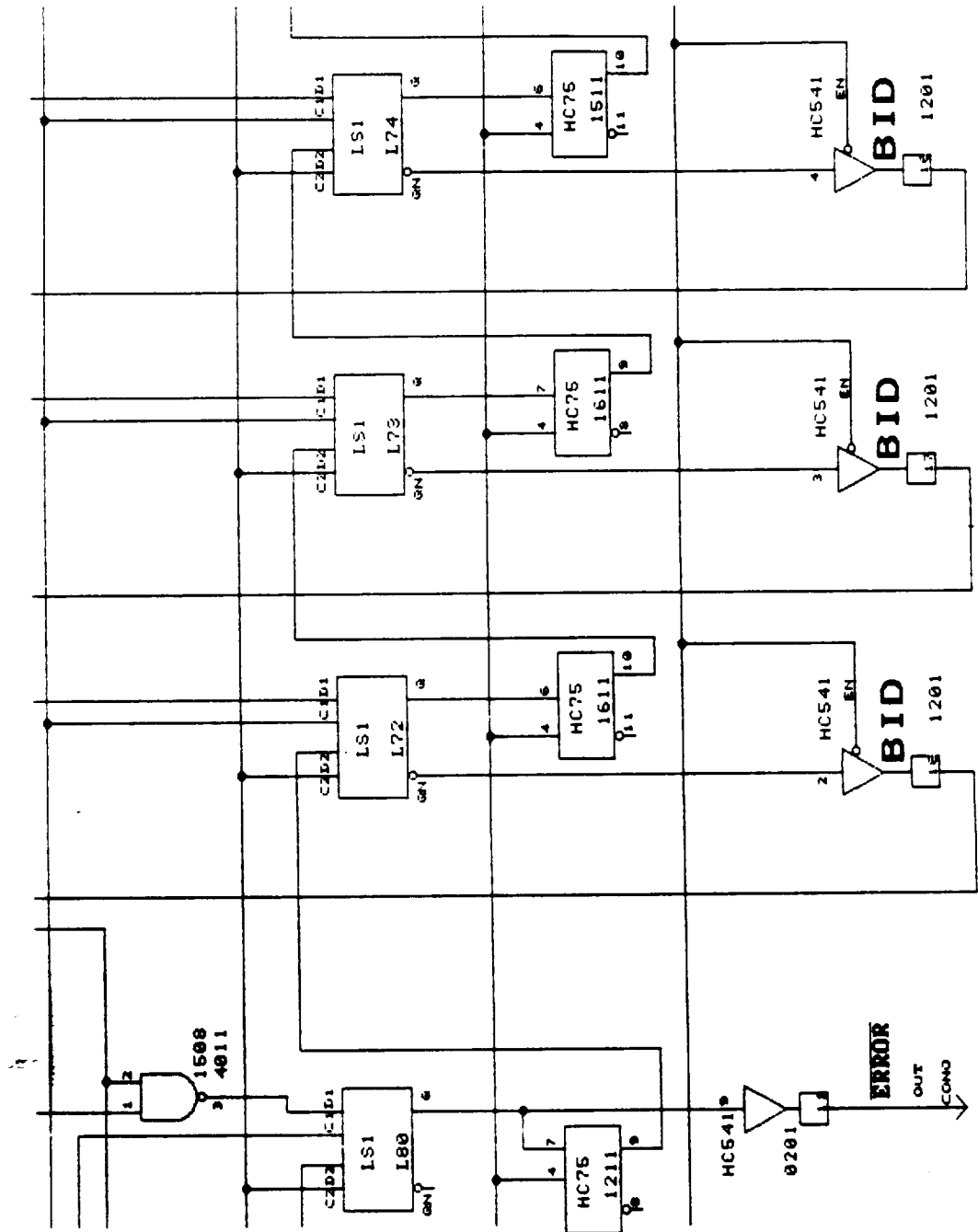
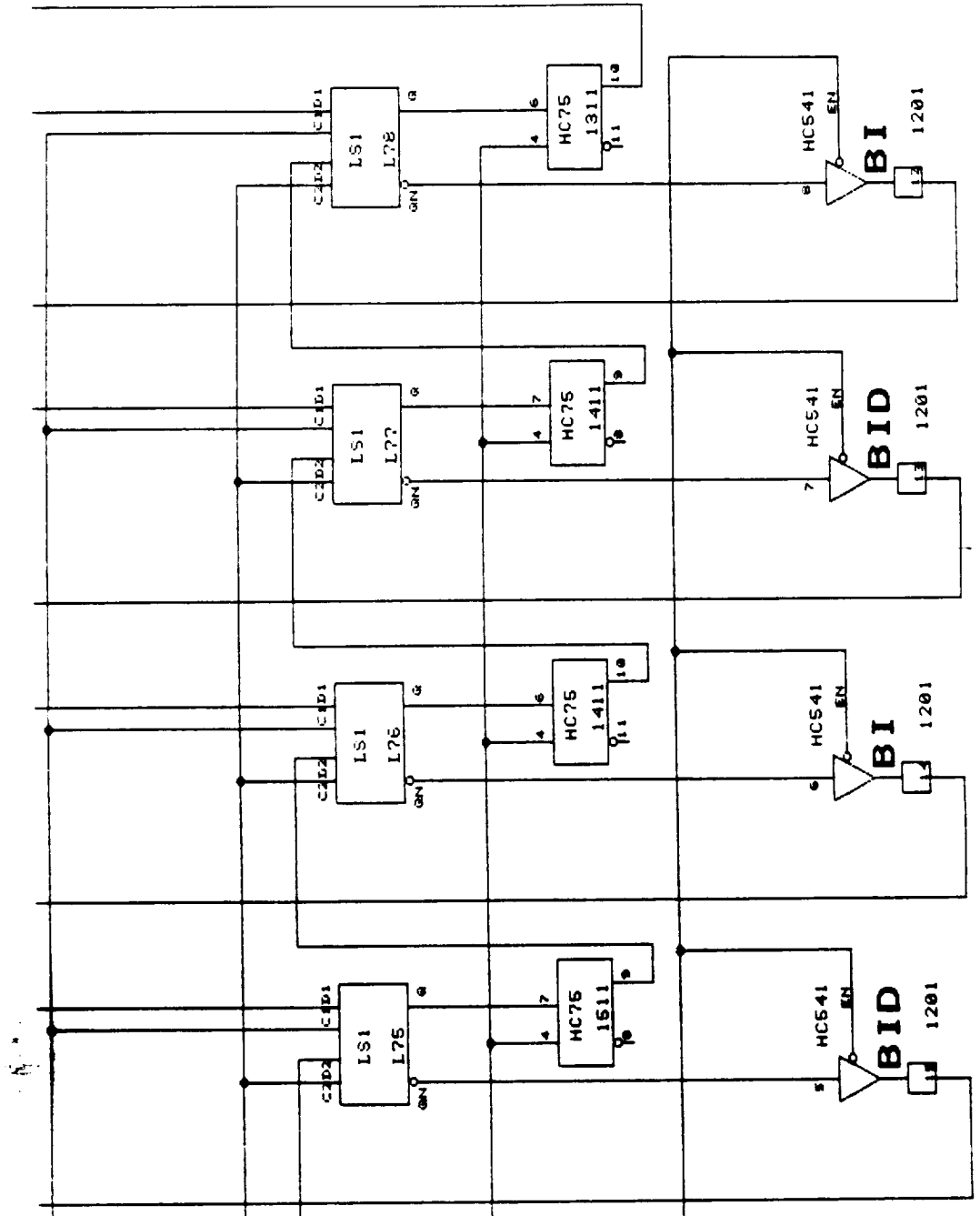


Figure 59



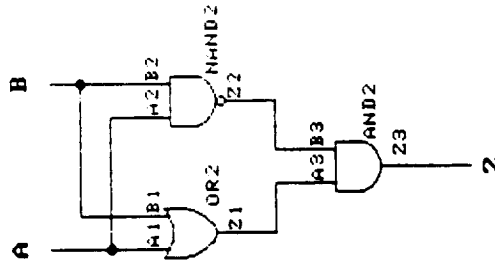
March 28 1989

EDC-BR.DWG

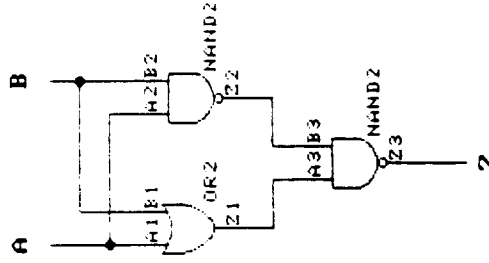
Figure 60

Realization of Exclusive-OR and Exclusive-NOR

Exclusive OR



Exclusive NOR



Position of the Exclusive-OR gates:

Nr.	OR: MS4071			NAND: MS4011			AND: MS4081		
	loc.	A1 B1 Z1		loc.	A2 B2 Z2		loc.	A3 B3 Z3	
1	3205	1 2 3		3206	1 2 3		3207	1 2 3	
2	3205	5 6 4		3206	5 6 4		3207	5 6 4	
3	3205	14 15 16		3206	14 15 16		3207	14 15 16	
4	3205	18 19 17		3206	18 19 17		3207	18 19 17	
5	3105	1 2 3		3106	1 2 3		3107	1 2 3	
6	3105	5 6 4		3106	5 6 4		3107	5 6 4	
7	3105	14 15 16		3106	14 15 16		3107	14 15 16	
8	3105	18 19 17		3106	18 19 17		3107	18 19 17	
9	3005	1 2 3		3006	1 2 3		3007	1 2 3	
10	3005	5 6 4		3006	5 6 4		3007	5 6 4	
11	3005	14 15 16		3006	14 15 16		3007	14 15 16	
12	3005	18 19 17		3006	18 19 17		3007	18 19 17	
13	2905	1 2 3		2906	1 2 3		2907	1 2 3	
14	2905	5 6 4		2906	5 6 4		2907	5 6 4	
15	2905	14 15 16		2906	14 15 16		2907	14 15 16	
16	2905	18 19 17		2906	18 19 17		2907	18 19 17	
17	2805	1 2 3		2806	1 2 3		2807	1 2 3	
18	2805	5 6 4		2806	5 6 4		2807	5 6 4	
19	2805	14 15 16		2806	14 15 16		2807	14 15 16	
20	2805	18 19 17		2806	18 19 17		2807	18 19 17	
21	2705	1 2 3		2706	1 2 3		2707	1 2 3	
22	2705	5 6 4		2706	5 6 4		2707	5 6 4	
23	2705	14 15 16		2706	14 15 16		2707	14 15 16	
24	2705	18 19 17		2706	18 19 17		2707	18 19 17	
25	2605	1 2 3		2606	1 2 3		2607	1 2 3	
26	2605	5 6 4		2606	5 6 4		2607	5 6 4	
27	2605	14 15 16		2606	14 15 16		2607	14 15 16	
28	2605	18 19 17		2606	18 19 17		2607	18 19 17	
29	2505	1 2 3		2506	1 2 3		2507	1 2 3	
30	2505	5 6 4		2506	5 6 4		2507	5 6 4	
31	2505	14 15 16		2506	14 15 16		2507	14 15 16	
32	2505	18 19 17		2506	18 19 17		2507	18 19 17	
33	2405	1 2 3		2406	1 2 3		2407	1 2 3	
34	2405	5 6 4		2406	5 6 4		2407	5 6 4	
35	2405	14 15 16		2406	14 15 16		2407	14 15 16	
36	2405	18 19 17		2406	18 19 17		2407	18 19 17	

Nr.	loc.	A1 B1 Z1	loc.	A2 B2 Z2	loc.	A3 B3 Z3
37	2305	1 2 3	2306	1 2 3	2307	1 2 3
38	2305	5 6 4	2306	5 6 4	2307	5 6 4
39	2305	14 15 16	2306	14 15 16	2307	14 15 16
40	2305	18 19 17	2306	18 19 17	2307	18 19 17
41	2205	1 2 3	2206	1 2 3	2207	1 2 3
42	2205	5 6 4	2206	5 6 4	2207	5 6 4
43	2205	14 15 16	2206	14 15 16	2207	14 15 16
44	2205	18 19 17	2206	18 19 17	2207	18 19 17
45	2105	1 2 3	2106	1 2 3	2107	1 2 3
46	2105	5 6 4	2106	5 6 4	2107	5 6 4
47	2105	14 15 16	2106	14 15 16	2107	14 15 16
48	2105	18 19 17	2106	18 19 17	2107	18 19 17
49	2005	1 2 3	2006	1 2 3	2007	1 2 3
50	2005	5 6 4	2006	5 6 4	2007	5 6 4
51	2005	14 15 16	2006	14 15 16	2007	14 15 16
52	2005	18 19 17	2006	18 19 17	2007	18 19 17
53	1905	1 2 3	1906	1 2 3	1907	1 2 3
54	1905	5 6 4	1906	5 6 4	1907	5 6 4
55	1905	14 15 16	1906	14 15 16	1907	14 15 16
56	1905	18 19 17	1906	18 19 17	1907	18 19 17
57	1805	1 2 3	1806	1 2 3	1807	1 2 3
58	1805	5 6 4	1806	5 6 4	1807	5 6 4
59	1805	14 15 16	1806	14 15 16	1807	14 15 16
60	1805	18 19 17	1806	18 19 17	1807	18 19 17
61	1705	1 2 3	1706	1 2 3	1707	1 2 3
62	1705	5 6 4	1706	5 6 4	1707	5 6 4
63	1705	14 15 16	1706	14 15 16	1707	14 15 16
64	1705	18 19 17	1706	18 19 17	1707	18 19 17
65	1605	1 2 3	1606	1 2 3	1607	1 2 3
66	1605	5 6 4	1606	5 6 4	1607	5 6 4
67	1605	14 15 16	1606	14 15 16	1607	14 15 16
68	1605	18 19 17	1606	18 19 17	1607	18 19 17
69	1505	1 2 3	1506	1 2 3	1507	1 2 3
70	1505	5 6 4	1506	5 6 4	1507	5 6 4
71	1505	14 15 16	1506	14 15 16	1507	14 15 16
72	1505	18 19 17	1506	18 19 17	1507	18 19 17
73	1405	1 2 3	1406	1 2 3	1407	1 2 3
74	1405	5 6 4	1406	5 6 4	1407	5 6 4
75	1405	14 15 16	1406	14 15 16	1407	14 15 16
76	1405	18 19 17	1406	18 19 17	1407	18 19 17

ORIGINAL PAGE IS
OF POOR QUALITY

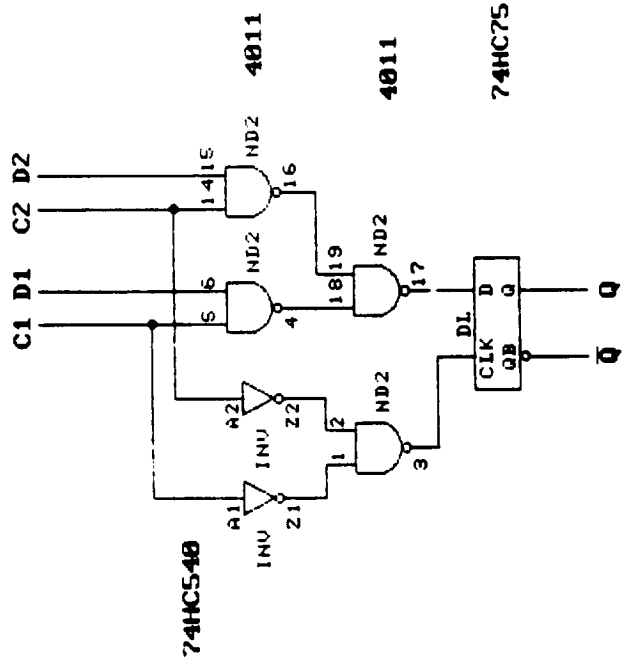
Nr.	loc.	A1	B1	Z1	loc.	A2	B2	Z2	loc.	A3	B3	Z3
77	1305	1	2	3	1306	1	2	3	1307	1	2	3
78	1305	5	6	4	1306	5	6	4	1307	5	6	4
79	1305	14	15	16	1306	14	15	16	1307	14	15	16
80	1305	18	19	17	1306	18	19	17	1307	18	19	17
81	1205	1	2	3	1206	1	2	3	1207	1	2	3
82	1205	5	6	4	1206	5	6	4	1207	5	6	4
83	1205	14	15	16	1206	14	15	16	1207	14	15	16
84	1205	18	19	17	1206	18	19	17	1207	18	19	17
85	1105	1	2	3	1106	1	2	3	1107	1	2	3
86	1105	5	6	4	1106	5	6	4	1107	5	6	4
87	1105	14	15	16	1106	14	15	16	1107	14	15	16
88	1105	18	19	17	1106	18	19	17	1107	18	19	17
89	1005	1	2	3	1006	1	2	3	1007	1	2	3
90	1005	5	6	4	1006	5	6	4	1007	5	6	4
91	1005	14	15	16	1006	14	15	16	1007	14	15	16
92	1005	18	19	17	1006	18	19	17	1007	18	19	17
93	0905	1	2	3	0906	1	2	3	0907	1	2	3
94	0905	5	6	4	0906	5	6	4	0907	5	6	4
95	0905	14	15	16	0906	14	15	16	0907	14	15	16
96	0905	18	19	17	0906	18	19	17	0907	18	19	17
97	0805	1	2	3	0806	1	2	3	0807	1	2	3
98	0805	5	6	4	0806	5	6	4	0807	5	6	4
99	0805	14	15	16	0806	14	15	16	0807	14	15	16
100	0805	18	19	17	0806	18	19	17	0907	18	19	17
101	0705	1	2	3	0706	1	2	3	0707	1	2	3
102	0705	5	6	4	0706	5	6	4	0707	5	6	4
103	0705	14	15	16	0706	14	15	16	0707	14	15	16
104	0705	18	19	17	0706	18	19	17	0707	18	19	17
105	0605	1	2	3	0606	1	2	3	0607	1	2	3
106	0605	5	6	4	0606	5	6	4	0607	5	6	4
107	0605	14	15	16	0606	14	15	16	0607	14	15	16
108	0605	18	19	17	0606	18	19	17	0607	18	19	17
109	0505	1	2	3	0506	1	2	3	0507	1	2	3
110	0505	5	6	4	0506	5	6	4	0507	5	6	4
111	0505	14	15	16	0506	14	15	16	0507	14	15	16
112	0505	18	19	17	0506	18	19	17	0507	18	19	17
113	0405	1	2	3	0406	1	2	3	0407	1	2	3
114	0405	5	6	4	0406	5	6	4	0407	5	6	4
115	0405	14	15	16	0406	14	15	16	0407	14	15	16
116	0405	18	19	17	0406	18	19	17	0407	18	19	17

Nr.	loc.	A1	B1	Z1	loc.	A2	B2	Z2	loc.	A3	B3	Z3
117	0305	1	2	3	0306	1	2	3	0307	1	2	3
118	0305	5	6	4	0306	5	6	4	0307	5	6	4
119	0305	14	15	16	0306	14	15	16	0307	14	15	16
120	0305	18	19	17	0306	18	19	17	0307	18	19	17
121	0205	1	2	3	0206	1	2	3	0207	1	2	3
122	0205	5	6	4	0206	5	6	4	0207	5	6	4
123	0205	14	15	16	0206	14	15	16	0207	14	15	16
124	0205	18	19	17	0206	18	19	17	0207	18	19	17
125	0105	1	2	3	0106	1	2	3	0107	1	2	3
126	0105	5	6	4	0106	5	6	4	0107	5	6	4
127	0105	14	15	16	0106	14	15	16	0107	14	15	16
128	0105	18	19	17	0106	18	19	17	0107	18	19	17
129	0109	1	2	3	0509	1	2	3	0309	1	2	3
130	0109	5	6	4	0509	5	6	4	0309	5	6	4
131	0109	14	15	16	0509	14	15	16	0309	14	15	16
132	0109	18	19	17	0509	18	19	17	0309	18	19	17
133	0209	1	2	3	0609	1	2	3	0409	1	2	3
134	0209	5	6	4	0609	5	6	4	0409	5	6	4

Position of the Exclusiv-NOR gate:

	OR: MS4071				NAND: MS4011				NAND: MS4011			
Nr.	loc.	A1	B1	Y1	loc.	A2	B2	Y2	loc.	A3	B3	Y3
=====												
1	0209	14	15	16	0609	14	15	16	0609	18	19	17

Realization of the scannable Latch LS1



C1	C2	Q_{n+1}	\bar{Q}_{n+1}
0	0	Q_n	\bar{Q}_n
0	1	D_2	\bar{D}_2
1	0	D_1	\bar{D}_1
1	1	0	1

CLK	Q_{n+1}	\bar{Q}_{n+1}
0	Q_n	\bar{Q}_n
1	D	\bar{D}

$$CLK = C_1 + C_2$$

$$D = C_1 * D_1 + C_2 * D_2$$

Position of the Scannable Latches:

NAND MS4011		INV: 74HC540				LATCH: 74HC75					
Nr.	loc.	loc.	A1	Z1	A2	Z2	loc.	CLK	D	Q	QB
1	3204	3201	2	18	3	17	3202	13	2	16	1
2	3104	3201	4	16	5	15	3102	13	2	16	1
3	3004	3101	2	18	3	17	3002	13	2	16	1
4	2904	3101	4	16	5	15	2902	13	2	16	1
5	2804	3001	2	18	3	17	2802	13	2	16	1
6	2704	3001	4	16	5	15	2702	13	2	16	1
7	2604	2901	2	18	3	17	2602	13	2	16	1
8	2504	2901	4	16	5	15	2502	13	2	16	1
9	2404	2801	2	18	3	17	2402	13	2	16	1
10	2304	2801	4	16	5	15	2302	13	2	16	1
11	2204	2701	2	18	3	17	2202	13	2	16	1
12	2104	2701	4	16	5	15	2102	13	2	16	1
13	2004	2601	2	18	3	17	2002	13	2	16	1
14	1904	2601	4	16	5	15	1902	13	2	16	1
15	1804	2501	2	18	3	17	1802	13	2	16	1
16	1704	2501	4	16	5	15	1702	13	2	16	1
17	1604	2401	2	18	3	17	1602	13	2	16	1
18	1504	2401	4	16	5	15	1502	13	2	16	1
19	1404	2301	2	18	3	17	1402	13	2	16	1
20	1304	2301	4	16	5	15	1302	13	2	16	1
21	1204	2201	2	18	3	17	1202	13	2	16	1
22	1104	2201	4	16	5	15	1102	13	2	16	1
23	1004	2101	2	18	3	17	1002	13	2	16	1
24	0904	2101	4	16	5	15	0902	13	2	16	1
25	0804	2001	2	18	3	17	0802	13	2	16	1
26	0704	2001	4	16	5	15	0702	13	2	16	1
27	0604	1901	2	18	3	17	0602	13	2	16	1
28	0504	1901	4	16	5	15	0502	13	2	16	1
29	0404	1801	2	18	3	17	0402	13	2	16	1
30	0304	1801	4	16	5	15	0302	13	2	16	1
31	0204	1701	2	18	3	17	0202	13	2	16	1
32	0104	1701	4	16	5	15	0102	13	2	16	1
33	3203	3201	6	14	7	13	3202	4	6	10	11
34	3103	3201	8	12	9	11	3102	4	6	10	11
35	3003	3101	6	14	7	13	3002	4	6	10	11
36	2903	3101	8	12	9	11	2902	4	6	10	11

Nr.	loc.	loc.	A1	Z1	A2	Z2	loc.	CLK	D	Q	QB
37	2803	3001	6	14	7	13	2802	4	6	10	11
38	2703	3001	8	12	9	11	2702	4	6	10	11
39	2603	2901	6	14	7	13	2602	4	6	10	11
40	2503	2901	8	12	9	11	2502	4	6	10	11
41	2403	2801	6	14	7	13	2402	4	6	10	11
42	2303	2801	8	12	9	11	2302	4	6	10	11
43	2203	2701	6	14	7	13	2202	4	6	10	11
44	2103	2701	8	12	9	11	2102	4	6	10	11
45	2003	2601	6	14	7	13	2002	4	6	10	11
46	1903	2601	8	12	9	11	1902	4	6	10	11
47	1803	2501	6	14	7	13	1802	4	6	10	11
48	1703	2501	8	12	9	11	1702	4	6	10	11
49	1603	2401	6	14	7	13	1602	4	6	10	11
50	1503	2401	8	12	9	11	1502	4	6	10	11
51	1403	2301	6	14	7	13	1402	4	6	10	11
52	1303	2301	8	12	9	11	1302	4	6	10	11
53	1203	2201	6	14	7	13	1202	4	6	10	11
54	1103	2201	8	12	9	11	1102	4	6	10	11
55	1003	2101	6	14	7	13	1002	4	6	10	11
56	0903	2101	8	12	9	11	0902	4	6	10	11
57	0803	2001	6	14	7	13	0802	4	6	10	11
58	0703	2001	8	12	9	11	0702	4	6	10	11
59	0603	1901	6	14	7	13	0602	4	6	10	11
60	0503	1901	8	12	9	11	0502	4	6	10	11
61	0403	1801	6	14	7	13	0402	4	6	10	11
62	0303	1801	8	12	9	11	0302	4	6	10	11
63	0203	1701	6	14	7	13	0202	4	6	10	11
64	0103	1701	8	12	9	11	0102	4	6	10	11
65	3208	2009	2	18	3	17	2909	13	2	16	1
66	3008	2009	4	16	5	15	2809	13	2	16	1
67	2808	1909	2	18	3	17	2709	13	2	16	1
68	2608	1909	4	16	5	15	2609	13	2	16	1
69	2408	1809	2	18	3	17	2509	13	2	16	1
70	2208	1809	4	16	5	15	2409	13	2	16	1
71	2008	1709	2	18	3	17	2309	13	2	16	1
72	3108	1709	4	16	5	15	2909	4	6	10	11
73	2908	2009	6	14	7	13	2809	4	6	10	11
74	2708	2009	8	12	9	11	2709	4	6	10	11
75	2508	1909	6	14	7	13	2609	4	6	10	11
76	2308	1909	8	12	9	11	2509	4	6	10	11

Nr.	loc.	loc.	A1	Z1	A2	Z2	loc.	CLK	D	Q	QB
77	2108	1809	6	14	7	13	2409	4	6	10	11
78	1908	1809	8	12	9	11	2309	4	6	10	11
79	1808	1709	6	14	7	13	2209	13	2	16	1
80	1708	1709	8	12	9	11	2209	4	6	10	11
81	1608	1609	2	18	3	17	2109	4	6	10	11

Connector Assignment

CONN 2A

1	DATA 0
2	DATA 1
3	DATA 2
4	DATA 3
5	DATA 4
6	DATA 5
7	DATA 6
8	DATA 7
9	DATA 8
10	DATA 9
11	DATA 10
12	DATA 11
13	DATA 12
14	DATA 13
15	DATA 14
16	DATA 15
17	
18	

CONNECTOR C1

CONN 2B

1	DATA 16
2	DATA 17
3	DATA 18
4	DATA 19
5	DATA 20
6	DATA 21
7	DATA 22
8	DATA 23
9	DATA 24
10	DATA 25
11	DATA 26
12	DATA 27
13	DATA 28
14	DATA 29
15	DATA 30
16	DATA 31
17	
18	

CONNECTOR C2

CONN 2C

1	CB/SY 0
2	CB/SY 1
3	CB/SY 2
4	CB/SY 3
5	CB/SY 4
6	CB/SY 5
7	CB/SY 6
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	

CONNECTOR C3

CONN 1A

1	LE DATA IN
2	LE R/U
3	LE DATA OUT
4	LE CB/SY
5	LE ERROR
6	LE HE
7	FDI
8	FCLK
9	reserved
10	OE DATA
11	OE SC
12	CLK A
13	CLK B
14	CORRECT
15	R/U
16	reserved
17	FE1
18	FE2

CONNECTOR C4

CONN 1B

1	
2	
3	
4	
5	
6	
7	
8	
9	OE BYTE 0
10	OE BYTE 1
11	OE BYTE 2
12	OE BYTE 3
13	OE SC
14	
15	
16	
17	
18	

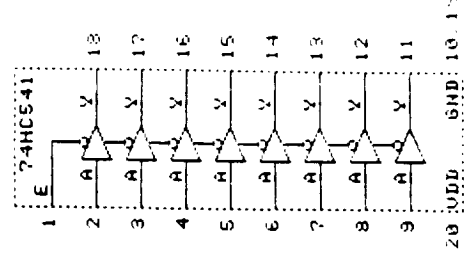
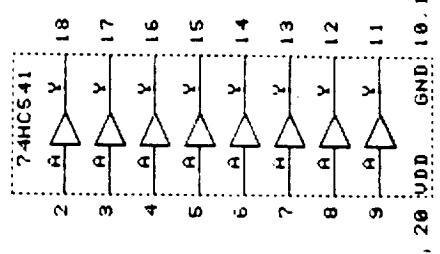
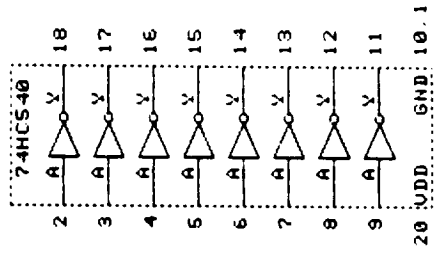
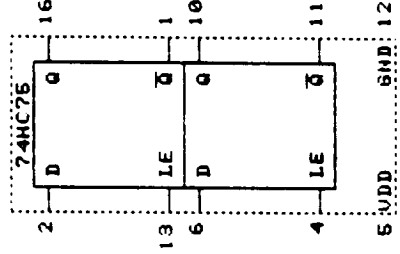
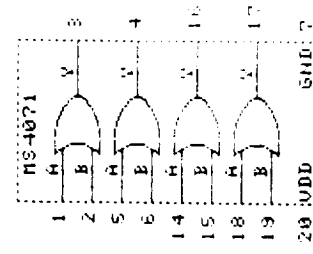
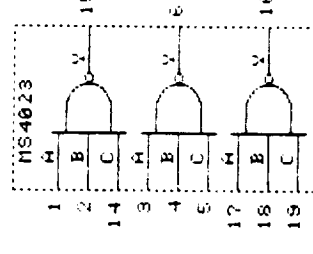
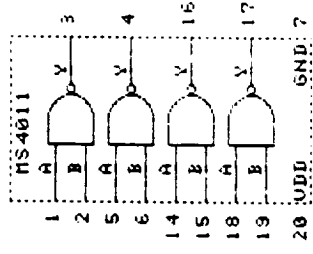
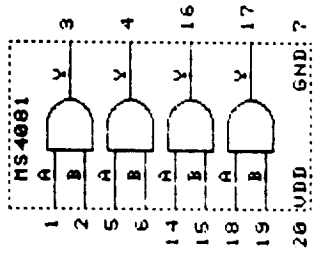
NO EX1 CONNECTOR

CONN 1C

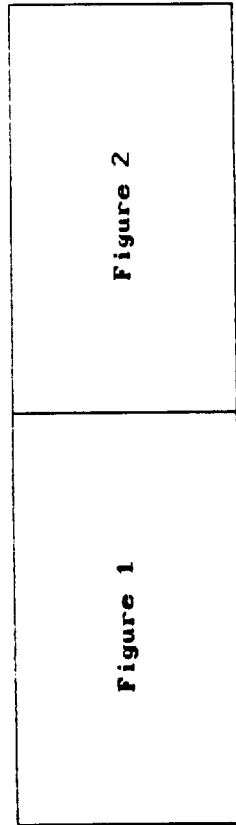
1	MULT ERROR
2	ERROR
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	

CONNECTOR C5

Pin Assignments



Map of IDT 49C460 Adapter



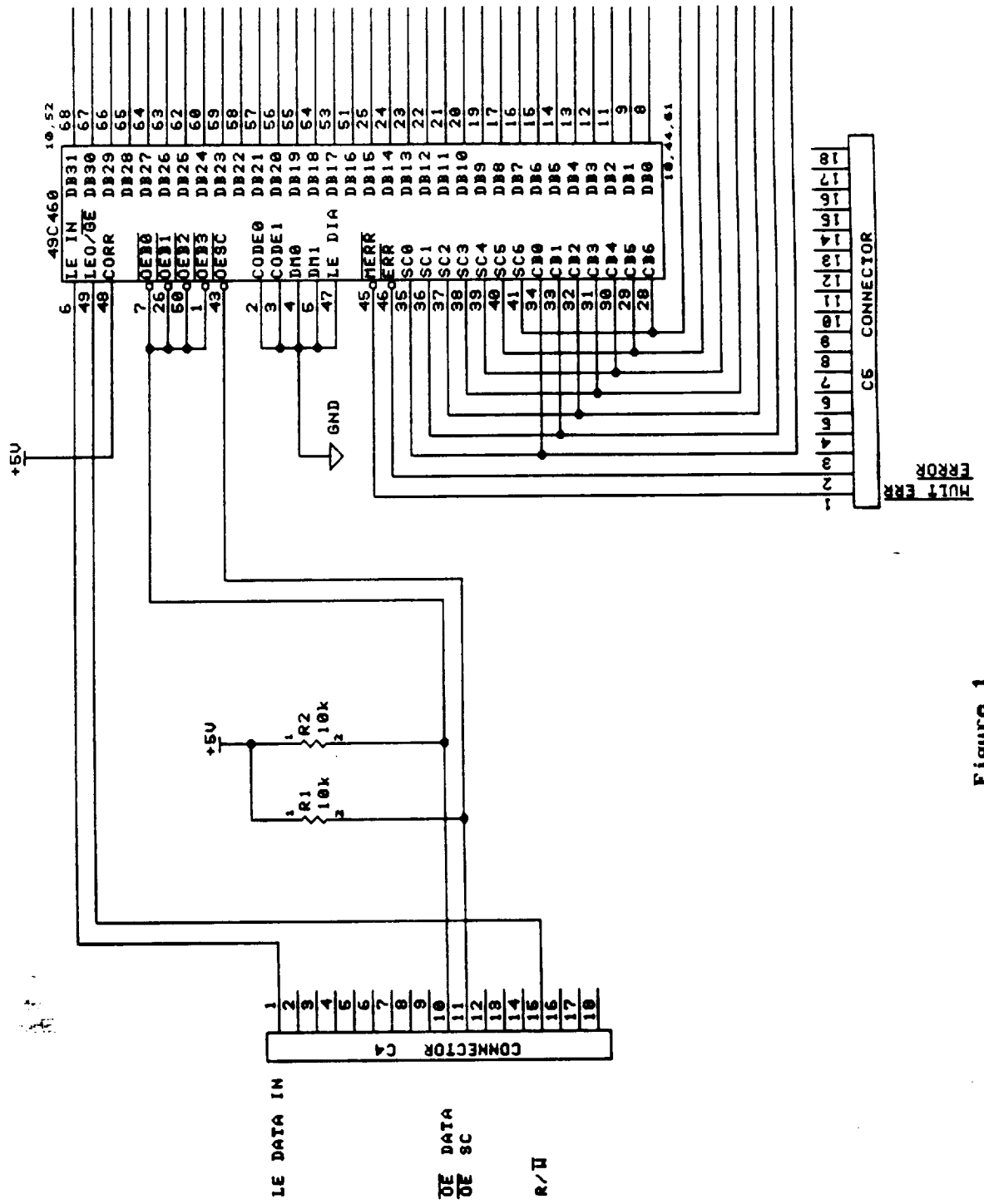


Figure 1

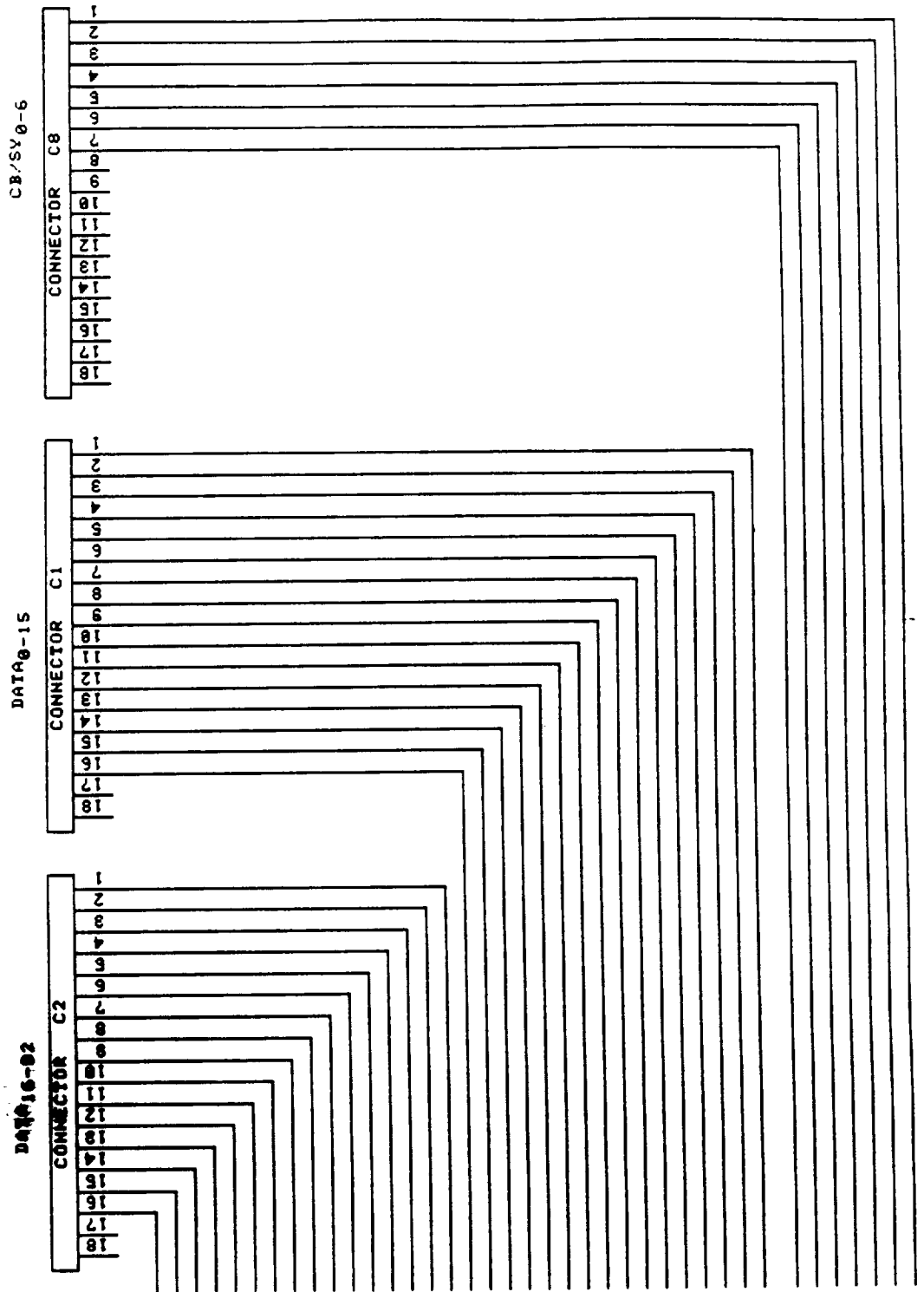
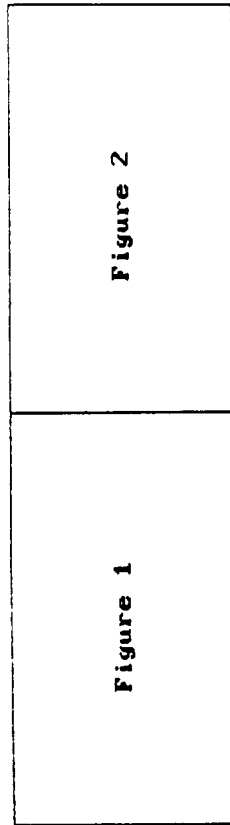


Figure 2

Map of TI 74ALS632 Adapter



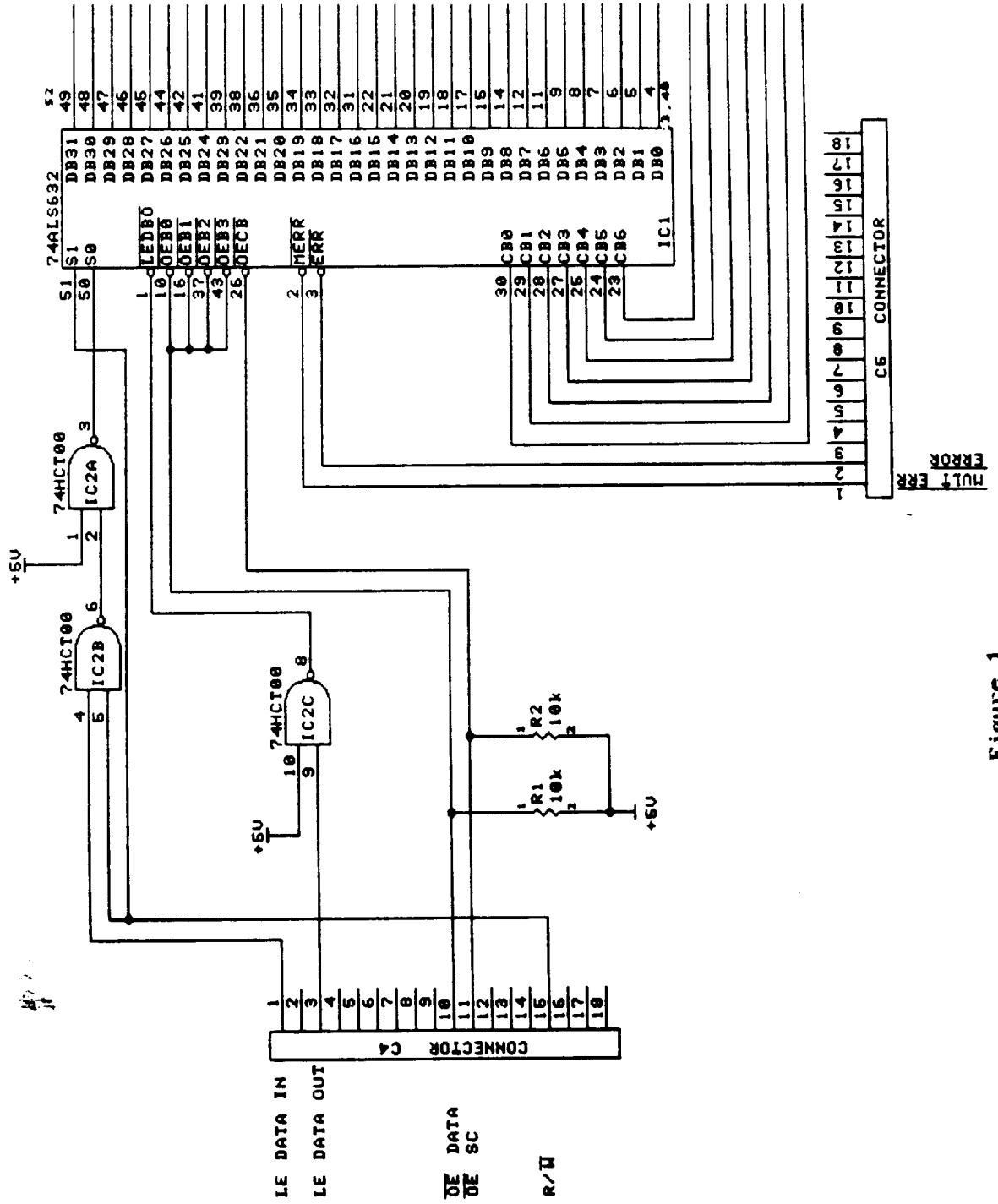


Figure 1

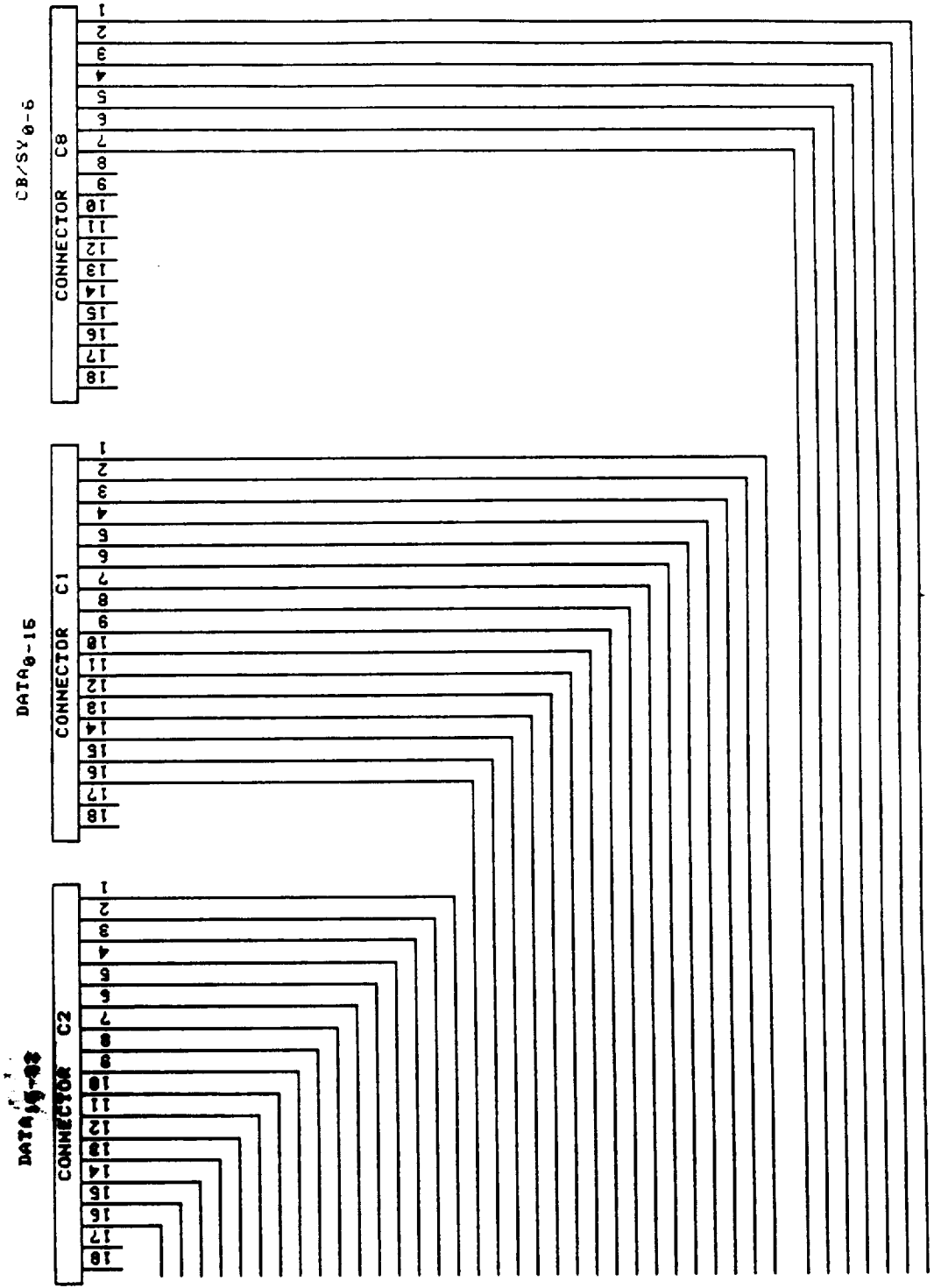


Figure 2

EDAC - Error Detection and Correction Unit

Testing of the Parity Tree Network - exhaustive

	Status	# vectors tested
Checkbit 0	ok	8192
Checkbit 1	ok	16384
Checkbit 2	ok	16384
Checkbit 3	ok	16384
Checkbit 4	ok	16384
Checkbit 5	ok	16384
Checkbit 6	ok	8192

Test completed successfully! No Errors detected!

Teststart: 08:28:31

Teststop: 09:09:26

Testtime: 00:39:55

F9-Stop F10-Exit

EDAC - Error Detection and Correction Unit
Testing of the Error Detector - exhaustive

Status	# vectors tested
ok	128

Test completed successfully! No Errors detected!

Teststart: 09:35:55 Teststop: 09:35:55 Testtime: 00:00:00

F10-Exit

EDAC - Error Detection and Correction Unit
Testing of the Error Locator - exhaustive

Status	# vectors tested
ok	32

Test completed successfully! No Errors detected!

Teststart: 09:50:08 Teststop: 09:50:08 Testtime: 00:00:00

F10-Exit

APPENDIX D: FAULT SIMULATION

D.1 Fault Simulator Breadboard Layout

D.1.1 Map of Fault Simulator Breadboard Layout....D-1

D.1.2 Figure 1.....D-2

D.1.3 Figure 2.....D-3

D.1.4 Figure 3.....D-4

D.1.5 Figure 4.....D-5

D.1.6 Figure 5.....D-6

D.1.7 Figure 6.....D-7

D.1.8 Figure 7.....D-8

D.1.9 Figure 8.....D-9

D.1.10 Figure 9.....D-10

D.1.11 Figure 10.....D-11

D.1.12 Figure 11.....D-12

D.1.13 Figure 12.....D-13

D.1.14 Figure 13.....D-14

D.1.15 Figure 14.....D-15

D.1.16 Figure 15.....D-16

D.1.17 Figure 16.....D-17

D.1.18 Figure 17.....D-18

D.1.19 Figure 18.....D-19

D.1.20 Figure 19.....D-20

D.1.21 Figure 20.....D-21

D.1.22 Figure 21.....D-22

D.1.23 Figure 22.....D-23

D.1.24 Figure 23.....D-24

D.1.25 Figure 24.....D-25

D.2 Microsimulator

- D.2.1 SP4HFS011: 2-Input NAND Gates.....D-26
- D.2.2 SP4HFS023: 3-Input NAND Gates.....D-29
- D.2.3 SP4HFS071: 2-Input OR Gates.....D-32
- D.2.4 SP4HFS081: 2-Input AND Gates.....D-35

D.3 Fault Simulation

- D.3.1 Combinational Logic.....D-38
- D.3.2 Scannable Latches.....D-39
- D.3.3 Results of Stuck-at Fault Simulation
 - D.3.3.1 Combinational Logic: 10 cycles.....D-40
 - D.3.3.2 Combinational Logic: 20 cycles.....D-41
 - D.3.3.3 Combinational Logic: 100 cycles.....D-42
 - D.3.3.4 Combinational Logic: 300 cycles.....D-43
 - D.3.3.5 Combinational Logic: 400 cycles.....D-44
 - D.3.3.6 Scannable Latches: 2 cycles.....D-45
 - D.3.3.7 Scannable Latches: 4 cycles.....D-46
 - D.3.3.8 Scannable Latches: 6 cycles.....D-47
- D.3.4 Results of Stuck-open Fault Simulation
 - D.3.4.1 Combinational Logic: 10 cycles.....D-48
 - D.3.4.2 Combinational Logic: 100 cycles.....D-49
 - D.3.4.3 Combinational Logic: 400 cycles.....D-50
 - D.3.4.4 Combinational Logic: 1000 cycles.....D-51
 - D.3.4.5 Combinational Logic: 10000 cycles.....D-52
 - D.3.4.6 Combinational Logic: 100000 cycles.....D-53

MAP of EDAC Fault Simulator Layout

Figure 1	Figure 2	Figure 3	Figure 4
Figure 5	Figure 6	Figure 7	Figure 8
Figure 9	Figure 10	Figure 11	Figure 12
Figure 13	Figure 14	Figure 15	Figure 16
Figure 17	Figure 18	Figure 19	Figure 20
Figure 21	Figure 22	Figure 23	Figure 24

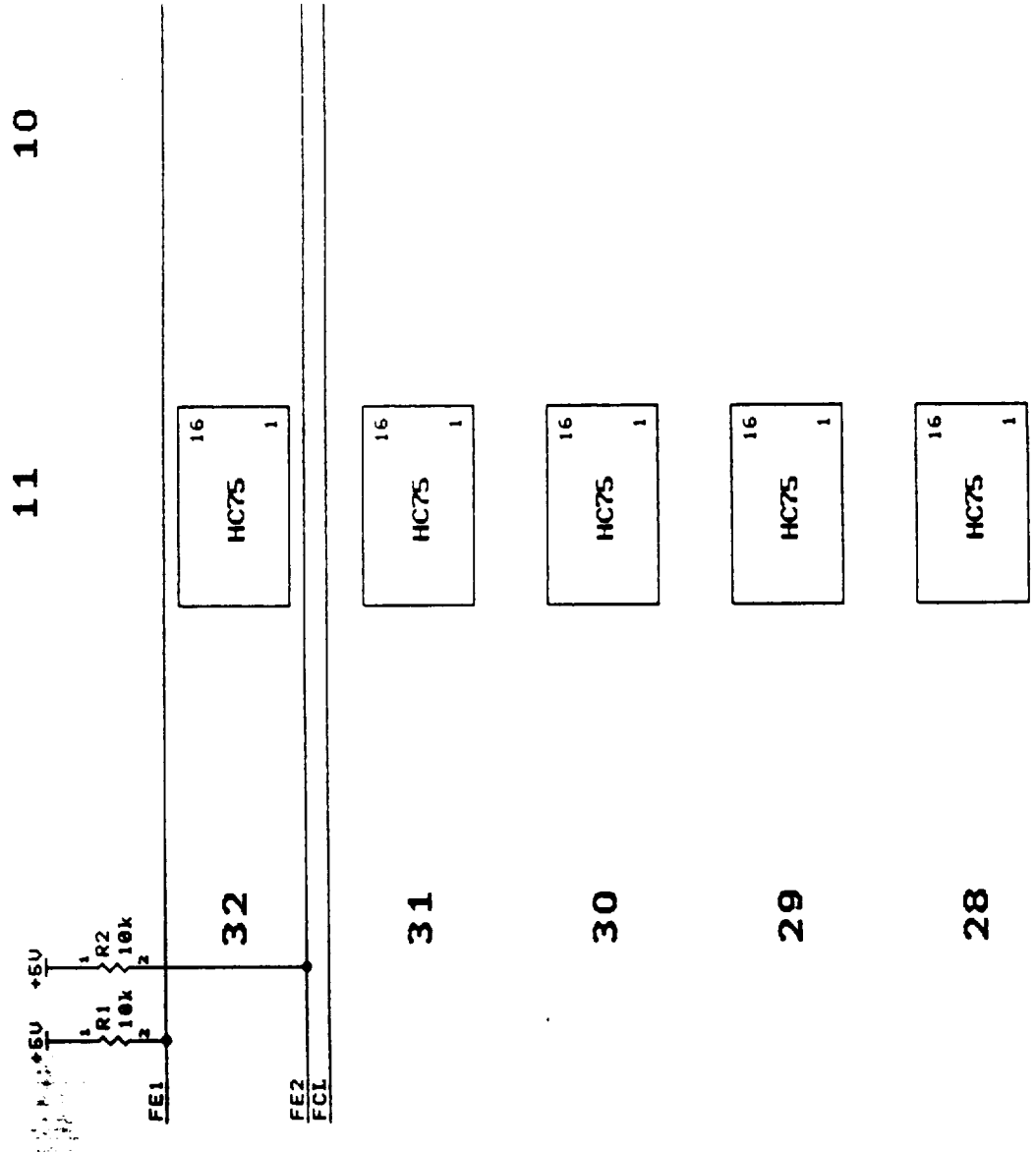


Figure 1

07

08

09

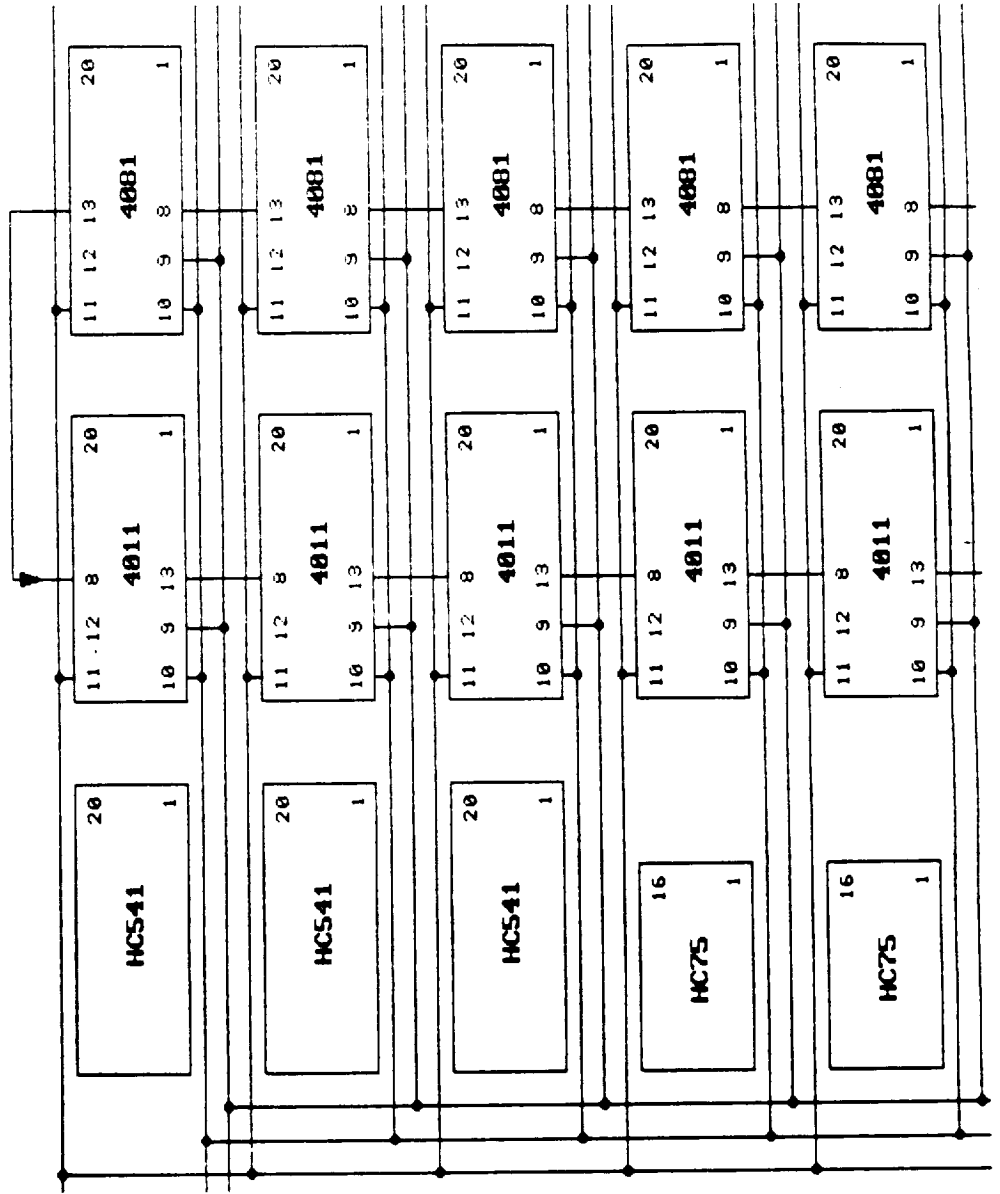


Figure 2

04

05

06

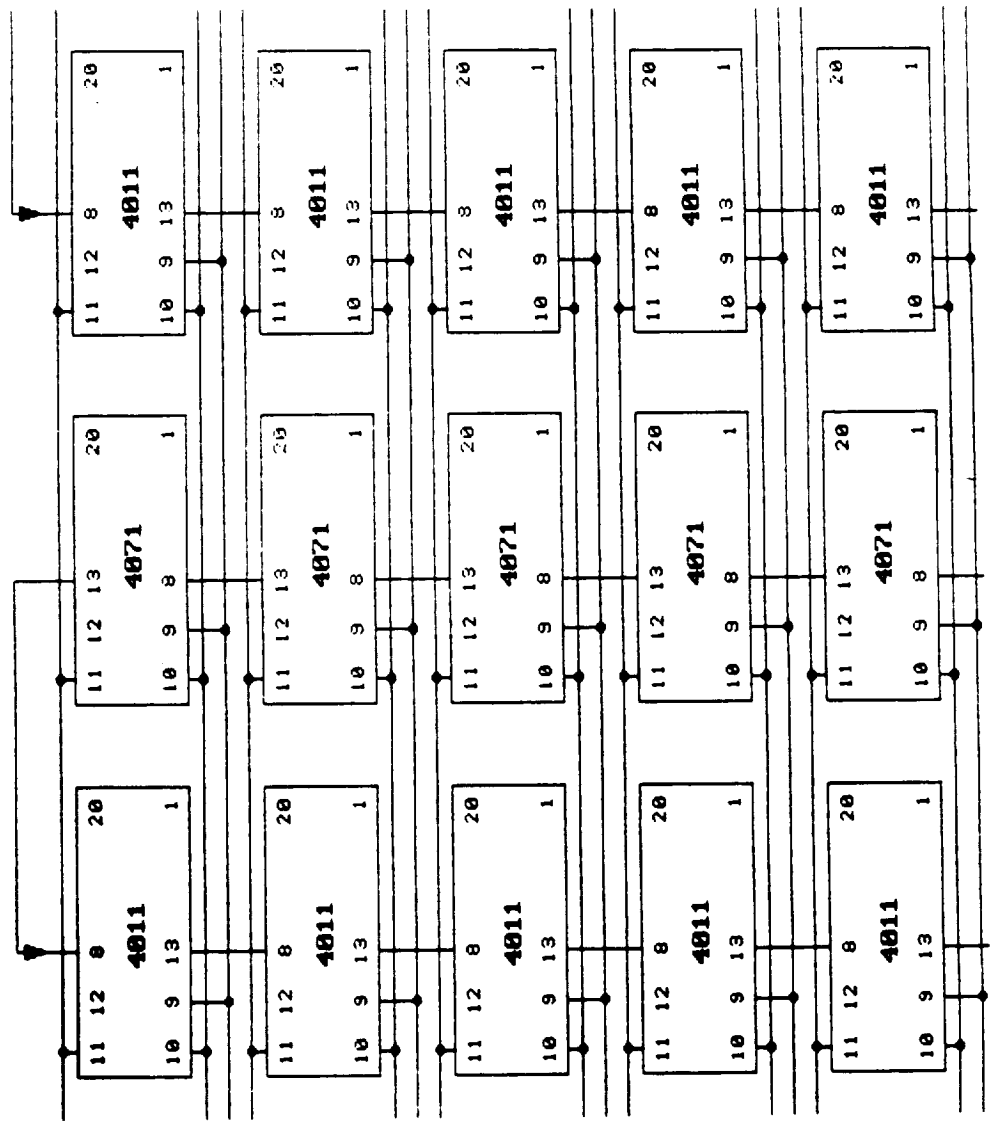
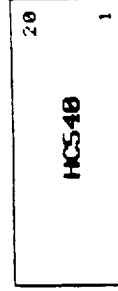
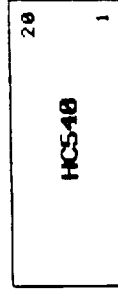
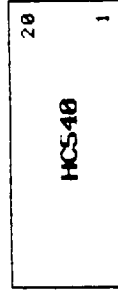
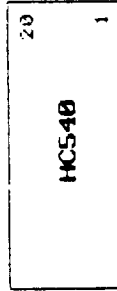
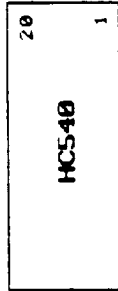
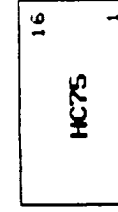
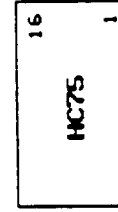
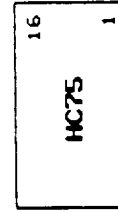
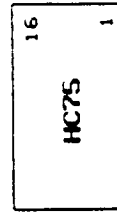
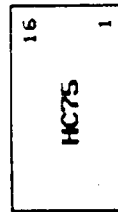


Figure 3

01



02



03

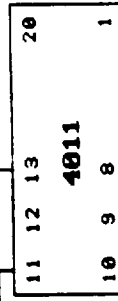
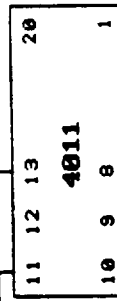
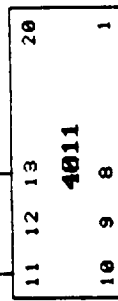
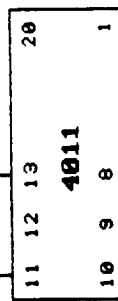
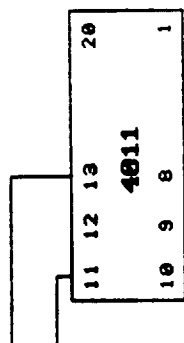


Figure 4

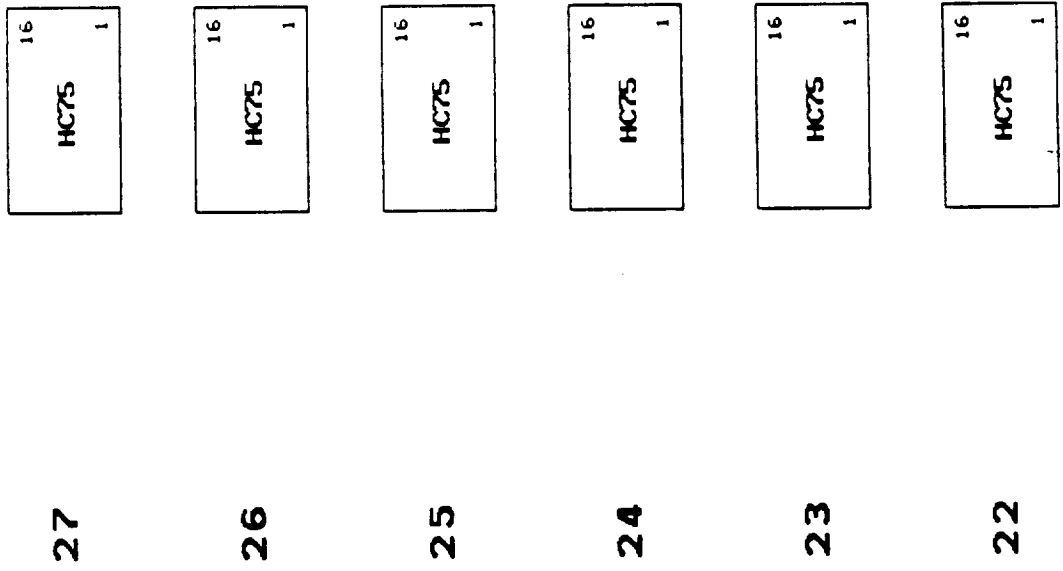


Figure 5

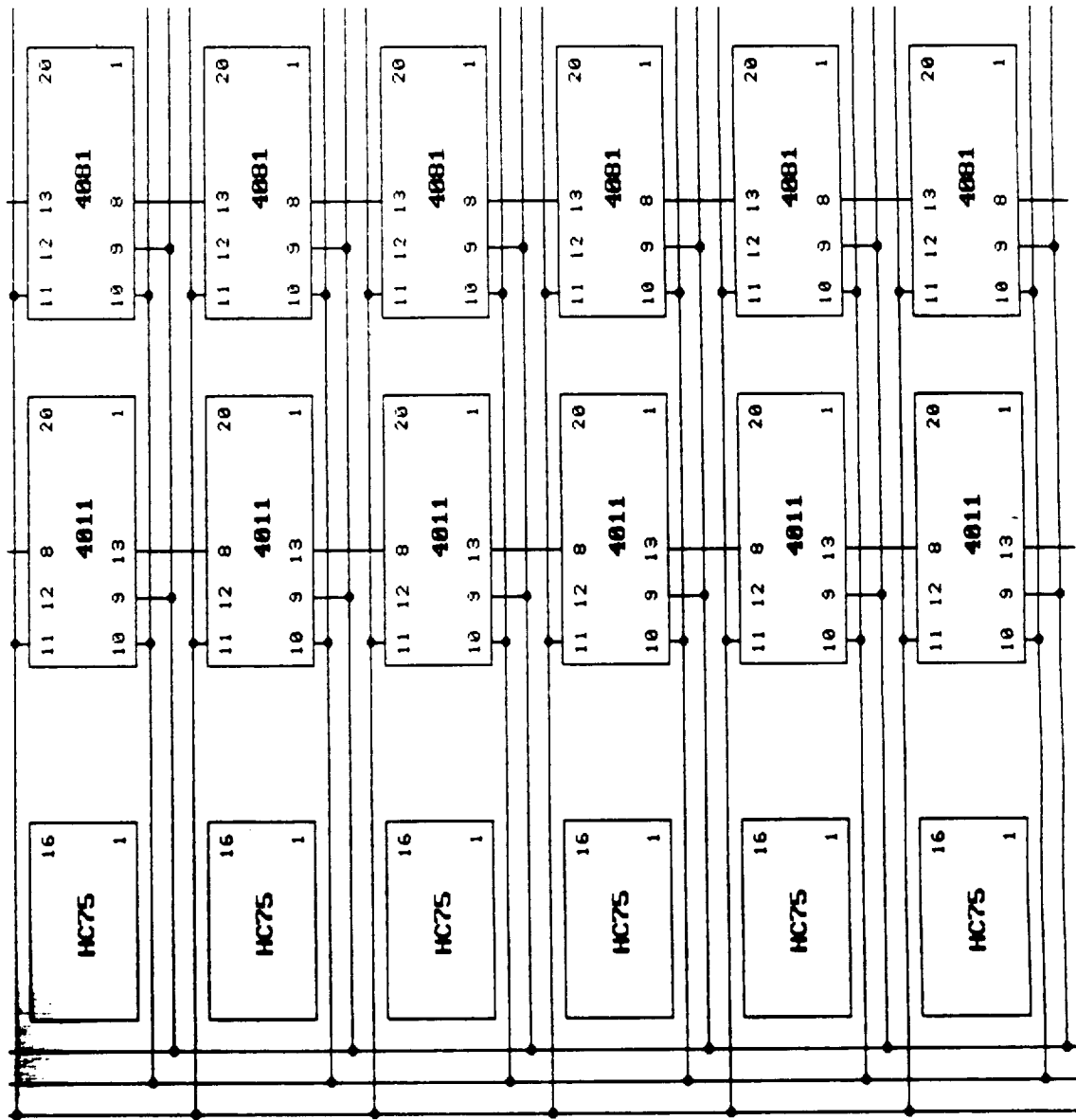


Figure 6

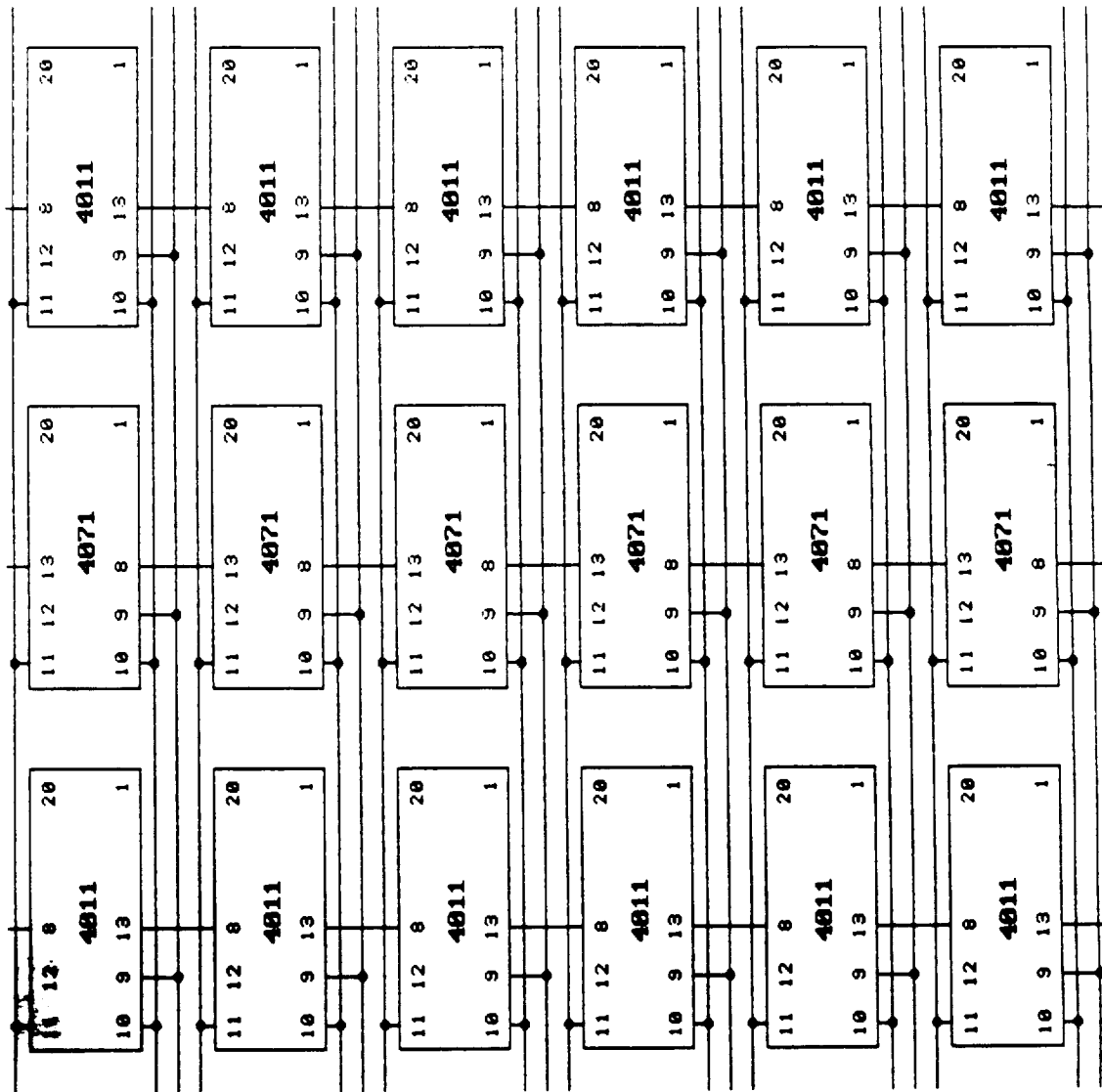


Figure 7

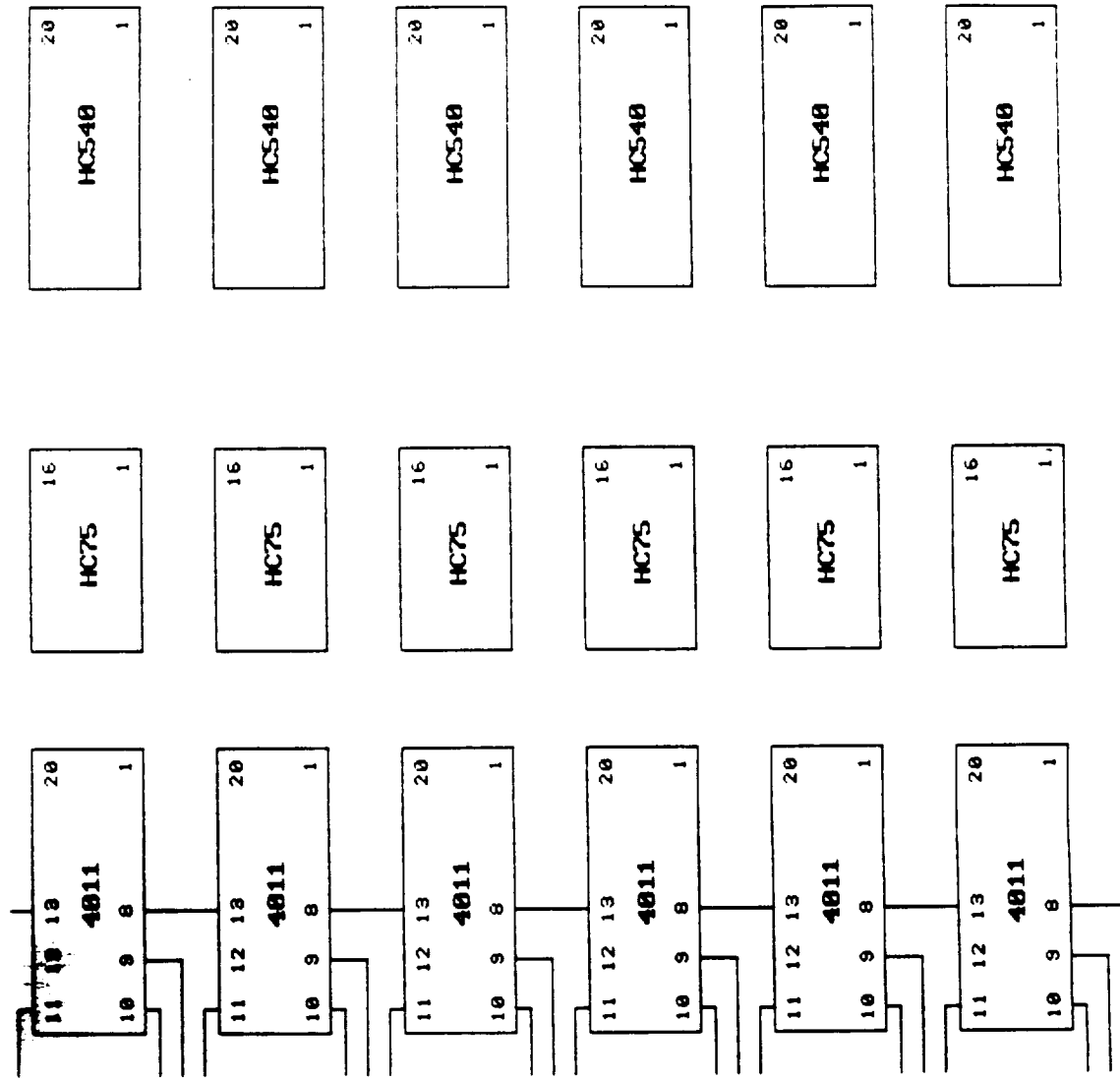


Figure 8

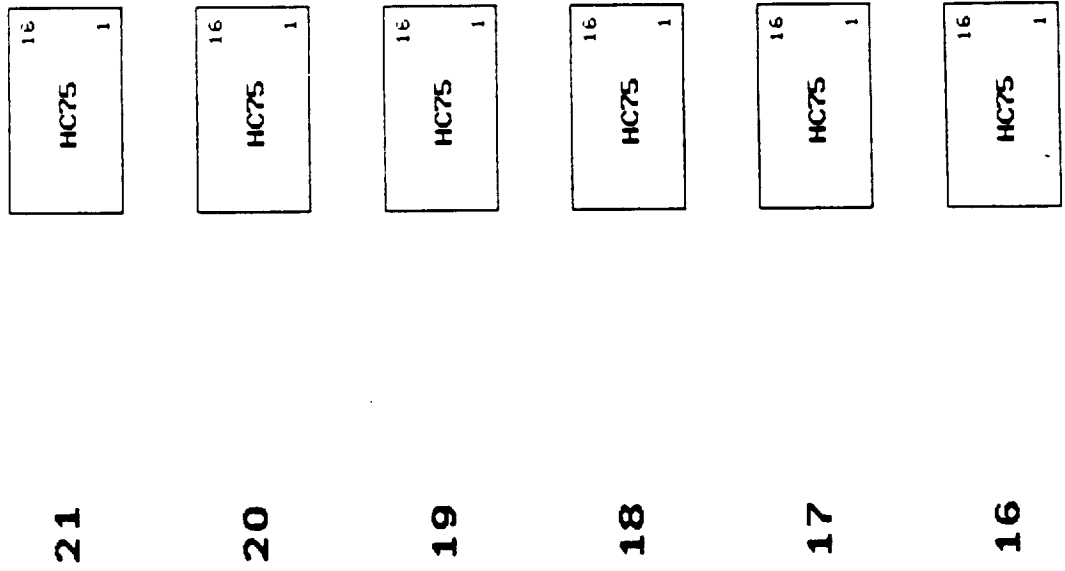


Figure 9

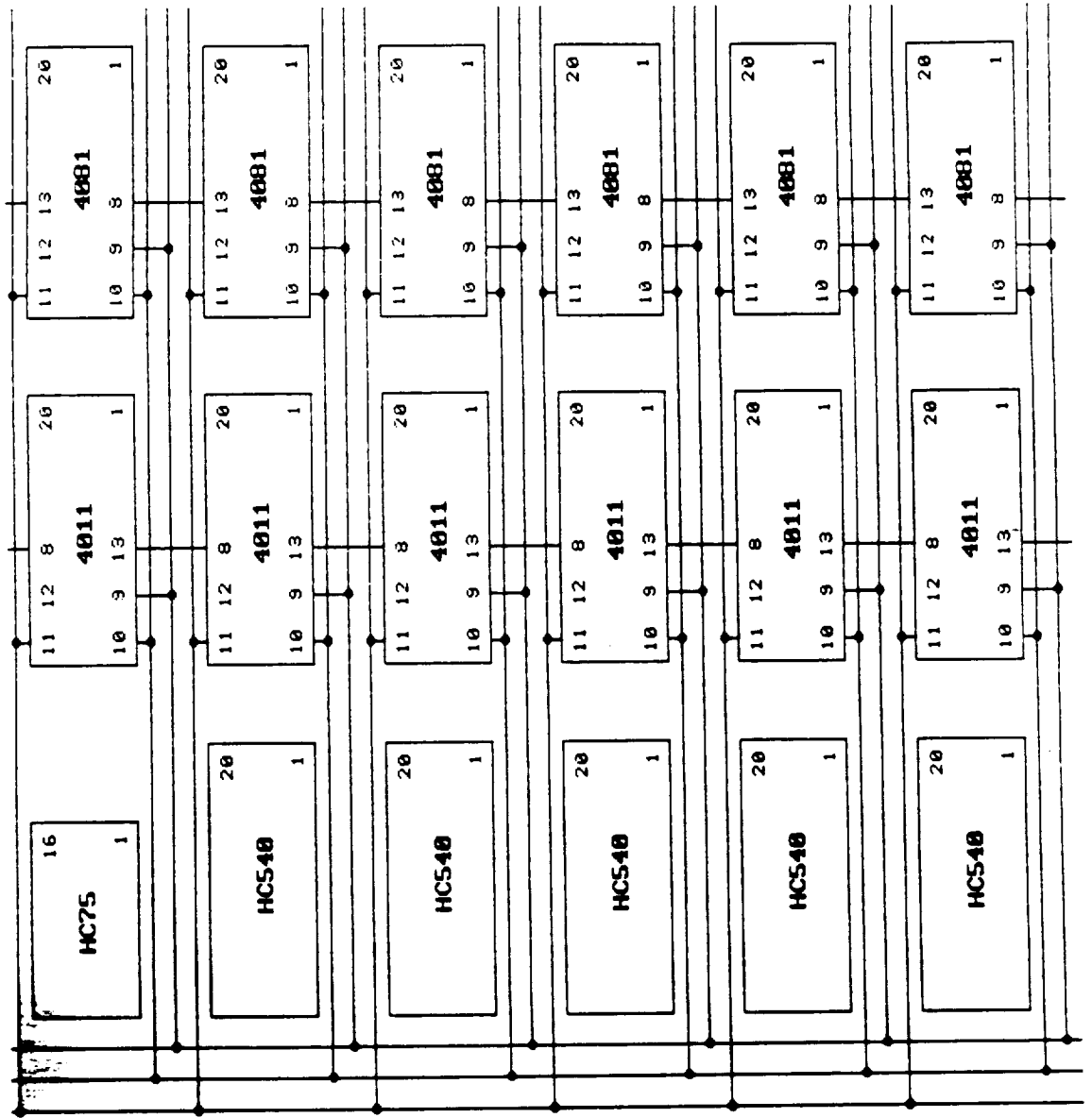


Figure 10

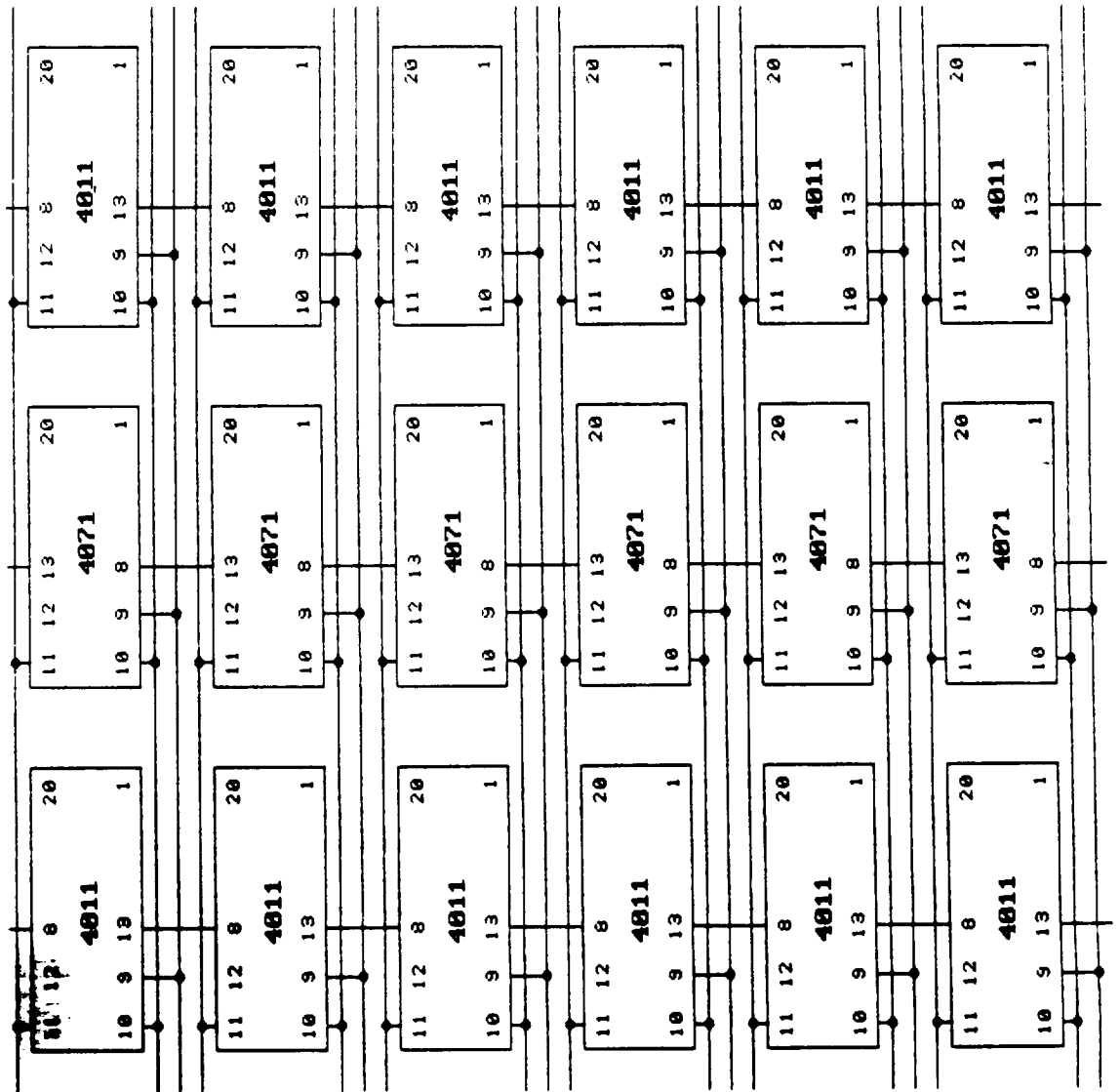


Figure 11

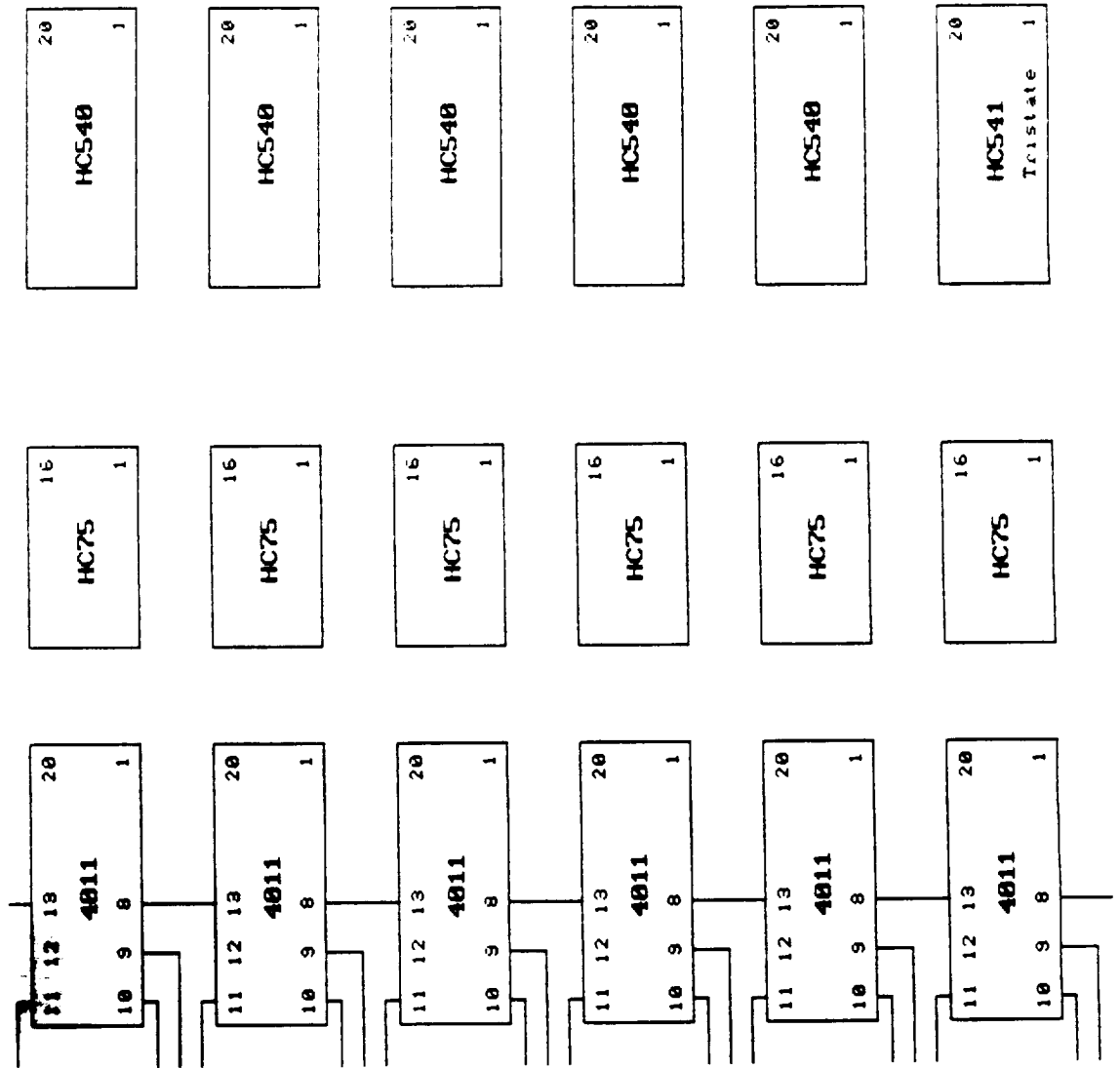


Figure 12

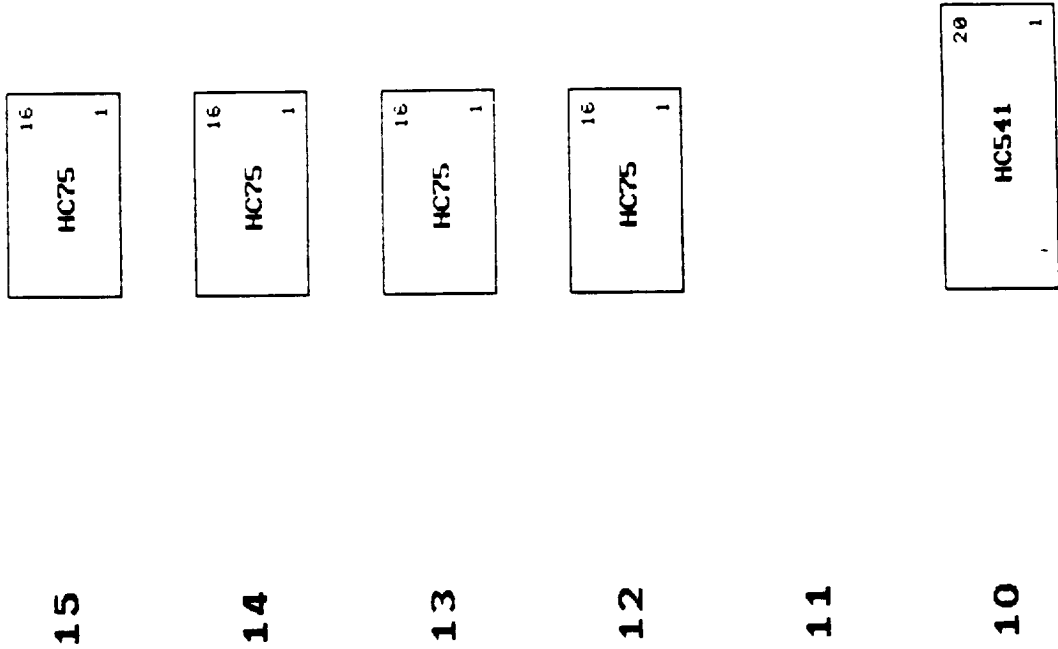


Figure 13

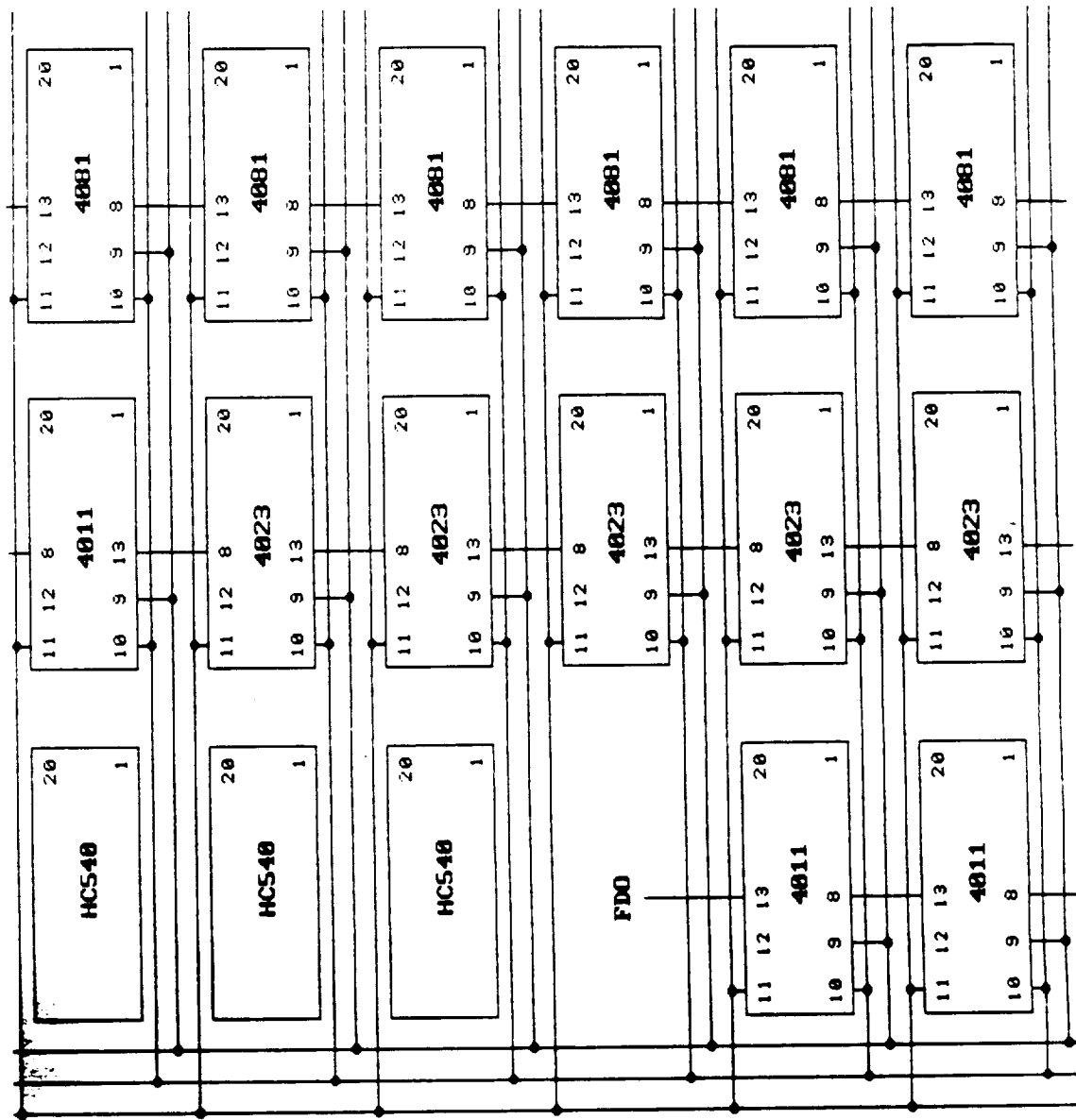


Figure 14

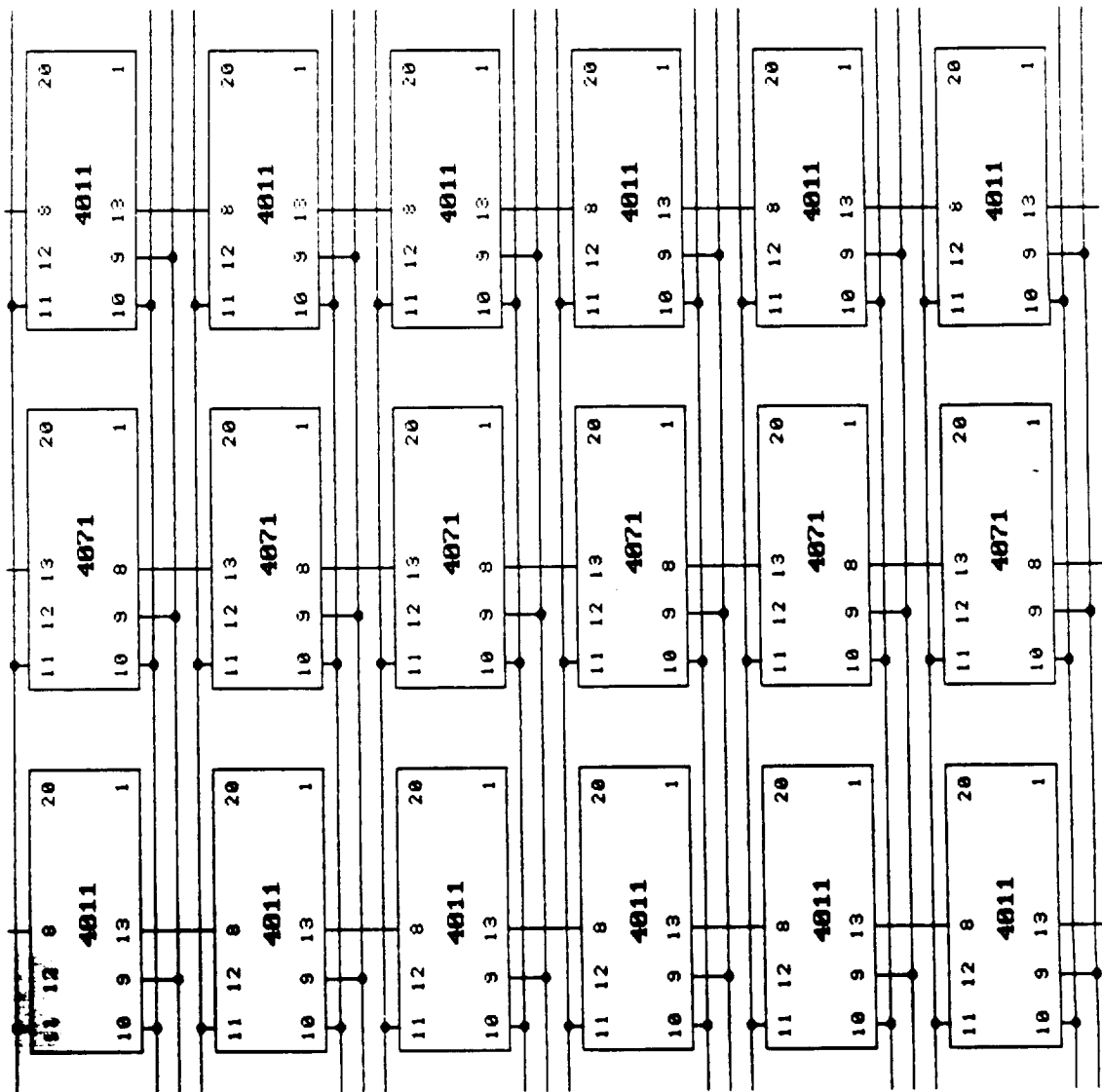


Figure 15

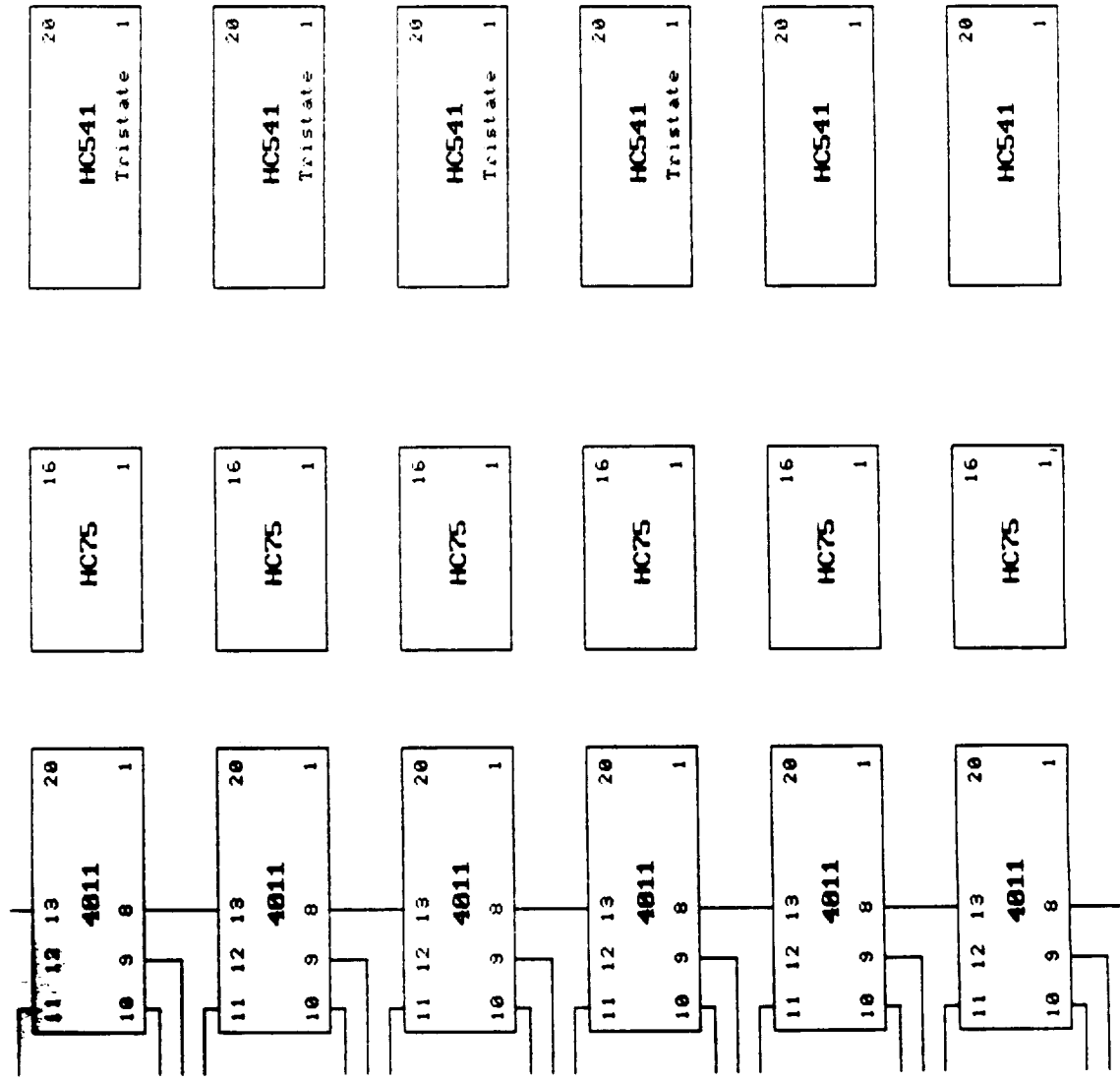


Figure 16

HC541
20
1

09

08

07

06

05

04

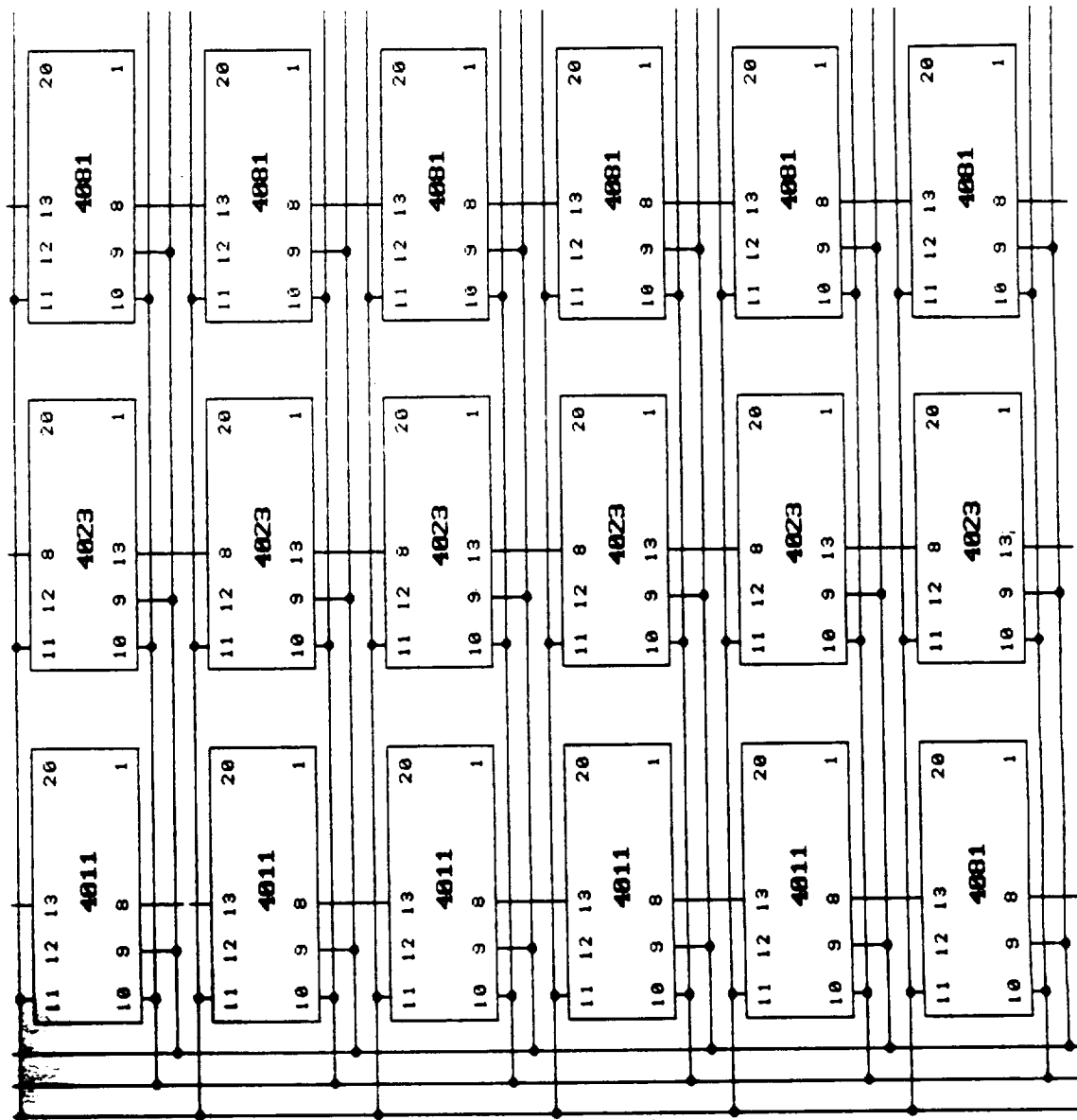


Figure 18

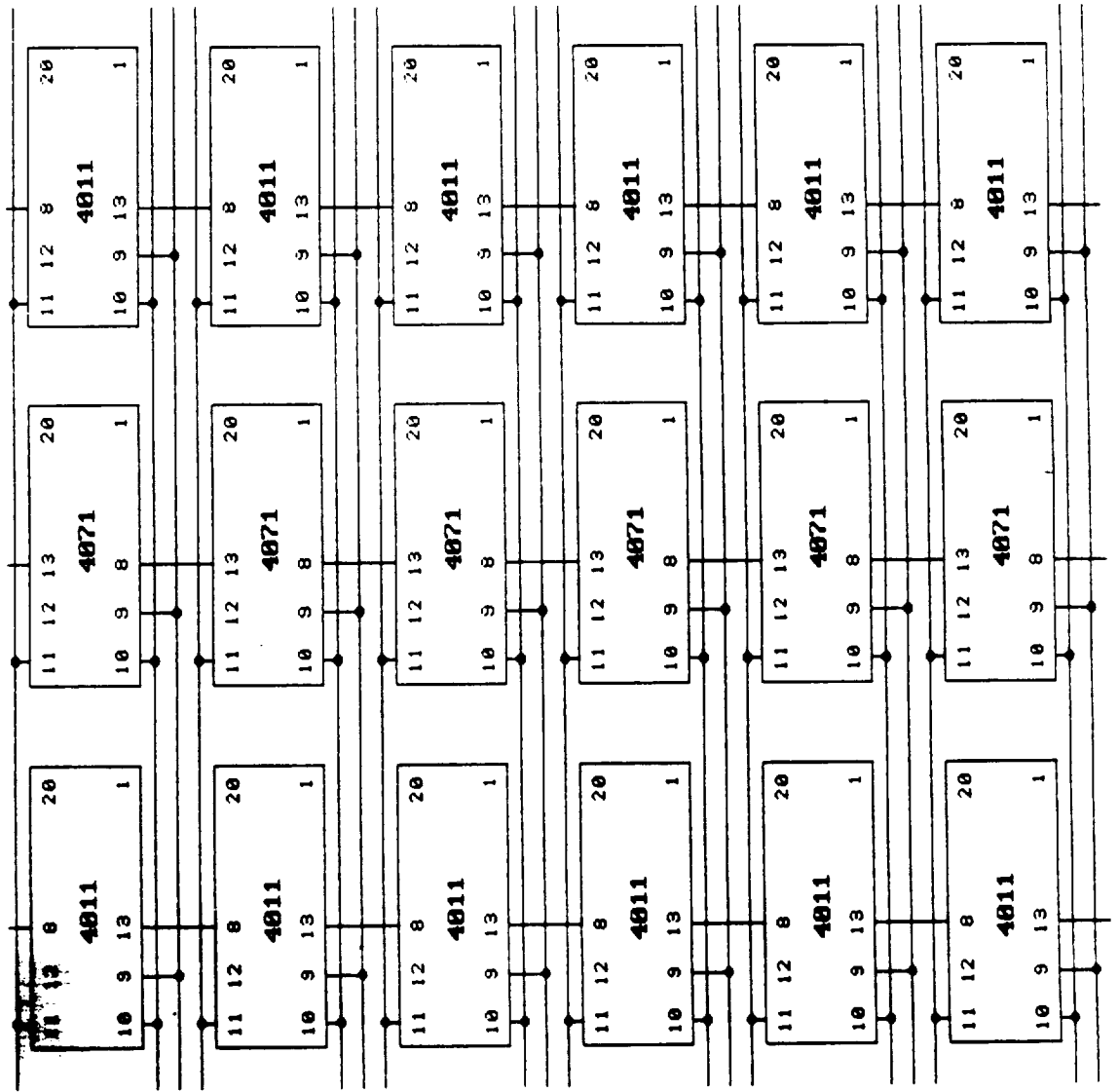


Figure 19

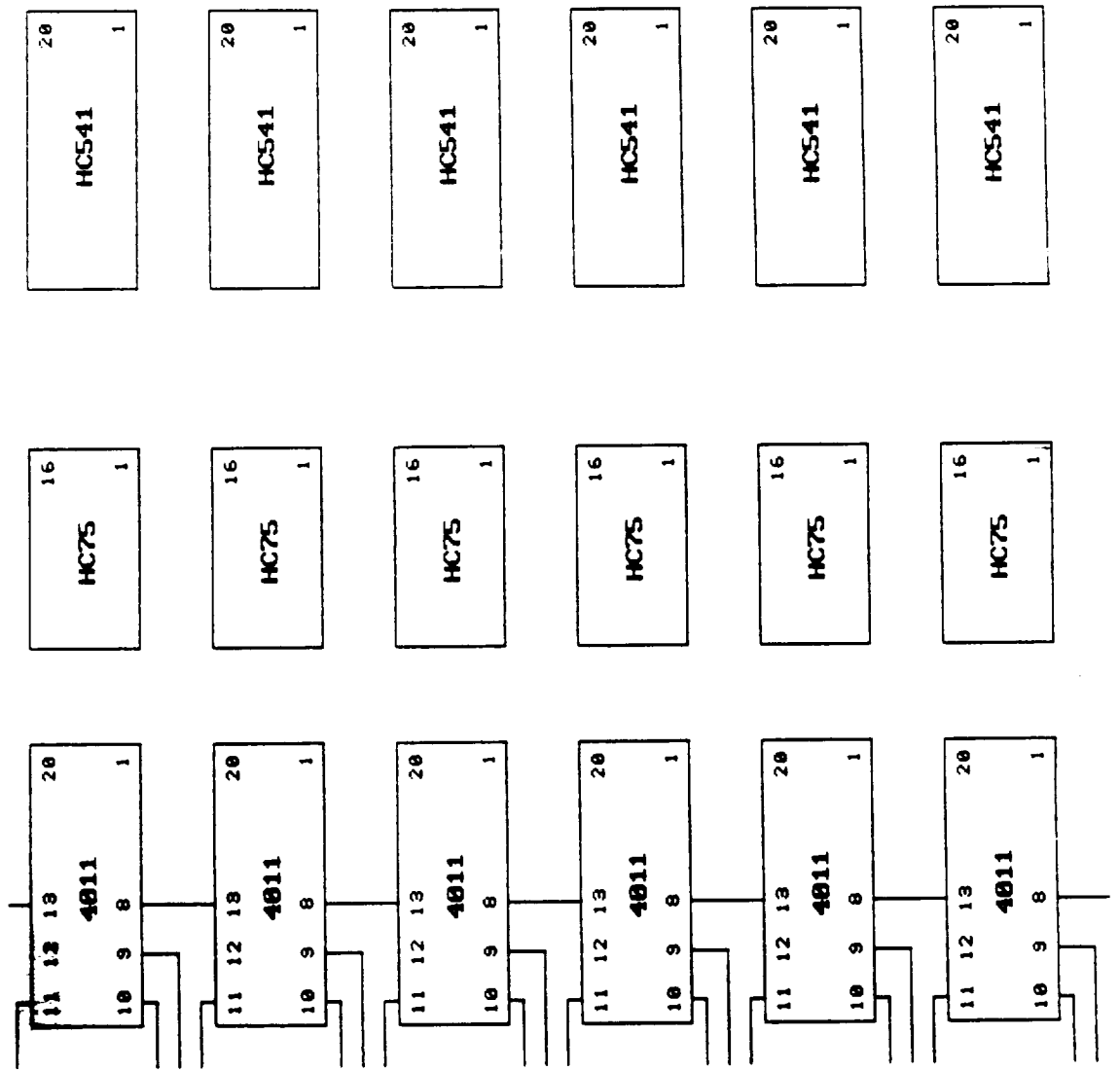


Figure 20

03

02

01

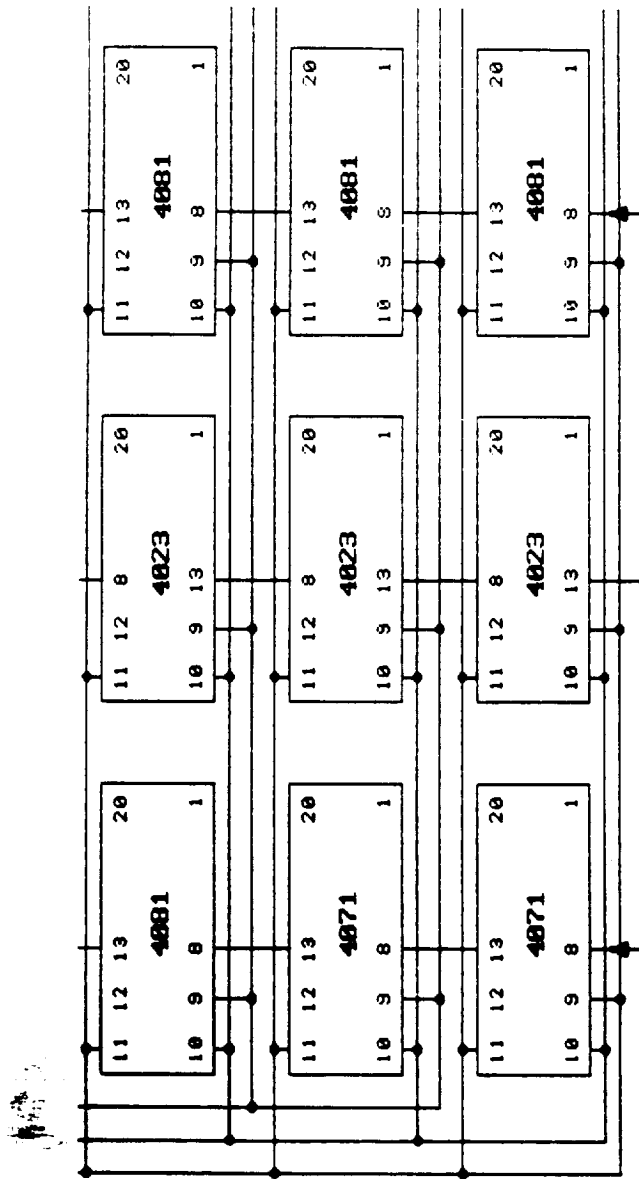
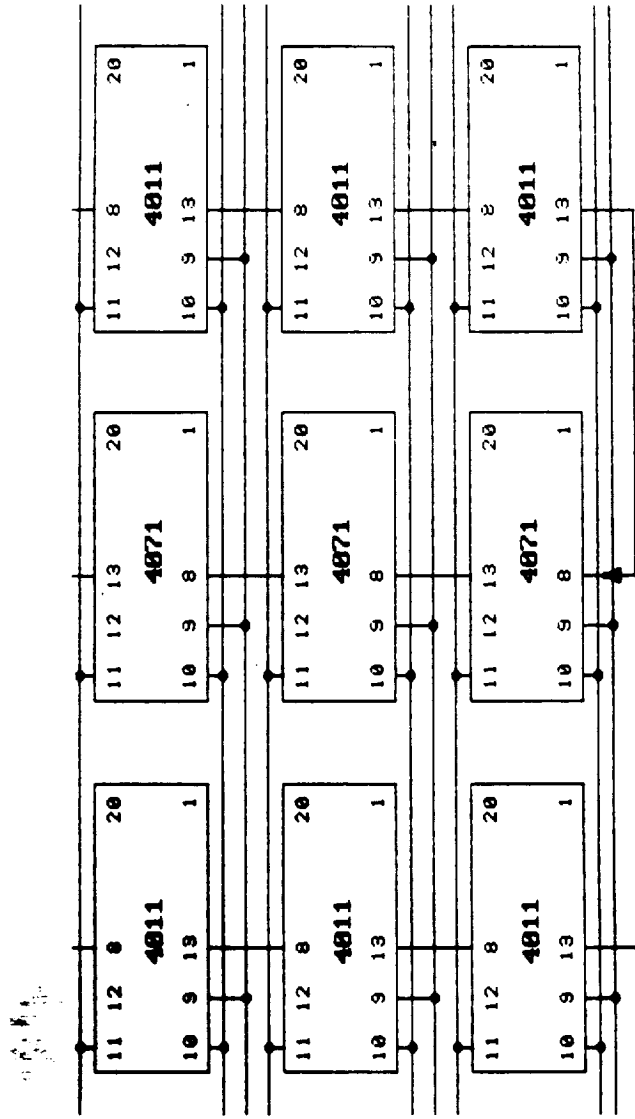


Figure 22



ERROR DETECTION AND CORRECTION UNIT WITH BUILT-IN SELF
TEST CAPABILITY FOR SPACECRAFT APPLICATIONS

NAS7-1028

PART 3

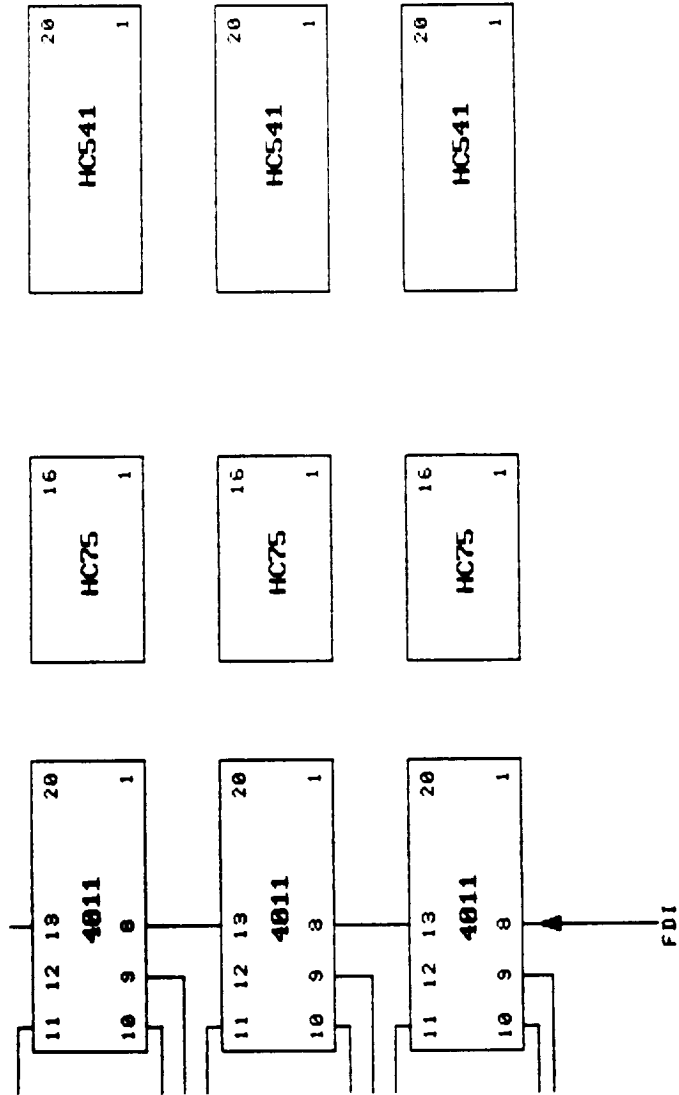


Figure 24

Quad 2-Input NAND Gate
Microsimulator
Part Number 4HFS11

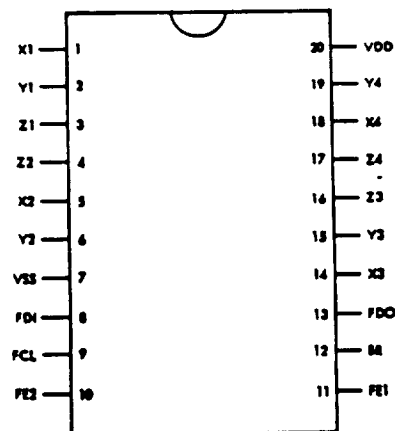
1 March 1981

<u>Pin Definitions</u>	
X1, Y1	Gate 1 inputs
Z1	Gate 1 output
X2, Y2	Gate 2 inputs
Z2	Gate 2 outputs
X3, Y3	Gate 3 inputs
Z3	Gate 3 output
X4, Y4	Gate 4 inputs
Z4	Gate 4 output
FE1, FE2	These fault-enable pins select one of the four possible fault-simulation modes as shown in Table 1.
FCL, FDI, FDO	The lock input (FCL) and data input (FDI) to the 12-bit shift register are used to shift in data on the rising edge of the clock (FCL). The output of the shift register appears at FDO.
BR	Bridging output.

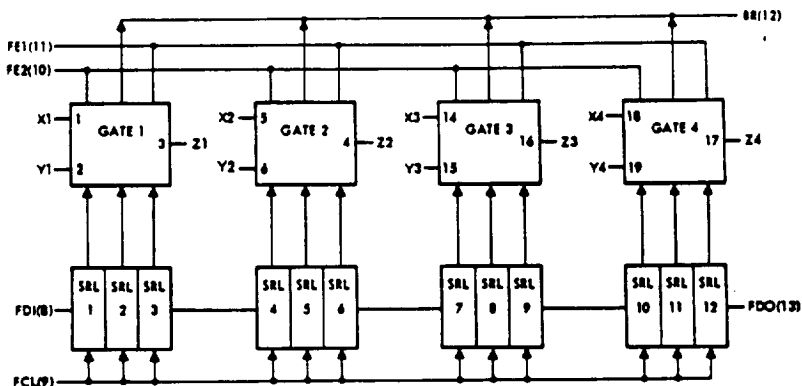
Each nand gate with inputs X1, Y1 and output Z1 is connected to three shift register latches (SRL's) from the 12-bit shift register as follows:

SRL1, SRL2, SRL3 are connected to Gate 1
SRL4, SRL5, SRL6 are connected to Gate 2
SRL7, SRL8, SRL9 are connected to Gate 3
SRL10, SRL11, SRL12 are connected to Gate 4

The content of the SRL's and the simulation mode determine which fault and what type of fault is being injected.



FUNCTIONAL BLOCK DIAGRAM



Description

This CMOS integrated circuit is used to perform fault simulation. It has four modes of operation which are determined by the logic values applied at inputs FE1 and FE2 (Table 1). In one mode of operation, called Good Machine (FE1 = FE2 = 0), the integrated circuit is functionally equivalent to the industry standard CMOS 4011 quad 2-input NAND at inputs Xi and Yi, and outputs Zi. Additionally, the logic value appearing at Zi can be made observable at BR if a "1" is shifted into SRL1 while the remaining SRL's are all "0". Similarly, Z2 or Z3 or Z4 are observable at BR if a "1" is shifted in SRL4 or SRL7 or SRL10, respectively. Note that only one output Zi can be made observable at BR at one time (Table 2).

Another mode of operation, called Stuck-at (FE1 = 0, FE2 = 1), enables the injection of stuck-at faults at the inputs and outputs Xi, Yi and Zi. If the SRL's are all "0", although the machine is in Stuck-at mode, no faults are being injected. By shifting a "1" into the first SRL of any one of the gates, a stuck-at-one on Xi is being injected. A "1" in the second SRL of a gate injects a stuck-at-one at Yi and a "1" in the third SRL of a gate injects a stuck-at-one at Zi. A stuck-at-one at an input of a gate forces the input to "1" independent on the logic value applied to that input. The BR output is floating in the Stuck-at mode (Table 3).

Yet another mode of operation, called Bridging (FE1 = 1, FE2 = 0) is used to simulate short-

circuit faults between the outputs of any two gates in the network. When all SRL's are "0" the chip is fault free. By shifting a "1" in the first SRL of a gate, the input of that gate is made observable at BR. If a "1" is shifted in the second SRL of a gate, then a stuck-at-zero at the output of that gate is injected and a "1" in the third SRL of a gate injects a stuck-at-one at the output (Table 4).

Still another mode of operation, called Stuck-open (FE1 = FE2 = 1), enables the injection of stuck-open faults. These types of faults disconnect the source or drain of the transistors performing the NAND function from VDD or VSS. Xi stuck-open disconnects the VDD from the p transistor whose gate is connected to Xi; this is being accomplished by shifting a "1" only in the first SRL of a gate. Shifting a "1" in the second SRL of a gate will inject Yi stuck-open, which means that the p transistor whose gate is connected to Yi is disconnected from VDD. Moreover, shifting a "1" in the third SRL of a gate, Zi stuck-open is injected; this fault disconnects both n-transistors from VSS (Table 5).

It is worth noting that a chip must always be powered up with FE1 = FE2 = 1 followed by a sequence that clears the shift register; at least 12 "0" must be shifted in. Moreover, when simulating multiple stuck-at or stuck-open faults, do not put the chip in either Good Machine or Bridging modes because more than one gate can be enabled simultaneously on the BR line, thus, increasing excessively the power dissipation of the chip.

Table 1 Modes of operation

FE1	FE2	Fault simulation mode
0	0	Good machine (fault free circuit)
0	1	Stuck-at
1	0	Bridging
1	1	Stuck-open

Table 2 Good machine mode (FE1 = 0, FE2 = 0)

SRL1	SRL2	SRL3	Fault injected
0	0	0	No fault injected
1	0	0	Gate output Z1 appears at pin RR
0	1	0	No fault injected
0	0	1	No fault injected

Table 3 Stuck-at mode (FE1 = 0, FE2 = 1)

SRL1	SRL2	SRL3	Stuck-at fault injected
0	0	0	No fault injected
1	0	0	Stuck-at-one on input X1
0	1	0	Stuck-at-one on input Y1
0	0	1	Stuck-at-one on output Z1

Table 4 Bridging mode (FE1 = 1, FE2 = 0)

SRL1	SRL2	SRL3	Bridging fault
0	0	0	No fault is injected
1	0	0	Gate output Z1 appears at pin RR
0	1	0	Output Z1 is stuck-at-zero
0	0	1	Output Z1 is stuck-at-one

Table 5 Stuck-open mode (FE1 = 1, FE2 = 1)

SRL1	SRL2	SRL3	Stuck open fault location
0	0	0	No fault is injected
1	0	0	X1 Stuck-open (p-transistor whose gate is connected to input X1 is disconnected from V_{DD})
0	1	0	Y1 Stuck-open (p-transistor whose gate is connected to input Y1 is disconnected from V_{DD})
0	0	1	Z Stuck-open (n-transistors are disconnected from V_{SS})

Table 6 Switching characteristics

Parameter name	Output Load	Typical Value
Propagation delay from X1 or Y1 to Z1	50 pF	400 nsec
Propagation delay from Z1 to RR	1000 pF	800 nsec
Max. clock at PCL	-	500 KHz
Input capacitance	-	5 pF

ORIGINAL PAGE IS
OF POOR QUALITY

Triple 3-Input NAND Gate
Microsimulator
Part Number 4HFS023

1 March 1981

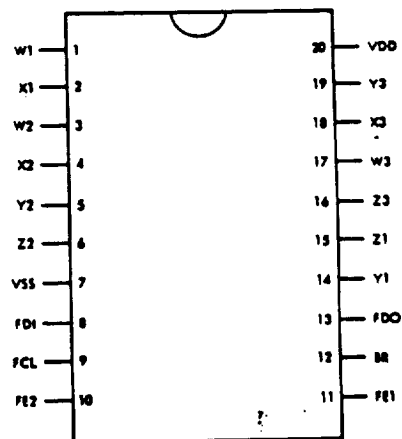
Pin Definitions

W1, X1, Y1	Gate 1 inputs
Z1	Gate 1 output
W2, X2, Y2	Gate 2 inputs
Z2	Gate 2 output
W3, X3, Y3	Gate 3 inputs
Z3	Gate 3 output
FE1, FE2	These fault-enable pins select one of the four possible fault-simulation modes as shown in Table 1.
FCL, FDI, FDO	The clock input (FCL) and data input (FDI) to the 12-bit shift register are used to shift in data on the rising edge of the clock (FCL). The output of the shift register appears at FDO.
BR	Bridging output

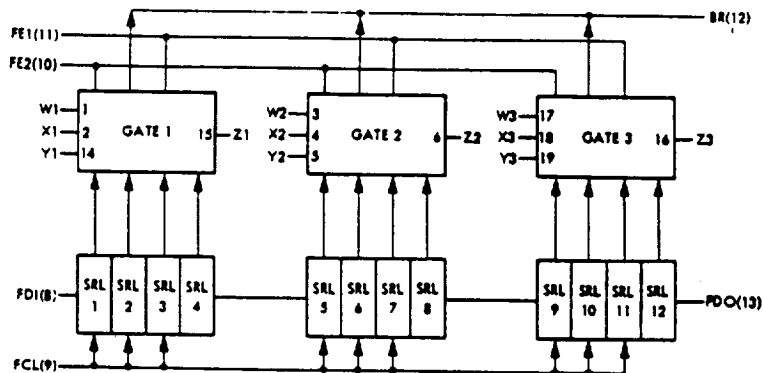
Each OR gate with inputs W_i, X_i, Y_i and output Z_i is connected to four shift register latches (SRL's) from the 12-bit shift register as follows:

SRL1, SRL2, SRL3, SRL4 are connected to Gate 1
SRL5, SRL6, SRL7, SRL8 are connected to Gate 2
SRL9, SRL10, SRL11, SRL12 are connected to Gate 3

The content of the SRL's and the fault simulation mode determine which fault and what type of fault is being injected.



FUNCTIONAL BLOCK DIAGRAM



Description

This CMOS integrated circuit is used to perform fault simulation. It has four modes of operation which are determined by the logic values applied at inputs FE1 and FE2 (Table 1). In one mode of operation, called Good Machine (FE1 = FE2 = 0), the integrated circuit is functionally equivalent to the industry standard CMOS 4023 triple 3-input NAND. Additionally, the logic value appearing at Z1 can be made observable at BR if a "1" is shifted into SRL1 while the remaining SRL's are all "0". Similarly, Z2 or Z3 observable at BR if a "1" is shifted in SRL5 or SRL9, respectively. Note that only one output can be made observable at BR at one time (Table 2).

Another mode of operation, called Stuck-at (FE1 = 0, FE2 = 1), enables the injection of stuck-at faults at the inputs and outputs W1, X1, Y1, and Z1. If the SRL's are all "0", although the machine is in Stuck-at mode, no faults are being injected. By shifting a "1" into the first SRL of any one of the gates, a stuck-at-one on W1 is being injected. A "1" in the second SRL of a gate injects a stuck-at-one at X1 and a "1" in the third SRL of a gate injects a stuck-at-one at Y1. Moreover, a "1" in the fourth SRL injects a stuck-at-one at Z1. A stuck-at-one at an input of a gate forces the input to "1" independent on the logic value applied to that input. The BR output is floating in the Stuck-at mode (Table 3).

Yet another mode of operation, called Bridging (FE1 = 1, FE2 = 0), is used to simulate short-

circuit faults between the outputs of any two gates in the network. When all SRL's are "0" the chip is fault free. By shifting a "1" in the first SRL of a gate, the input of that gate is made observable at BR. If a "1" is shifted in the second SRL of a gate, then a stuck-at-zero at the output of that gate is injected and a "1" in the third SRL of a gate injects a stuck-at-one at the output (Table 4).

Still another mode of operation, called Stuck-open (FE1 = FE2 = 1), enables the injection of stuck-open faults. These types of faults disconnect the source or drain of the transistors performing the OR function from VDD or VSS. For example, W1 stuck-open disconnects VDD from the p-transistor whose gate is connected to W1; this is being accomplished by shifting a "1" only in the first SRL of a gate. Moreover, shifting a "1" in the fourth SRL of a gate, Z1 stuck-open is injected; this fault disconnects all three n-transistors from VSS (Table 5).

It is worth noting that a chip must always be powered up with FE1 = FE2 = 1 followed by a sequence that clears the shift register; at least 12 "0" must be shifted in. Moreover, when simulating multiple stuck-at or stuck-open faults, do not put the chip in either Good Machine or Bridging modes because more than one gate can be enabled simultaneously on the BR line, thus, increasing excessively the power dissipation of the chip.

ORIGINAL PAGE IS
OF POOR QUALITY

Table 1 Modes of operation

FE1	FE2	Fault simulation mode
0	0	Good machine (fault free circuit)
0	1	Stuck-at
1	0	Bridging
1	1	Stuck-open

Table 3 Stuck-at mode (FE1 = 0, FE2 = 1)

SRL1	SRL2	SRL3	SRL4	Stuck-at fault injected
0	0	0	0	No fault injected
1	0	0	0	Stuck-at-one input W1
0	1	0	0	Stuck-at-one input X1
0	0	1	0	Stuck-at-one input Y1
0	0	0	1	Stuck-at-one at output Z1

Table 5 Stuck-open mode (FE1 = 1, FE2 = 1)

SRL1	SRL2	SRL3	SRL4	Stuck open fault location
0	0	0	0	No fault is injected
1	0	0	0	W1 Stuck-open (p-transistor whose gate is connected to input X1 is disconnected from V _{DD})
0	1	0	0	X1 Stuck-open (p-transistor whose gate is connected to input X1 is disconnected from V _{DD})
0	0	1	0	Y1 Stuck-open (p-transistor whose gate is connected to input Y1 is disconnected from V _{DD})
0	0	0	1	Z Stuck-open (n-transistors are disconnected from V _{SS})

Table 2 Good Machine mode (FE1 = 0, FE2 = 0)

SRL1	SRL2	SRL3	SRL4	Fault injected
0	0	0	0	No fault injected
1	0	0	0	Gate output Z1 appears at pin BR
0	1	0	0	No fault injected
0	0	1	0	No fault injected
0	0	0	1	No fault injected

Table 4 Bridging mode (FE1 = 1, FE2 = 0)

SRL1	SRL2	SRL3	SRL4	Bridging fault
0	0	0	0	No fault is injected
1	0	0	0	Gate output Z1 appears at pin BR
0	1	0	0	Output Z1 is stuck-at-zero
0	0	1	0	Output Z1 is stuck-at-one

Table 6 Switching characteristics

Parameter name	Output Load	Typical Value
Propagation delay from X1 or Y1 to Z1	50 pF	400 nsec
Propagation delay from Z1 to BR	1000 pF	800 nsec
Max. clock at FCL	-	500 kHz
Input capacitance	-	5 pF

ORIGINAL PAGE IS
OF POOR QUALITY

Quad 2-Input OR Gate
Microsimulator
Part Number 4HFS071

1 March 1981

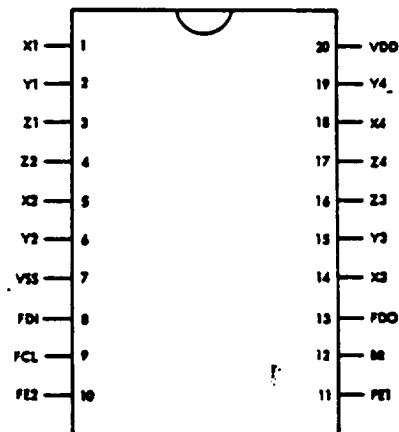
Pin Definitions

Z1, Y1	Gate 1 inputs
Z1	Gate 1 output
X2, Y2	Gate 2 inputs
Z2	Gate 2 output
X3, Y3	Gate 3 inputs
Z3	Gate 3 output
X4, Y4	Gate 4 inputs
Z4	Gate 4 output
FE1, FE2	These fault-enable pins select one of the four possible fault-simulation modes as shown in Table 1.
FCL, FDI, FDO	The clock input (FCL) and data input (FDI) to the 12-bit shift register are used to shift in data on the rising edge of the clock (FCL). The output of the shift register appears at FDO.
BR	Bridging output

Each OR gate with inputs X1, Y1 and output Z1 is connected to three shift register latches (SRL's) from the 12-bit shift register as follows:

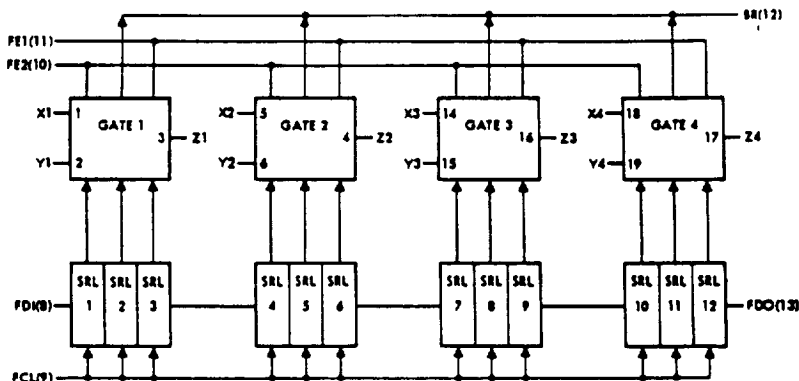
SRL1, SRL2, SRL3 are connected to Gate 1
SRL4, SRL5, SRL6 are connected to Gate 2
SRL7, SRL8, SRL9 are connected to Gate 3
SRL10, SRL11, SRL12 are connected to Gate 4

The content of the SRL's and the simulation mode determine which fault and what type of fault is being injected.



ORIGINAL PAGE IS
OF POOR QUALITY

FUNCTIONAL BLOCK DIAGRAM



Description

This CMOS integrated circuit is used to perform fault simulation. It has four modes of operation which are determined by the logic values applied at inputs FE1 and FE2 (Table 1). In one mode of operation, called Good Machine (FE1 = FE2 = 0), the integrated circuit is functionally equivalent to the industry standard CMOS 4071 quad 2-input OR. Additionally, the logic value appearing at Zi can be made observable at BR if a "1" is shifted into SRL1 while the remaining SRL's are all "0". Similarly, Z2 or Z3 or Z4 are observable at BR if a "1" is shifted in SRL4 or SRL7 or SRL10, respectively. Note that only one output Zi can be made observable at BR at one time (Table 2).

Another mode of operation, called Stuck-at (FE1 = 0, FE2 = 1), enables the injection of stuck-at faults at the inputs and outputs Xi, Yi and Zi. If the SRL's are all "0", although the machine is in Stuck-at mode, no faults are being injected. By shifting a "1" into the first SRL of any one of the gates, a stuck-at-zero on Xi is being injected. A "1" in the second SRL of a gate injects a stuck-at-zero at Yi and a "1" in the third SRL of a gate injects a stuck-at-one at Zi. A stuck-at-zero at an input of a gate forces the input to "0" independent of the logic value applied to that input. The BR output is floating in the Stuck-at mode (Table 3).

Yet another mode of operation, called Bridging (FE1 = 1, FE2 = 0) is used to simulate short-circuit faults between the outputs of any two

gates in the network. When all SRL's are "0" the chip is fault free. By shifting a "1" in the first SRL of a gate, the input of that gate is made observable at BR. If a "1" is shifted in the second SRL of a gate, then a stuck-at-zero at the output of that gate is injected and a "1" in the third SRL of a gate injects a stuck-at-one at the output (Table 4).

Still another mode of operation, called Stuck-open (FE1 = FE2 = 1), enables the injection of stuck-open faults. These types of faults disconnect the source or drain of the transistors performing the OR function from VDD or VSS. Xi stuck-open disconnects the VSS from the a transistor whose gate is connected to Xi; this is being accomplished by shifting a "1" only in the first SRL of a gate. Shifting a "1" in the second SRL of a gate will inject Yi stuck-open, which means that the n transistor whose gate is connected to Yi is disconnected from VSS. Moreover, shifting a "1" in the third SRL of a gate, Zi stuck-open is injected; this fault disconnects both p-transistors from VDD (Table 5).

It is worth noting that a chip must always be powered up with FE1 = FE2 = 1 followed by a sequence that clears the shift register; at least 12 "0" must be shifted in. Moreover, when simulating multiple stuck-at or stuck-open faults, do not put the chip in either Good Machine or Bridging modes because more than one gate can be enabled simultaneously on the BR line, thus, increasing excessively the power dissipation of the chip.

Table 1 Modes of operation

FE1	FE2	Fault simulation mode
0	0	Good machine (fault free circuit)
0	1	Stuck-at
1	0	Bridging
1	1	Stuck-open

Table 2 Good machine mode (FE1 = 0, FE2 = 0)

SRL1	SRL2	SRL3	Fault injected
0	0	0	No fault injected
1	0	0	Gate output Z1 appears at pin BR
0	1	0	No fault injected
0	0	1	No fault injected

Table 3 Stuck-at mode (FE1 = 0, FE2 = 1)

SRL1	SRL2	SRL3	Stuck-at fault injected
0	0	0	No fault injected
1	0	0	Stuck-at-zero on input X1
0	1	0	Stuck-at-zero on input Y1
0	0	1	Stuck-at-one on output Z1

Table 4 Bridging mode (FE1 = 1, FE2 = 0)

SRL1	SRL2	SRL3	Bridging fault
0	0	0	No fault is injected
1	0	0	Gate output Z1 appears at pin BR
0	1	0	Output Z1 is stuck-at-zero
0	0	1	Output Z1 is stuck-at-one

Table 5 Stuck-open mode (FE1 = 1, FE2 = 1)

SRL1	SRL2	SRL3	Stuck open fault location
0	0	0	No fault is injected
1	0	0	X1 Stuck-open (n-transistor whose gate is connected to input X1 is disconnected from V_{SS})
0	1	0	Y1 Stuck-open (n-transistor whose gate is connected to input Y1 is disconnected from V_{SS})
0	0	1	Z Stuck-open (p-transistors are disconnected from V_{DD})

Table 6 Switching characteristics

Parameter name	Output Load	Typical Value
Propagation delay from X1 or Y1 to Z1	50 pF	400 nsec
Propagation delay from Z1 to BR	1000 pF	800 nsec
Max. clock at FCL	-	500 KHz
Input capacitance	-	5 pF

Quad 2-Input AND Gate
Microsimulator
Part Number 4HFS081

1 March 1981

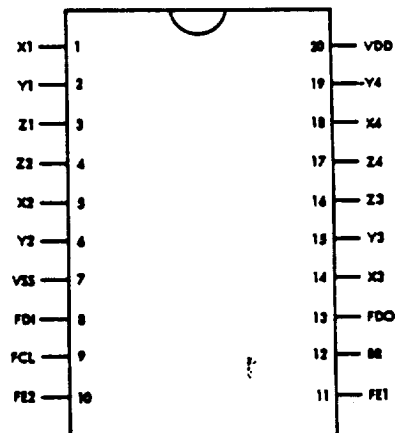
Pin Definitions

X1, Y1	Gate 1 inputs
Z1	Gate 1 output
X2, Y2	Gate 2 inputs
Z2	Gate 2 output
X3, Y3	Gate 3 inputs
Z3	Gate 3 output
X4, Y4	Gate 4 inputs
Z4	Gate 4 output
FE1, FE2	These fault-enable pins select one of the four possible fault-simulation modes as shown in Table 1.
FCL, FDI, FDO	The clock input (FCL) and data input (FDI) to the 12-bit shift register are used to shift in data on the rising edge of the clock (FCL). The output of the shift register appears at FDO.
BR	Bridging output

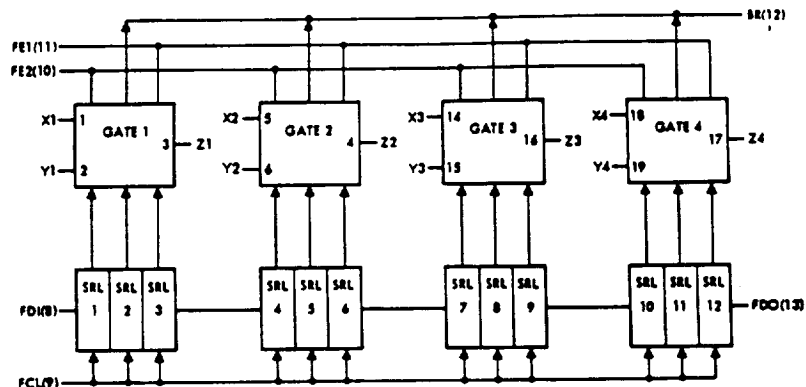
Each AND gate with inputs X1, Y1 and Output Z1 is connected to three shift register latches (SRL's) from the 12-bit shift register as follows:

SRL1, SRL2, SRL3 are connected to Gate 1
SRL4, SRL5, SRL6 are connected to Gate 2
SRL7, SRL8, SRL9 are connected to Gate 3
SRL10, SRL11, SRL 12 are connected to Gate 4

The content of the SRL's and the simulation mode determine which fault and what type of fault is being injected.



FUNCTIONAL BLOCK DIAGRAM



Description

This CMOS integrated circuit is used to perform fault simulation. It has four modes of operation which are determined by the logic values applied at inputs FE1 and FE2 (Table 1). In one mode of operation, called Good Machine ($FE1 = FE2 = 0$), the integrated circuit is functionally equivalent to the industry standard CMOS 4081 quad 2-input AND. Additionally, the logic value appearing at Z1 can be made observable at BR if a "1" is shifted into SRL1 while the remaining SRL's are all "0". Similarly, Z2 or Z3 or Z4 are observable at BR if a "1" is shifted in SRL4 or SRL7 or SRL10, respectively. Note that only one output Z1 can be made observable at BR at one time (Table 2).

Another mode of operation, called Stuck-at ($FE1 = 0, FE2 = 1$), enables the injection of stuck-at faults at the inputs and outputs X1, Y1 and Z1. If the SRL's are all "0", although the machine is in Stuck-at mode, no faults are being injected. By shifting a "1" into the first SRL of any one of the gates, a stuck-at-one on X1 is being injected. A "1" in the second SRL of a gate injects a stuck-at-one at Y1 and a "1" in the third SRL of a gate injects a stuck-at-zero at Z1. A stuck-at-one at an input of a gate forces the input to "1" independent of the logic value applied to that input. The BR output is floating in the Stuck-at mode (Table 3).

Yet another mode of operation, called Bridging ($FE1 = 1, FE2 = 0$) is used to simulate short-circuit faults between the outputs of any two

gates in the network. When all SRL's are "0" the chip is fault free. By shifting a "1" in the first SRL of a gate, the input of that gate is made observable at BR. If a "1" is shifted in the second SRL of a gate, then a stuck-at-one at the output of that gate is injected and a "1" in the third SRL of a gate injects a stuck-at-zero at the output. (Table 4).

Still another mode of operation, called Stuck-open ($FE1 = FE2 = 1$), enables the injection of stuck-open faults. These types of faults disconnect the source or drain of the transistors performing the AND function from VDD or VSS. X1 stuck-open disconnects the VDD from the p transistor whose gate is connected to X1; this is being accomplished by shifting a "1" only in the first SRL of a gate. Shifting a "1" in the second SRL of a gate will inject Y1 stuck-open, which means that the p transistor whose gate is connected to Y1 is disconnected from VDD. Moreover, shifting a "1" in the third SRL of a gate, Z1 stuck-open is injected; this fault disconnects both n-transistors from VSS (Table 5).

It is worth noting that a chip must always be powered up with $FE1 = FE2 = 1$ followed by a sequence that clears the shift register; at least 12 "0" must be shifted in. Moreover, when simulating multiple stuck-at or stuck-open faults, do not put the chip in either Good Machine or Bridging modes because more than one gate can be enabled simultaneously on the BR line, thus, increasing excessively the power dissipation of the chip.

Table 1 Modes of operation

FE1	FE2	Fault simulation mode
0	0	Good machine (fault free circuit)
0	1	Stuck-at
1	0	Bridging
1	1	Stuck-open

Table 2 Good machine mode (FE1 = 0, FE2 = 0)

SRL1	SRL2	SRL3	Fault injected
0	0	0	No fault injected
1	0	0	Gate output Z1 appears at pin BR
0	1	0	No fault injected
0	0	1	No fault injected

Table 3 Stuck-at mode (FE1 = 0, FE2 = 1)

SRL1	SRL2	SRL3	Stuck-at fault injected
0	0	0	No fault injected
1	0	0	Stuck-at-one on input X1
0	1	0	Stuck-at-one on input Y1
0	0	1	Stuck-at-zero on output Z1

Table 4 Bridging mode (FE1 = 1, FE2 = 0)

SRL1	SRL2	SRL3	Bridging fault
0	0	0	No fault is injected
1	0	0	Gate output Z1 appears at pin BR
0	1	0	Output Z1 is stuck-at-one
0	0	1	Output Z1 is stuck-at-zero

Table 5 Stuck-open mode (FE1 = 1, FE2 = 1)

SRL1	SRL2	SRL3	Stuck open fault location
0	0	0	No fault is injected
1	0	0	X1 Stuck-open (p-transistor whose gate is connected to input X1 is disconnected from V_{DD})
0	1	0	Y1 Stuck-open (p-transistor whose gate is connected to input Y1 is disconnected from V_{DD})
0	0	1	Z Stuck-open (n-transistors are disconnected from V_{SS})

Table 6 Switching characteristics

Parameter name	Output Load	Typical Value
Propagation delay from X1 or Y1 to Z1	50 pF	400 nsec
Propagation delay from Z1 to BR	1000 pF	800 nsec
Max. clock at FCL	-	500 KHz
Input capacitance	-	5 pF

EDAC Fault Simulation (EDAC combinational logic)

ICs used: 0105 - 3205 Exclusiv OR: OR gates
 0106 - 3206 Exclusiv OR: NAND gates
 0107 - 3207 Exclusiv OR: AND gates
 0108 - 1508 Error Corrector: NAND gates
 0109 - 1109 Error Detector, Exclusiv OR: mixed

 122 ICs = 1464 faults

after inserting the microsimulator chips connect following pins:

0103 / 8 to 0105 / 8
3207 / 13 to 1508 / 8

faults not used: 19

IC:	1208 / 17	fault number:	1197
	1208 / 18		1198
	1208 / 19		1199
	1208 / 16		1200
	0209 / 18		1354
	0209 / 19		1355
	0209 / 17		1356
	0409 / 14		1375
	0409 / 15		1376
	0409 / 16		1377
	0409 / 18		1378
	0409 / 19		1379
	0409 / 17		1380
	0709 / 5		1408
	0709 / 6		1409
	0709 / 4		1410
	0709 / 18		1414
	0709 / 19		1415
	0709 / 17		1416

undetectable faults: 0

EDAC Fault Simulation (scannable latches)

ICs used: 0103 - 3203 scanable latches, top row: NAND gates
0104 - 3204 scanable latches, bottom row: NAND gates
1608 - 3208 scanable latches, checkbit and syndrom

81 ICs = 972 faults

after inserting the microsimulator chips connect following pins:

0104 / 13 to 3208 / 8
1608 / 13 to 1109 / 13

faults not used: 0

undetectable faults: 3

IC:	3203 / 14	fault number:	379
	3203 / 16		381
	3203 / 19		383

EDAC Fault Simulation (EDAC logic)

Stuck-at

initial seed (hex): ff ff ff ff aa
good machine signature (hex): b6 84 66 a3 01 32
cycles completed: 10

total number of faults:	1464
number of faults not used:	13
number of undetectable faults:	7
number of detectable faults:	1444
number of faults not detected:	154
number of faults detected:	1290
fault coverage:	89.3%

Spaceborne, Inc.

Date: April 7, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-at

initial seed (hex): ff ff ff ff aa
good machine signature (hex): 59 06 cf 09 03 5a
cycles completed: 20

total number of faults:	1464
number of faults not used:	13
number of undetectable faults:	7
number of detectable faults:	1444
number of faults not detected:	70
number of faults detected:	1374
fault coverage:	95.2%

Spaceborne, Inc.

Date: April 7, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-at

initial seed (hex): ff ff ff ff aa
good machine signature (hex): b6 e4 48 85 03 4e
cycles completed: 100

total number of faults:	1464
number of faults not used:	13
number of undetectable faults:	7
number of detectable faults:	1444
number of faults not detected:	20
number of faults detected:	1424
fault coverage:	98.6%

Spaceborne, Inc.

Date: April 7, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-at

initial seed (hex): ff ff ff ff aa
good machine signature (hex): 46 26 94 1c 00 1b

cycles completed: 300

total number of faults: 1464
number of faults not used: 13
number of undetectable faults: 7
number of detectable faults: 1444
number of faults not detected: 2
number of faults detected: 1442
fault coverage: 99.9%

Spaceborne, Inc.

Date: April 7, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-at

initial seed (hex): ff ff ff ff aa
good machine signature (hex): 39 28 00 d1 03 5b

cycles completed: 400

total number of faults:	1464
number of faults not used:	13
number of undetectable faults:	7
number of detectable faults:	1444
number of faults not detected:	0
number of faults detected:	1444
fault coverage:	100.0%

Spaceborne, Inc.

Date: April 7, 1989

EDAC Fault Simulation (scannable latches)

Stuck-at

initial seed (hex): ff ff ff ff ff
good machine signature (hex): 06 6d ce d9 03 72
cycles completed: 2

total number of faults:	972
number of faults not used:	0
number of undetectable faults:	3
number of detectable faults:	969
number of faults not detected:	29
number of faults detected:	940
fault coverage:	97.0%

Spaceborne, Inc.

Date: April 12, 1989

EDAC Fault Simulation (scannable latches)

Stuck-at

initial seed (hex): ff ff ff ff ff
good machine signature (hex): 3b d7 e3 3c 03 0a
cycles completed: 4

total number of faults:	972
number of faults not used:	0
number of undetectable faults:	3
number of detectable faults:	969
number of faults not detected:	26
number of faults detected:	943
fault coverage:	97.3%

Spaceborne, Inc.

Date: April 12, 1989

EDAC Fault Simulation (scannable latches)

Stuck-at

initial seed (hex): ff ff ff ff ff
good machine signature (hex): 06 9e 48 52 00 07
cycles completed: 6

total number of faults:	972
number of faults not used:	0
number of undetectable faults:	3
number of detectable faults:	969
number of faults not detected:	0
number of faults detected:	969
fault coverage:	100.0%

Spaceborne, Inc.

Date: April 12, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-open

initial seed (hex): ff ff ff ff aa
good machine signature (hex): b6 84 66 a3 01 32
cycles completed: 10

total number of faults:	1464
number of faults not used:	19
number of undetectable faults:	0
number of detectable faults:	1445
number of faults not detected:	373
number of faults detected:	1072
fault coverage:	74.2%

Spaceborne, Inc.

Date: September 7, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-open

initial seed (hex): ff ff ff ff aa
good machine signature (hex): b6 e4 48 85 03 4e
cycles completed: 100

total number of faults:	1464
number of faults not used:	19
number of undetectable faults:	0
number of detectable faults:	1445
number of faults not detected:	100
number of faults detected:	1345
fault coverage:	93.1%

Spaceborne, Inc.

Date: September 7, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-open

initial seed (hex): ff ff ff ff aa
good machine signature (hex): 39 28 00 d1 03 5b
cycles completed: 400

total number of faults:	1464
number of faults not used:	19
number of undetectable faults:	0
number of detectable faults:	1445
number of faults not detected:	75
number of faults detected:	1370
fault coverage:	94.8%

Spaceborne, Inc.

Date: September 7, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-open

initial seed (hex): ff ff ff ff aa
good machine signature (hex): 87 5f 83 6b 03 5a
cycles completed: 1000

total number of faults:	1464
number of faults not used:	19
number of undetectable faults:	0
number of detectable faults:	1445
number of faults not detected:	69
number of faults detected:	1376
fault coverage:	95.2%

Spaceborne, Inc.

Date: September 7, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-open

initial seed (hex): ff ff ff ff aa
good machine signature (hex): 3e 84 ec 85 00 44
cycles completed: 10000

total number of faults: 1464
number of faults not used: 19
number of undetectable faults: 0
number of detectable faults: 1445
number of faults not detected: 40
number of faults detected: 1405
fault coverage: 97.2%

Spaceborne, Inc.

Date: September 7, 1989

EDAC Fault Simulation (EDAC logic)

Stuck-open

initial seed (hex): ff ff ff ff aa
good machine signature (hex): 3c 2d 1e 50 00 11
cycles completed: 100000

total number of faults: 1464
number of faults not used: 19
number of undetectable faults: 0
number of detectable faults: 1445
number of faults not detected: 34
number of faults detected: 1411
fault coverage: 97.6

Spaceborne, Inc.

Date: September 7, 1989

APPENDIX E: SILICON BREADBOARD

E.1 Logic Diagram of Silicon Breadboard

E.1.1 Map of Fault Simulator Breadboard.....E-1

E.1.2 Figure 1.....E-2

E.1.3 Figure 2.....E-3

E.1.4 Figure 3.....E-4

E.1.5 Figure 4.....E-5

E.1.6 Figure 5.....E-6

E.1.7 Figure 6.....E-7

E.1.8 Figure 7.....E-8

E.1.9 Figure 8.....E-9

E.1.10 Figure 9.....E-10

E.1.11 Figure 10.....E-11

E.1.12 Figure 11.....E-12

E.1.13 Figure 12.....E-13

E.1.14 Figure 13.....E-14

E.1.15 Figure 14.....E-15

E.1.16 Figure 15.....E-16

E.1.17 Figure 16.....E-17

E.1.18 Figure 17.....E-18

E.1.19 Figure 18.....E-19

E.1.20 Figure 19.....E-20

E.1.21 Figure 20.....E-21

E.1.22 Figure 21.....E-22

E.1.23 Figure 22.....E-23

E.1.24 Figure 23.....E-24

E.1.25 Figure 24.....E-25

E.1.26	Figure 25.....	E-26
E.1.27	Figure 26.....	E-27
E.1.28	Figure 27.....	E-28
E.1.29	Figure 28.....	E-29
E.1.30	Figure 29.....	E-30
E.1.31	Figure 30.....	E-31
E.1.32	Figure 31.....	E-32
E.1.33	Figure 32.....	E-33
E.1.34	Figure 33.....	E-34
E.1.35	Figure 34.....	E-35
E.1.36	Figure 35.....	E-36
E.1.37	Figure 36.....	E-37
E.1.38	Figure 37.....	E-38
E.1.39	Figure 38.....	E-39
E.1.40	Figure 39.....	E-40
E.1.41	Figure 40.....	E-41
E.1.42	Figure 41.....	E-42
E.1.43	Figure 42.....	E-43
E.1.44	Figure 43.....	E-44
E.1.45	Figure 44.....	E-45
E.1.46	Figure 45.....	E-46
E.1.47	Figure 46.....	E-47
E.1.48	Figure 47.....	E-48
E.1.49	Figure 48.....	E-49
E.1.50	Figure 49.....	E-50
E.1.51	Figure 50.....	E-51
E.1.52	Figure 51.....	E-52

E.1.53	Figure 52.....	E-53
E.1.54	Figure 53.....	E-54
E.1.55	Figure 54.....	E-55
E.1.56	Figure 55.....	E-56
E.1.57	Figure 56.....	E-57
E.1.58	Figure 57.....	E-58
E.1.59	Figure 58.....	E-59
E.1.60	Figure 59.....	E-60
E.1.61	Figure 60.....	E-61
E.2	Pin Assignment.....	E-62
E.3	Pin Description.....	E-63
E.4	Propagation Delays.....	E-65



Map of EDAC Logic Diagram

Fig. 1	Fig. 2	Fig. 3	Fig. 4	Fig. 5	Fig. 6	Fig. 7	Fig. 8	Fig. 9	Fig. 10	Fig. 11	Fig. 12
Fig. 13	Fig. 14	Fig. 15	Fig. 16	Fig. 17	Fig. 18	Fig. 19	Fig. 20	Fig. 21	Fig. 22	Fig. 23	Fig. 24
Fig. 25	Fig. 26	Fig. 27	Fig. 28	Fig. 29	Fig. 30	Fig. 31	Fig. 32	Fig. 33	Fig. 34	Fig. 35	Fig. 36
Fig. 37	Fig. 38	Fig. 39	Fig. 40	Fig. 41	Fig. 42	Fig. 43	Fig. 44	Fig. 45	Fig. 46	Fig. 47	Fig. 48
Fig. 49	Fig. 50	Fig. 51	Fig. 52	Fig. 53	Fig. 54	Fig. 55	Fig. 56	Fig. 57	Fig. 58	Fig. 59	Fig. 60

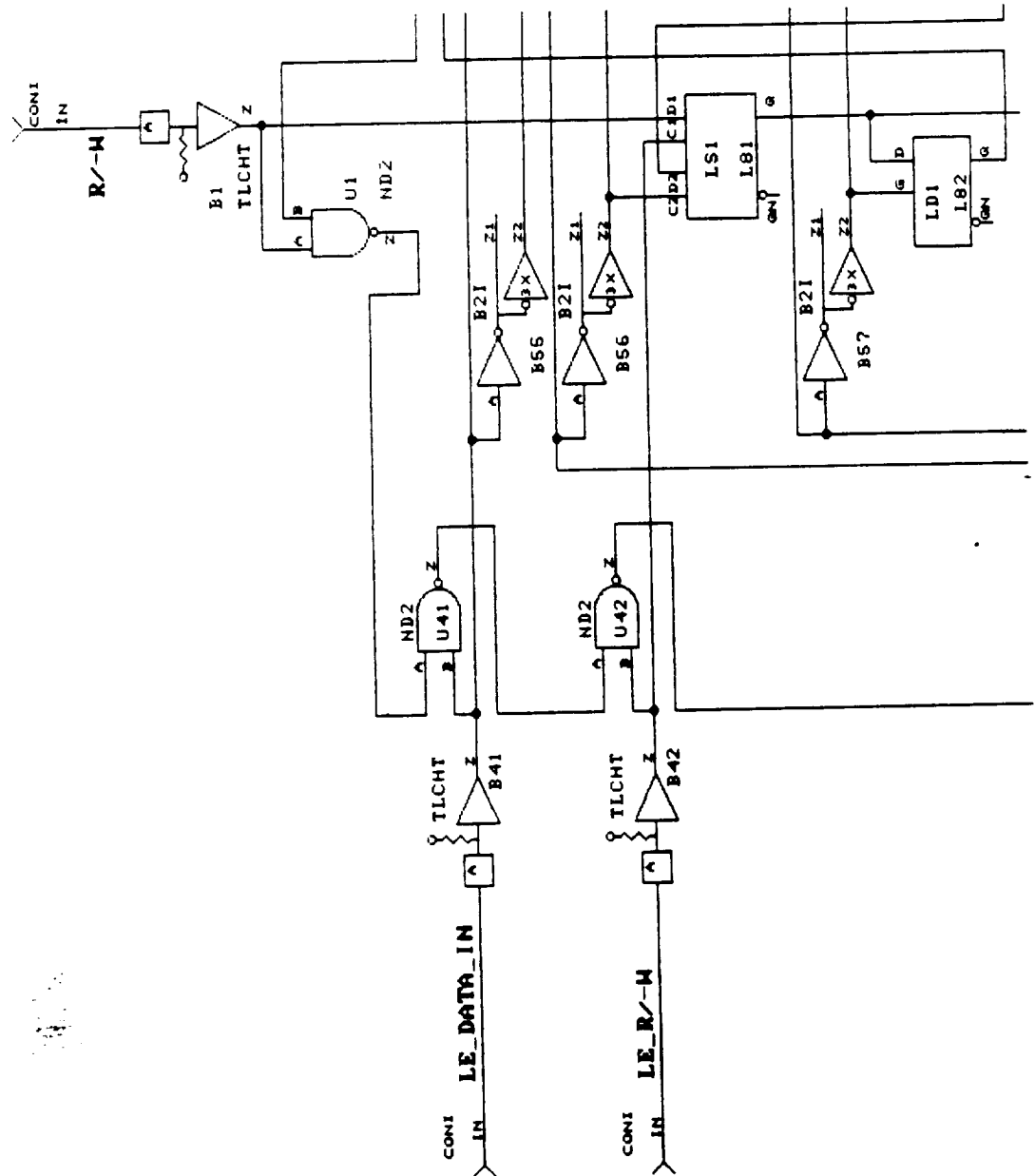


Figure 1

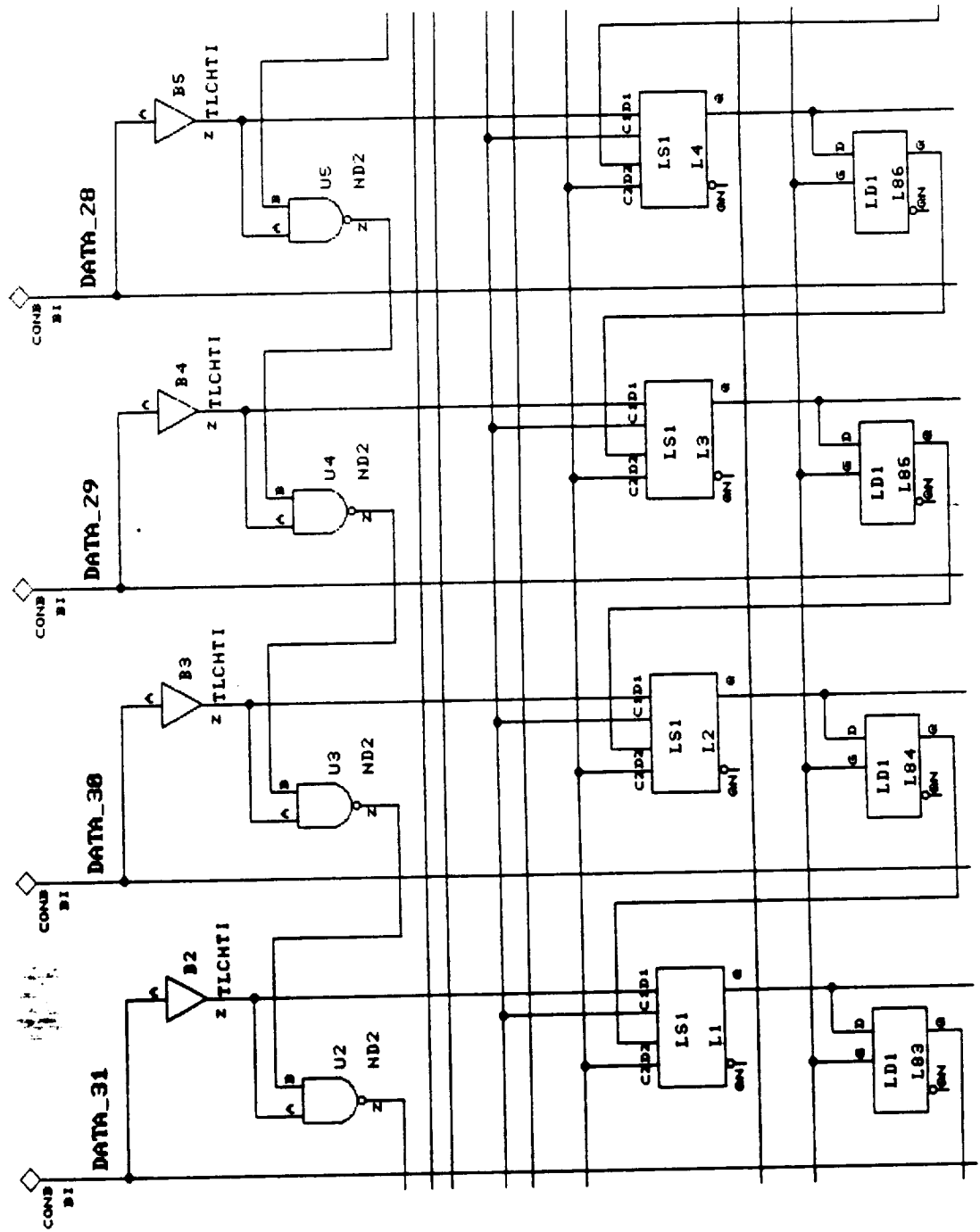


Figure 2

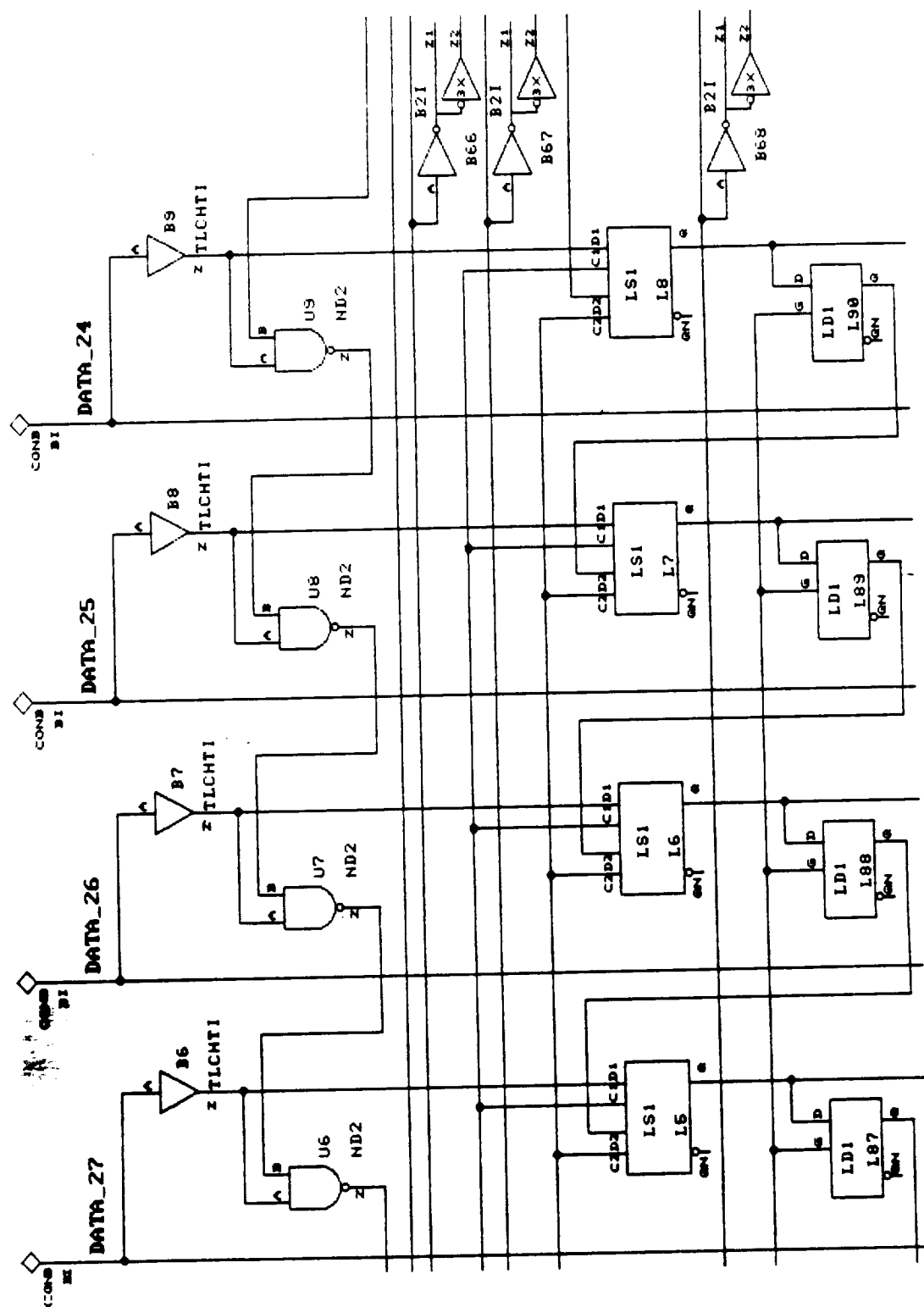


Figure 3

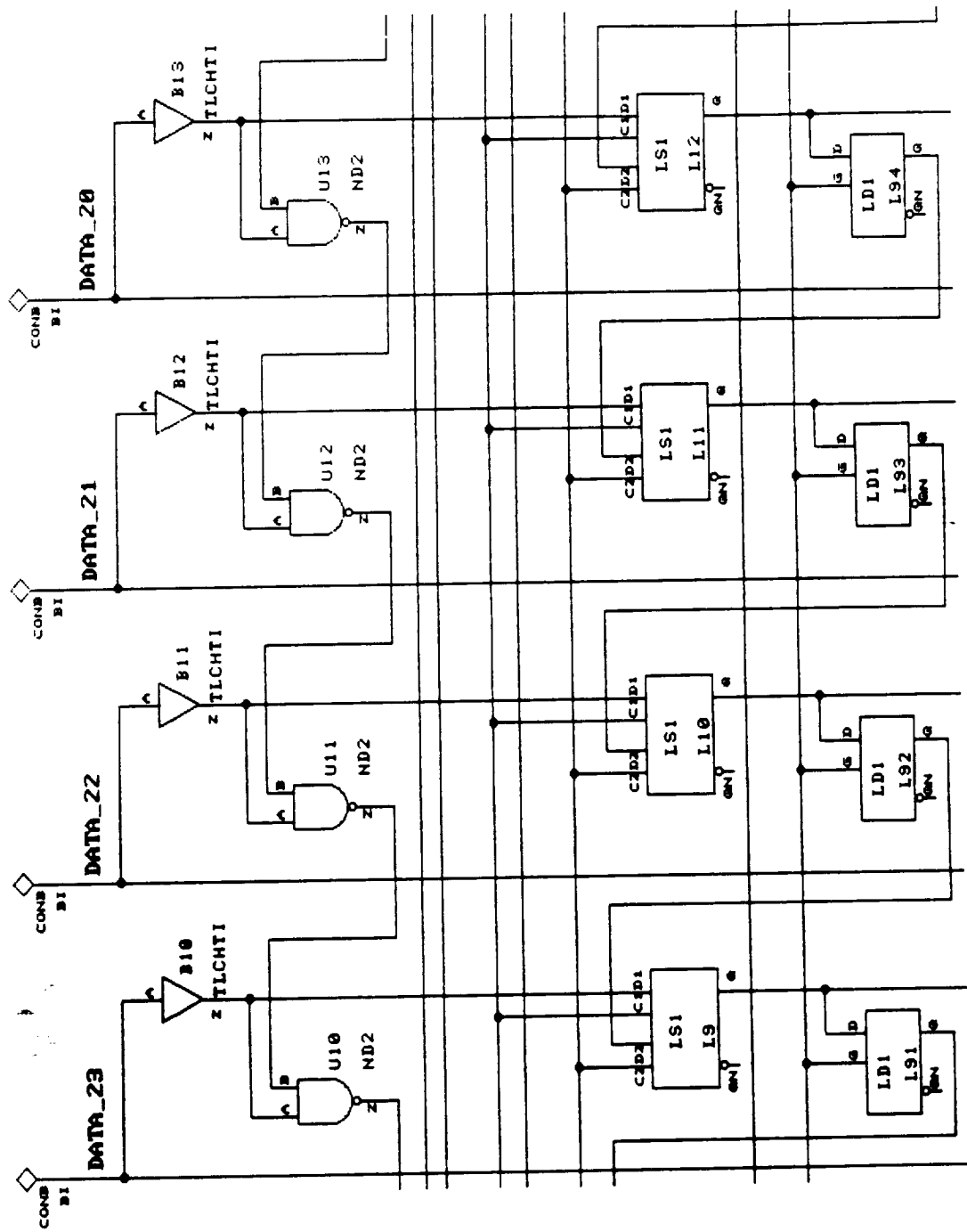


Figure 4

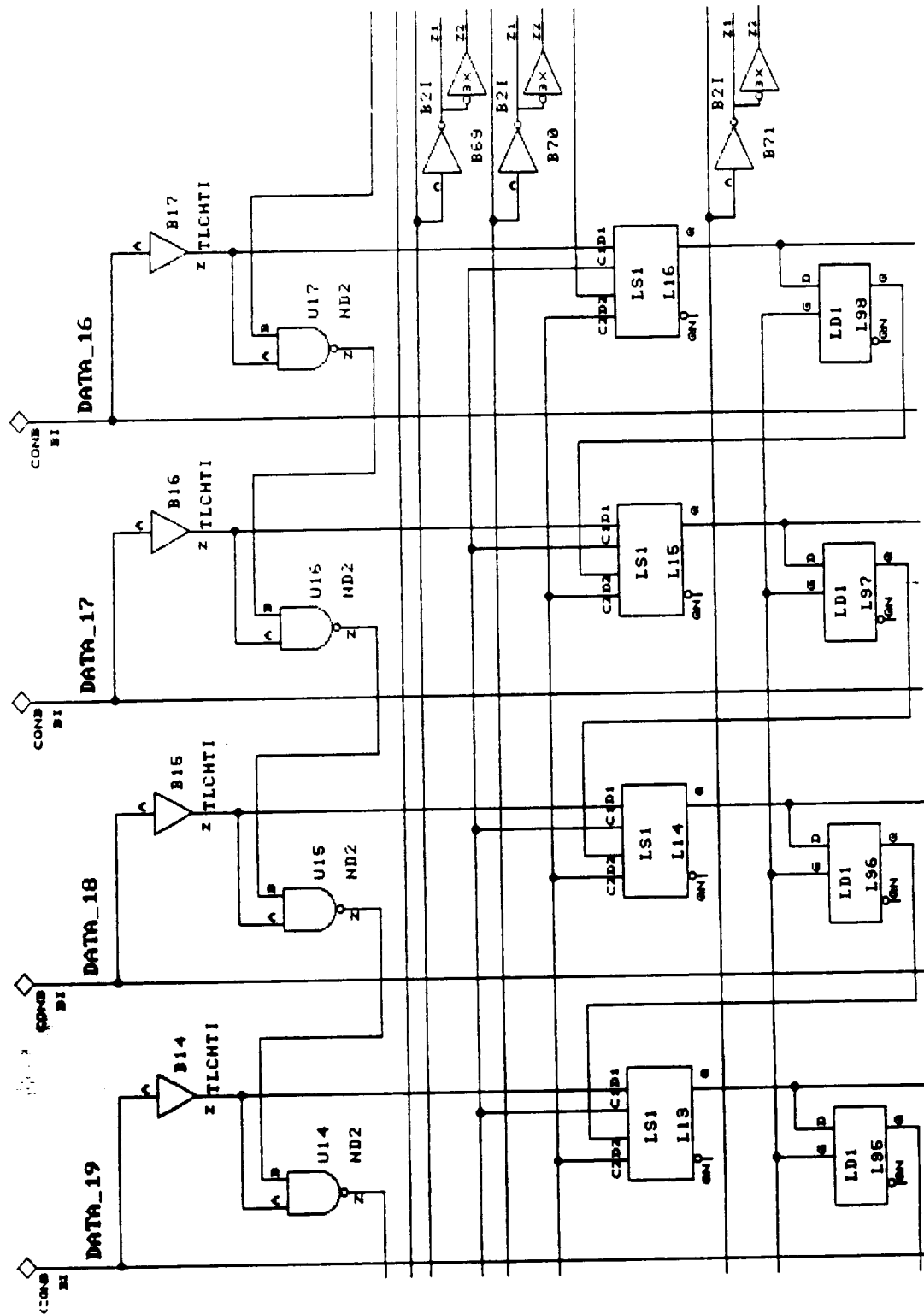


Figure 5

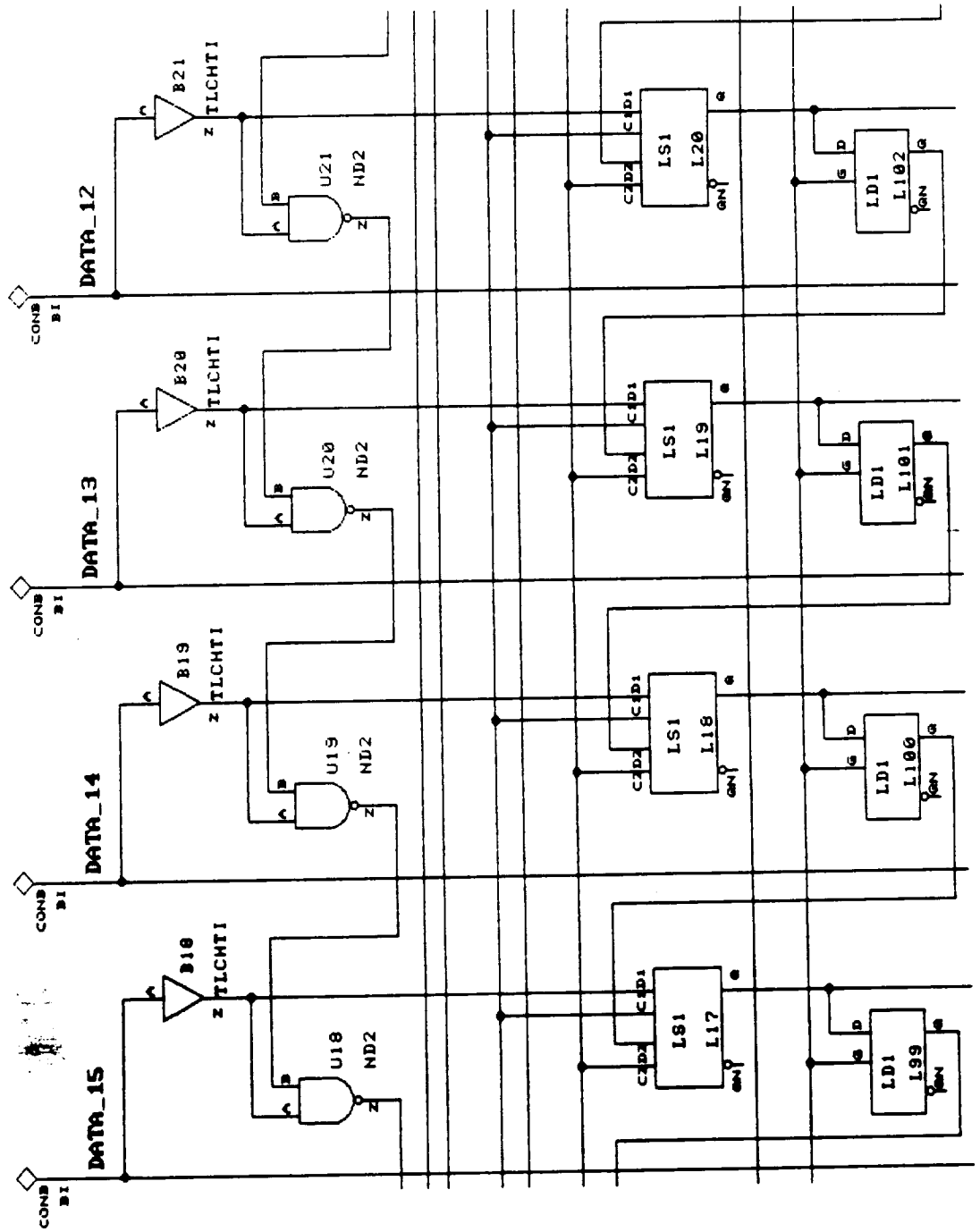


Figure 6

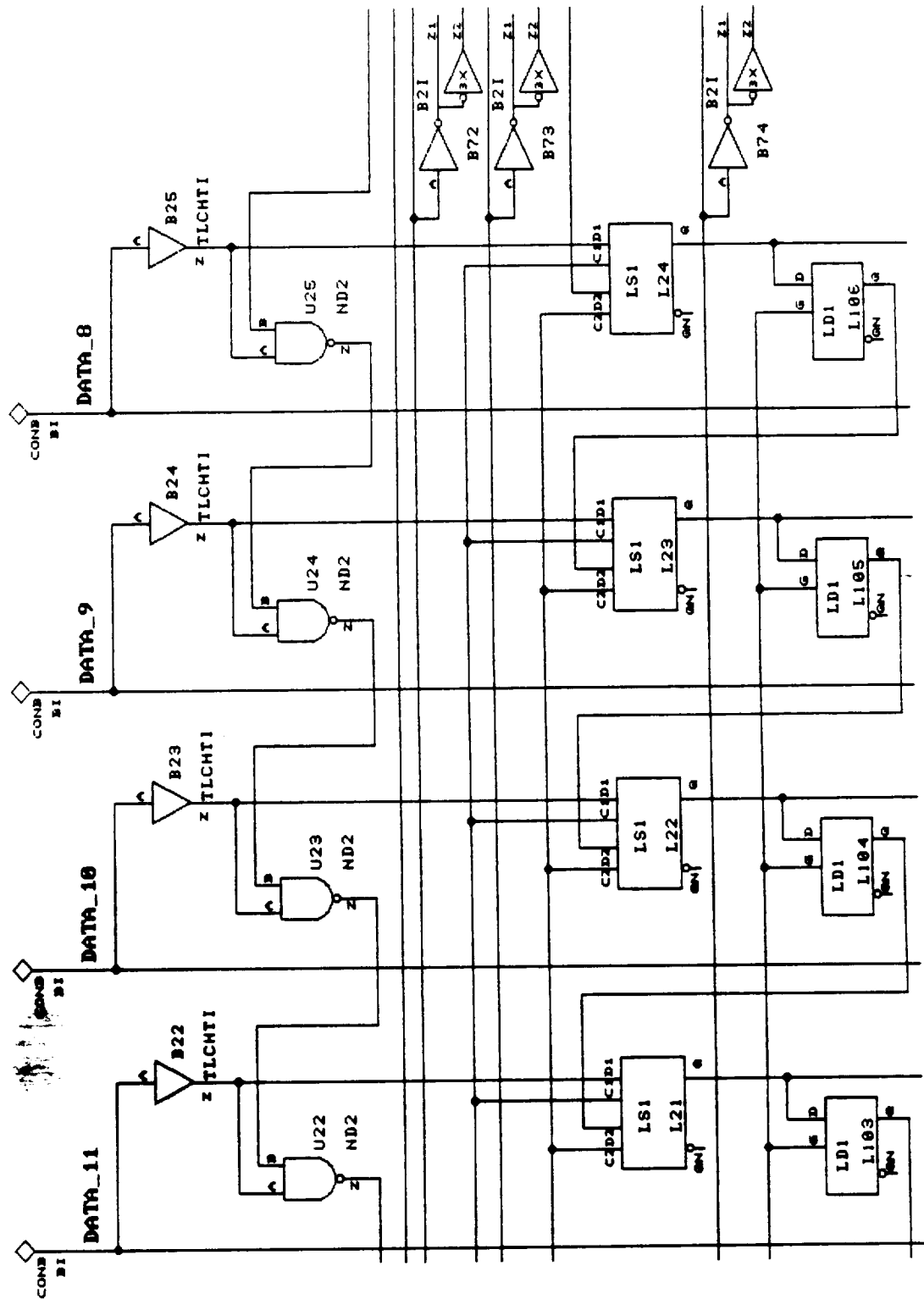


Figure 7

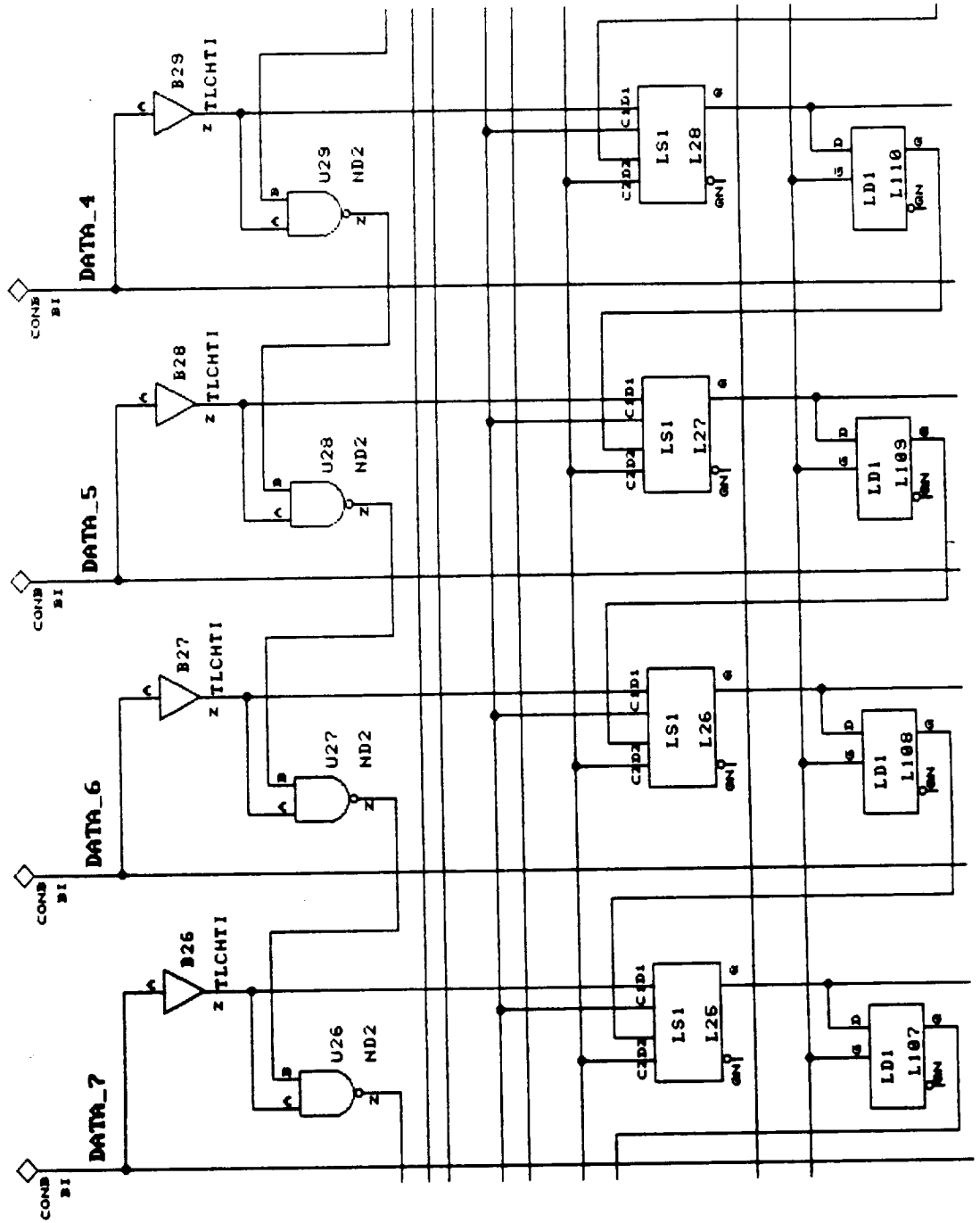


Figure 8

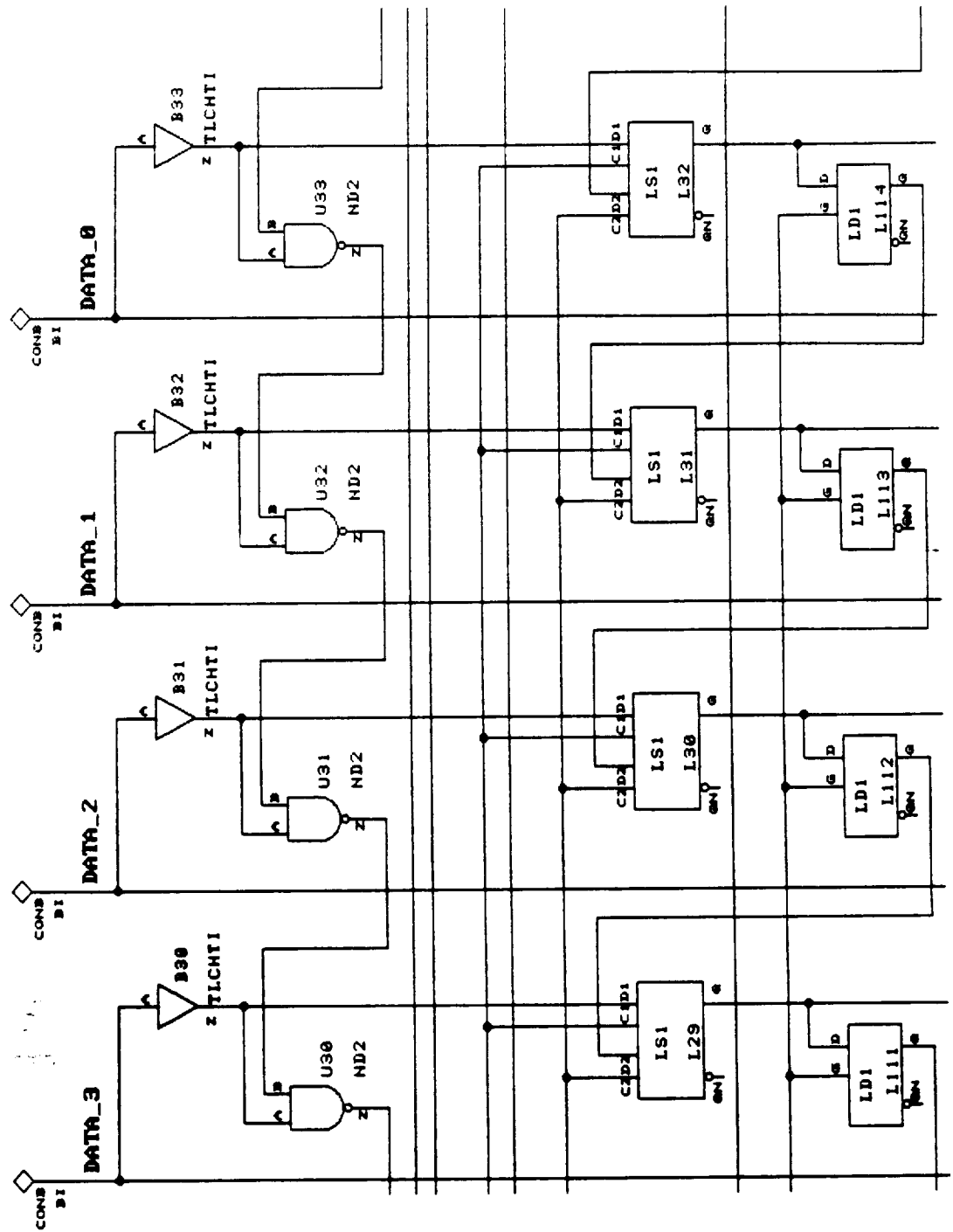


Figure 9

Part 10

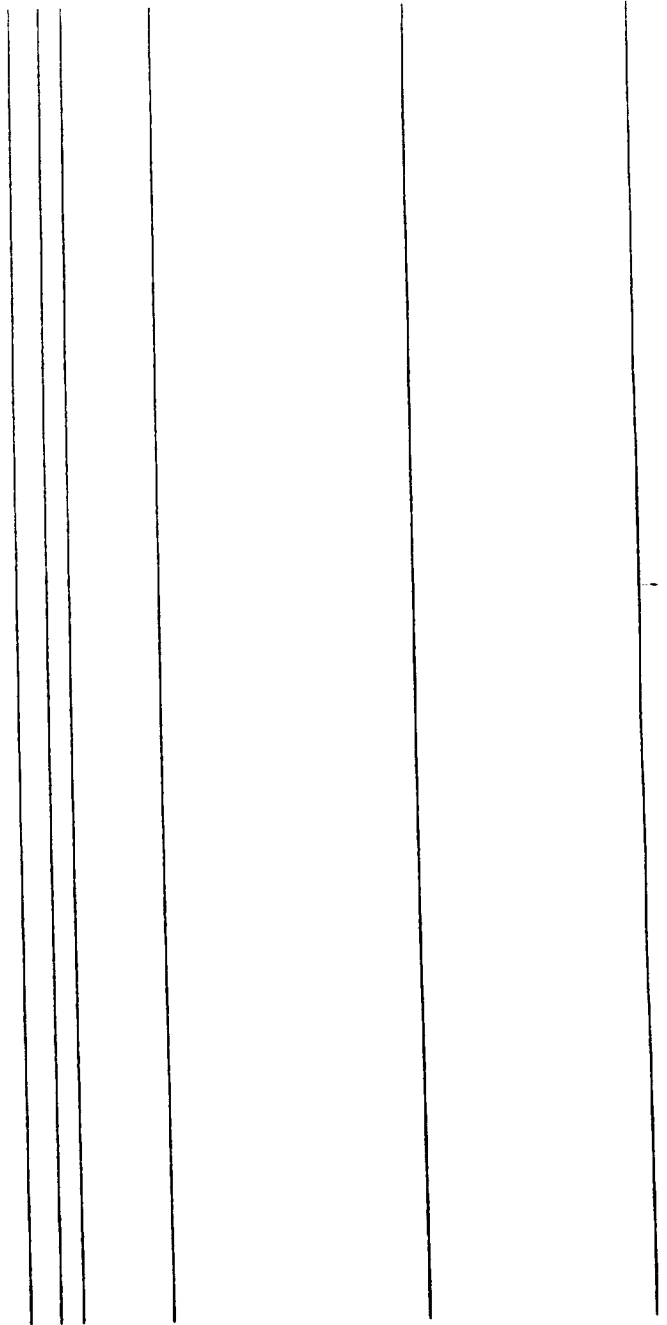


Figure 10

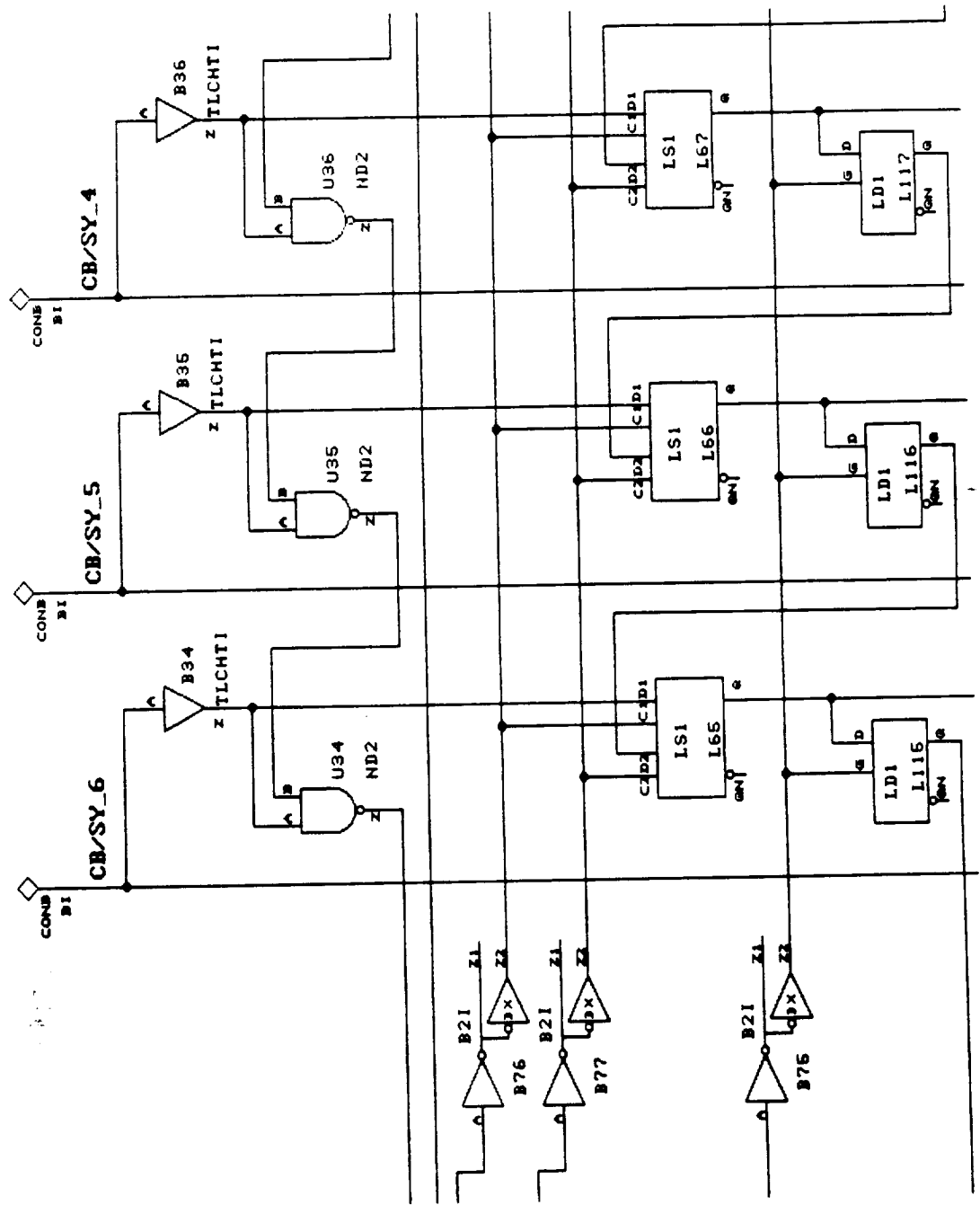


Figure 11

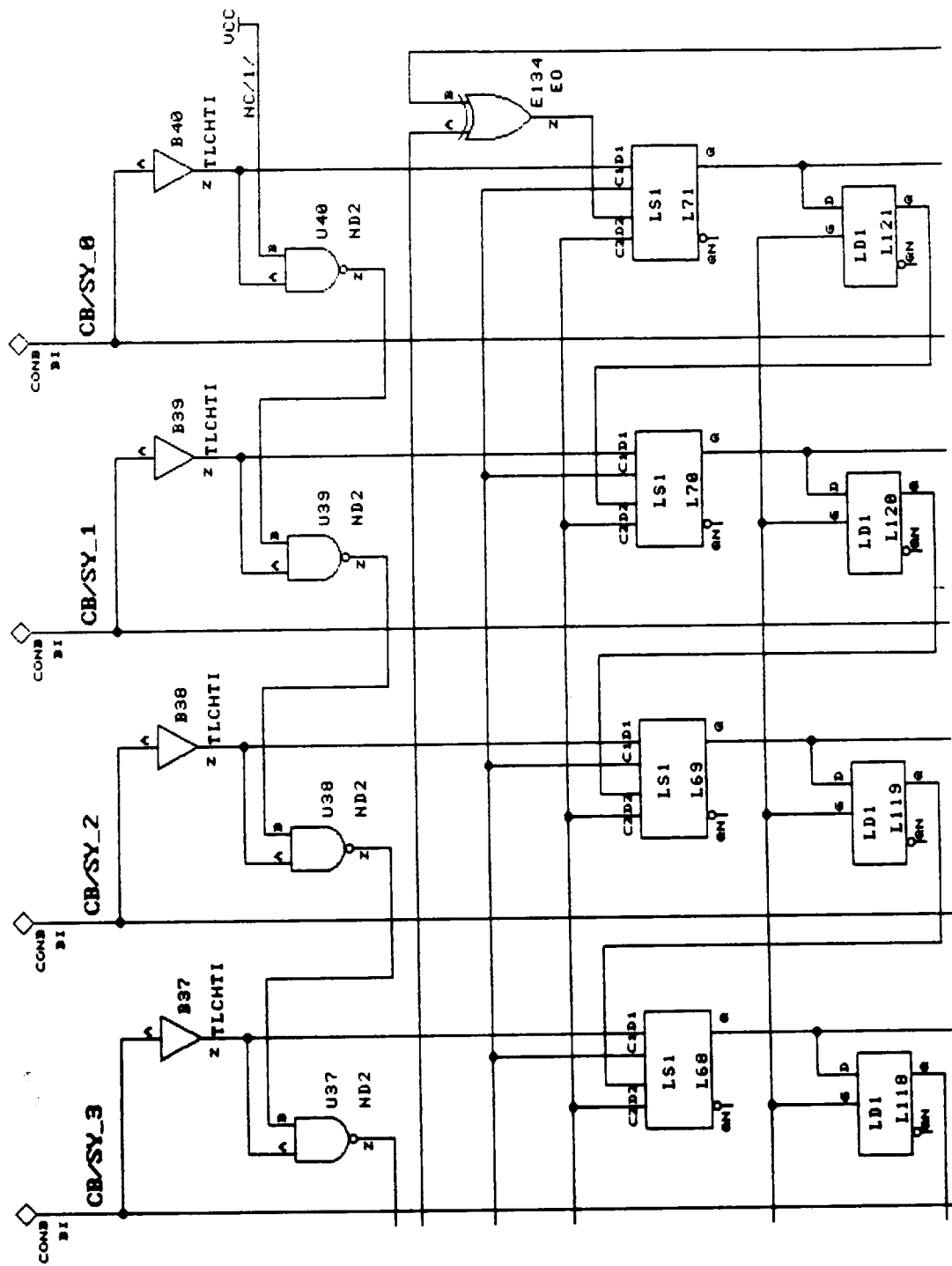


Figure 12

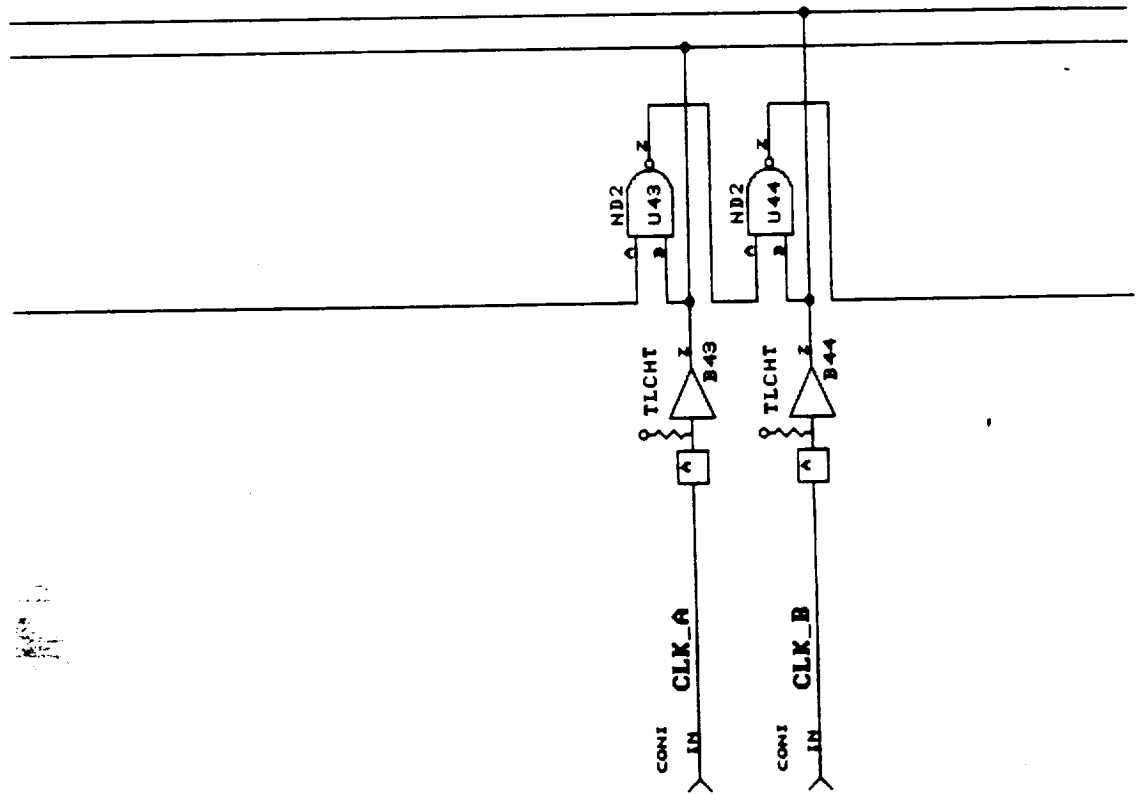
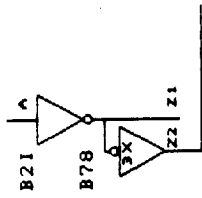


Figure 13

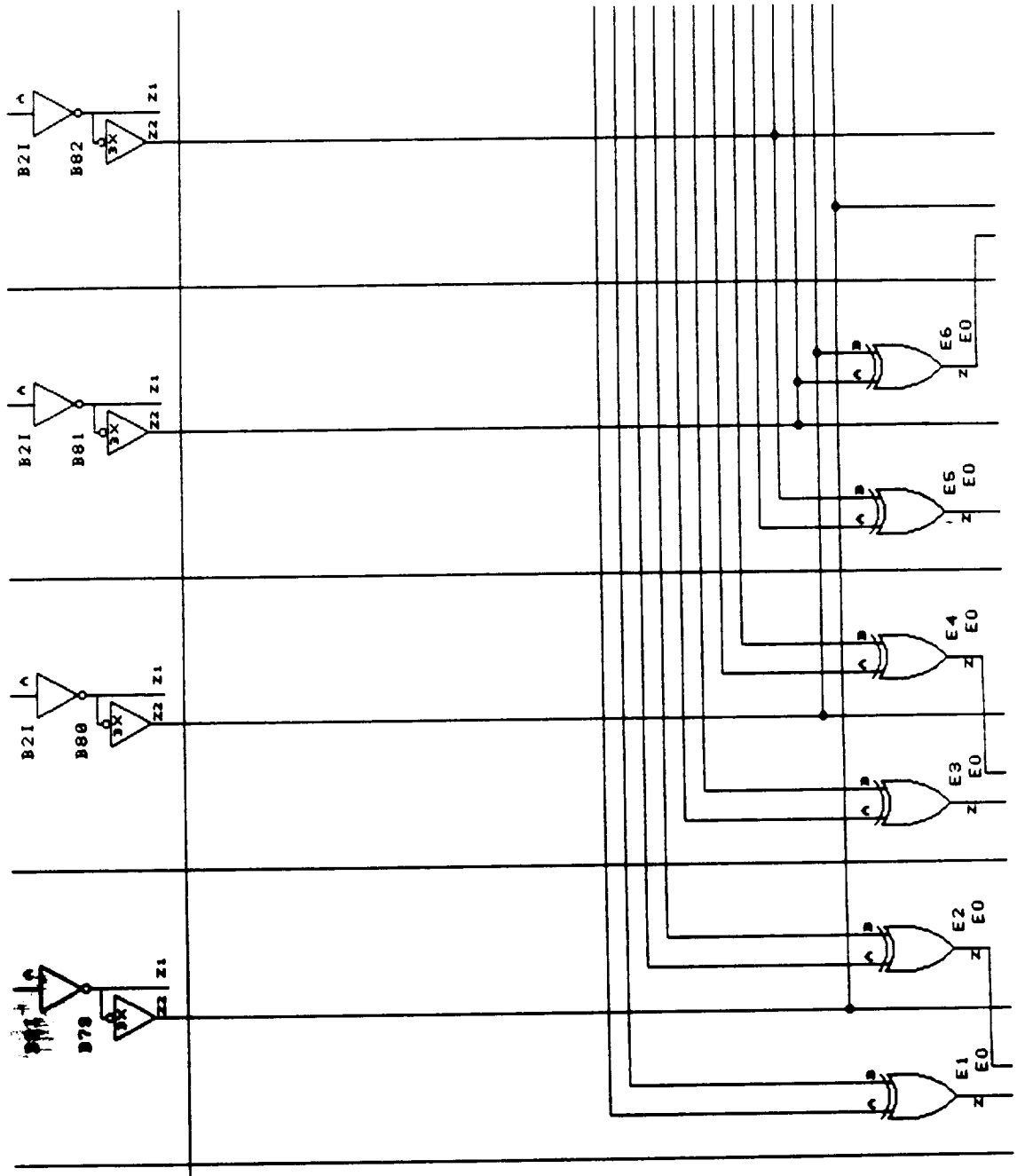


Figure 14

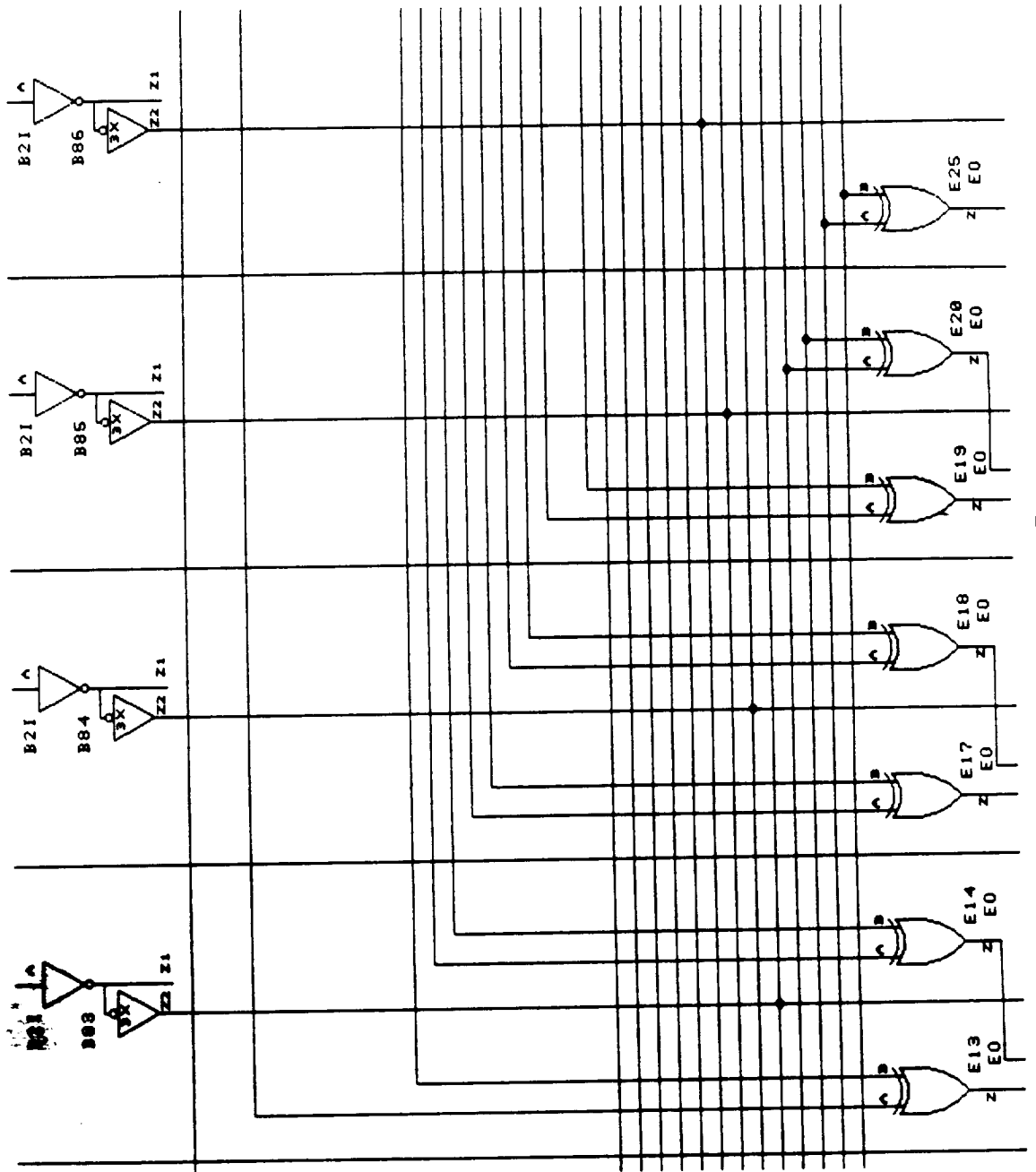


Figure 15

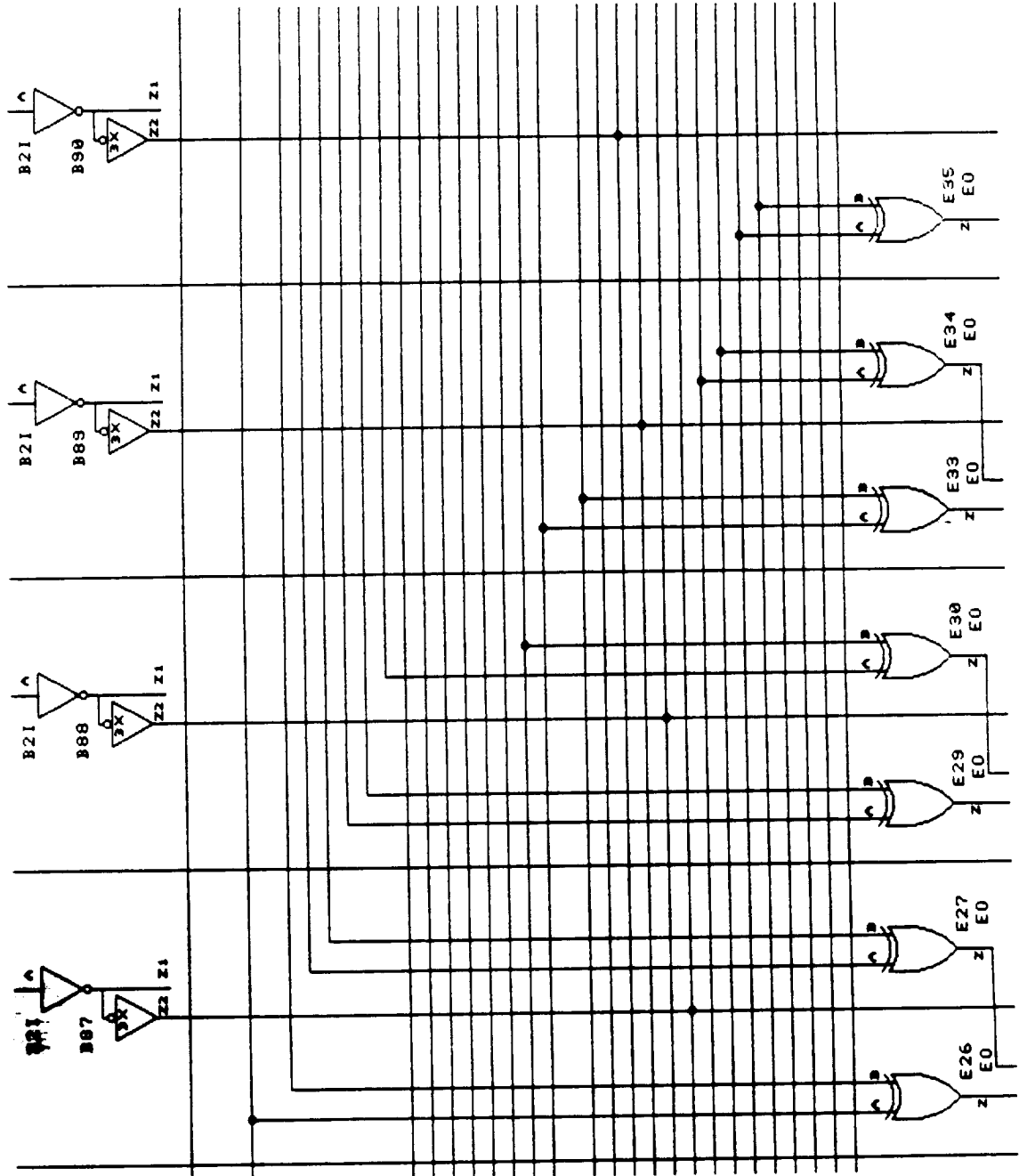


Figure 16

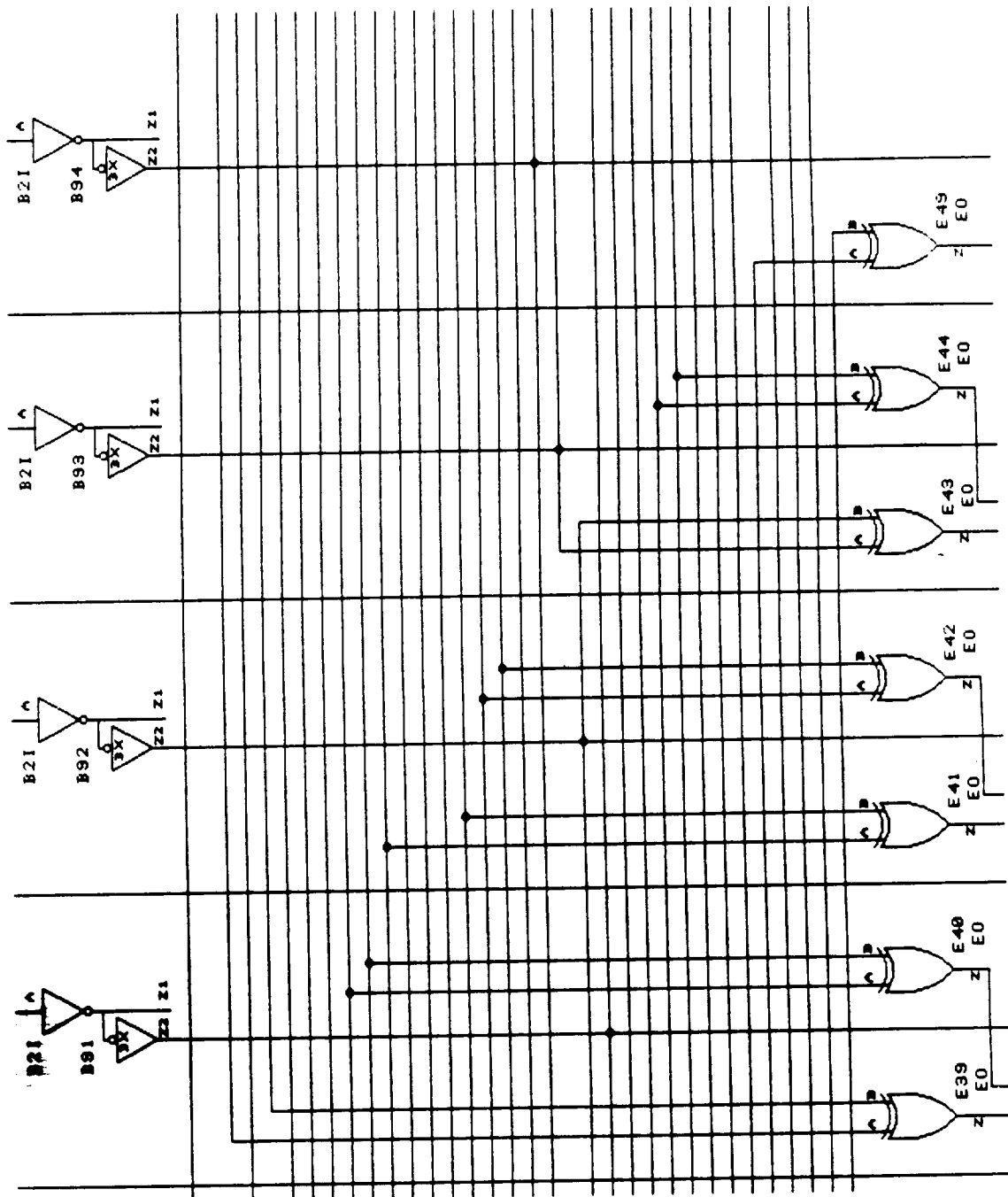


Figure 17

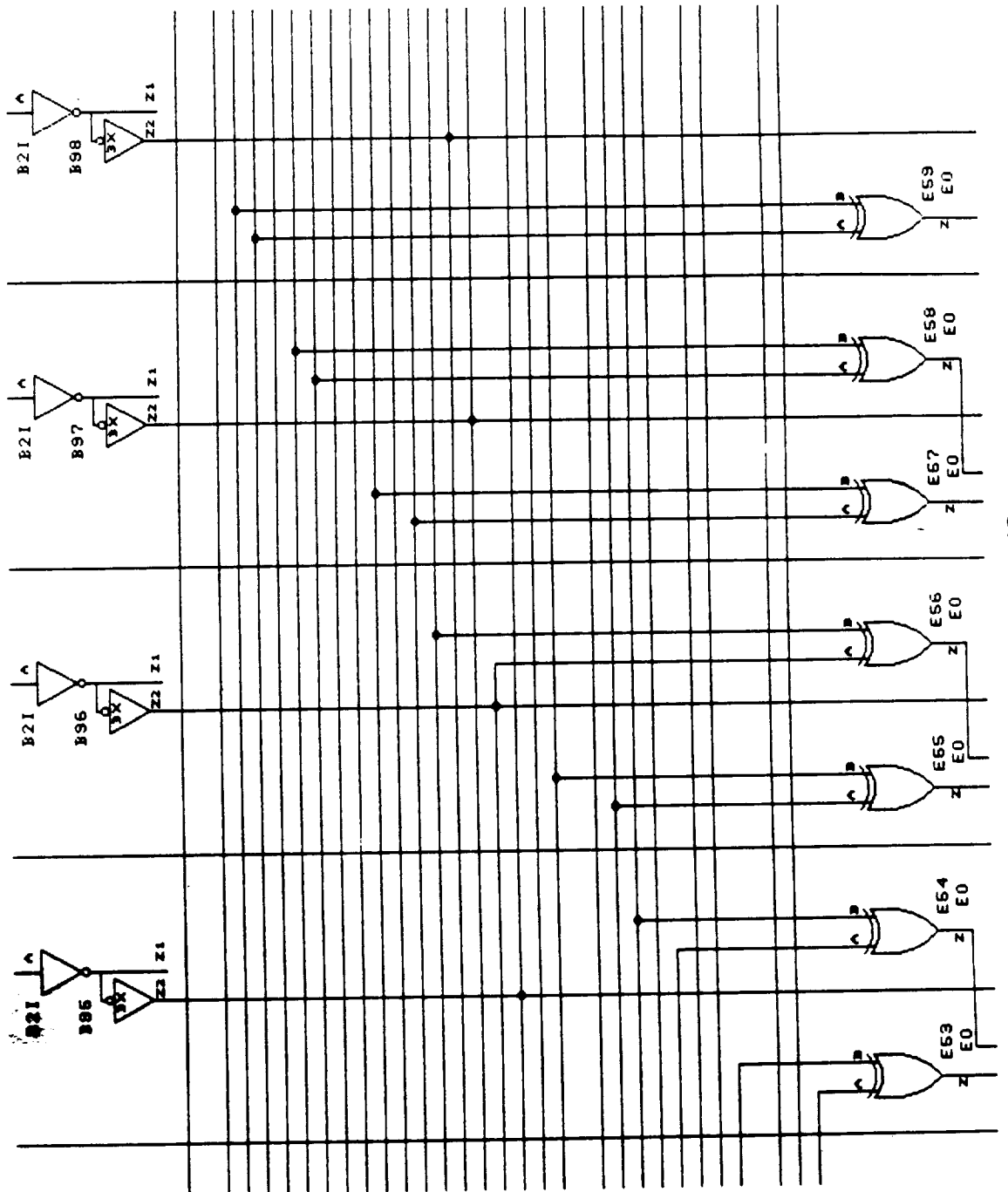


Figure 18

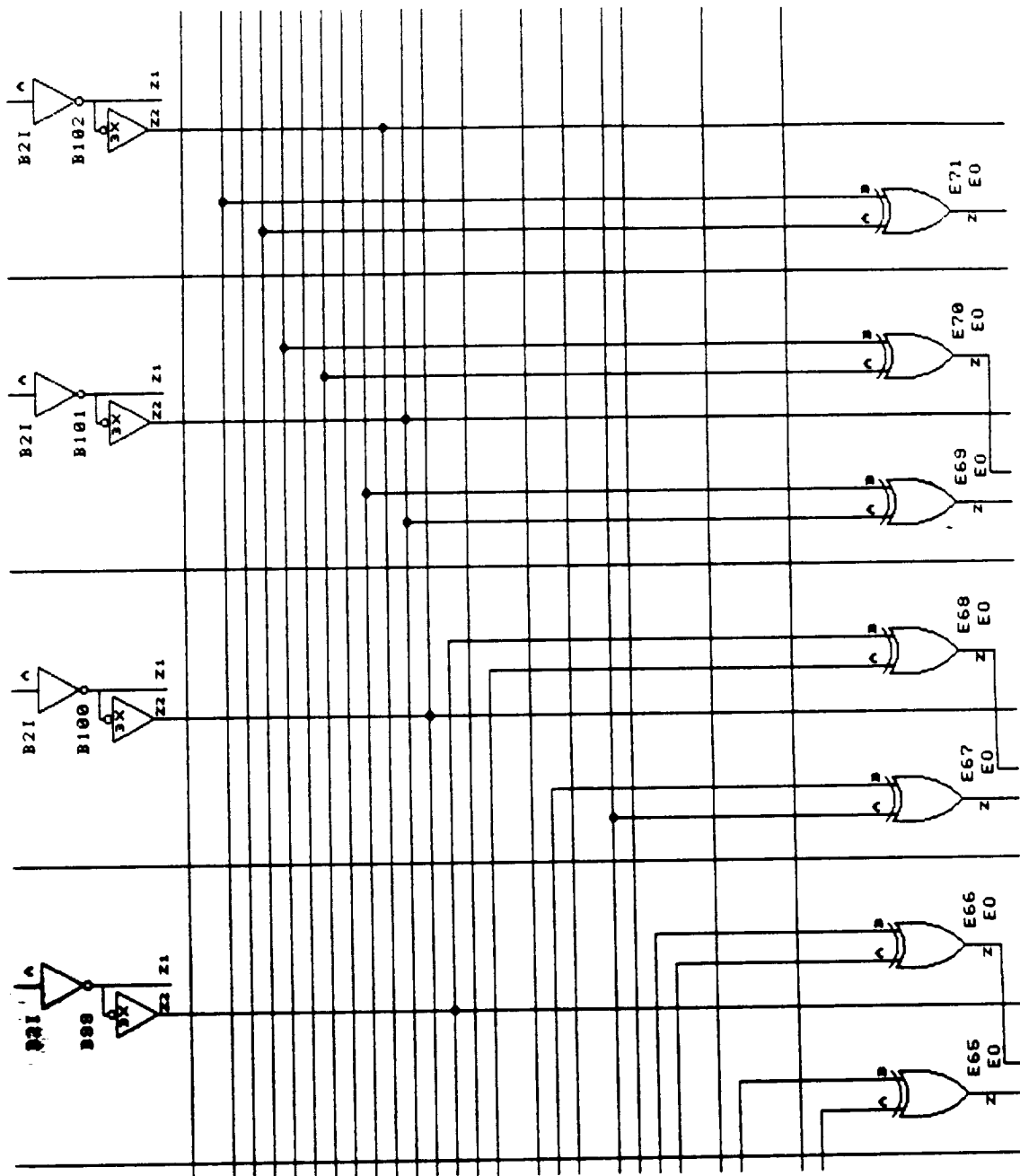


Figure 19

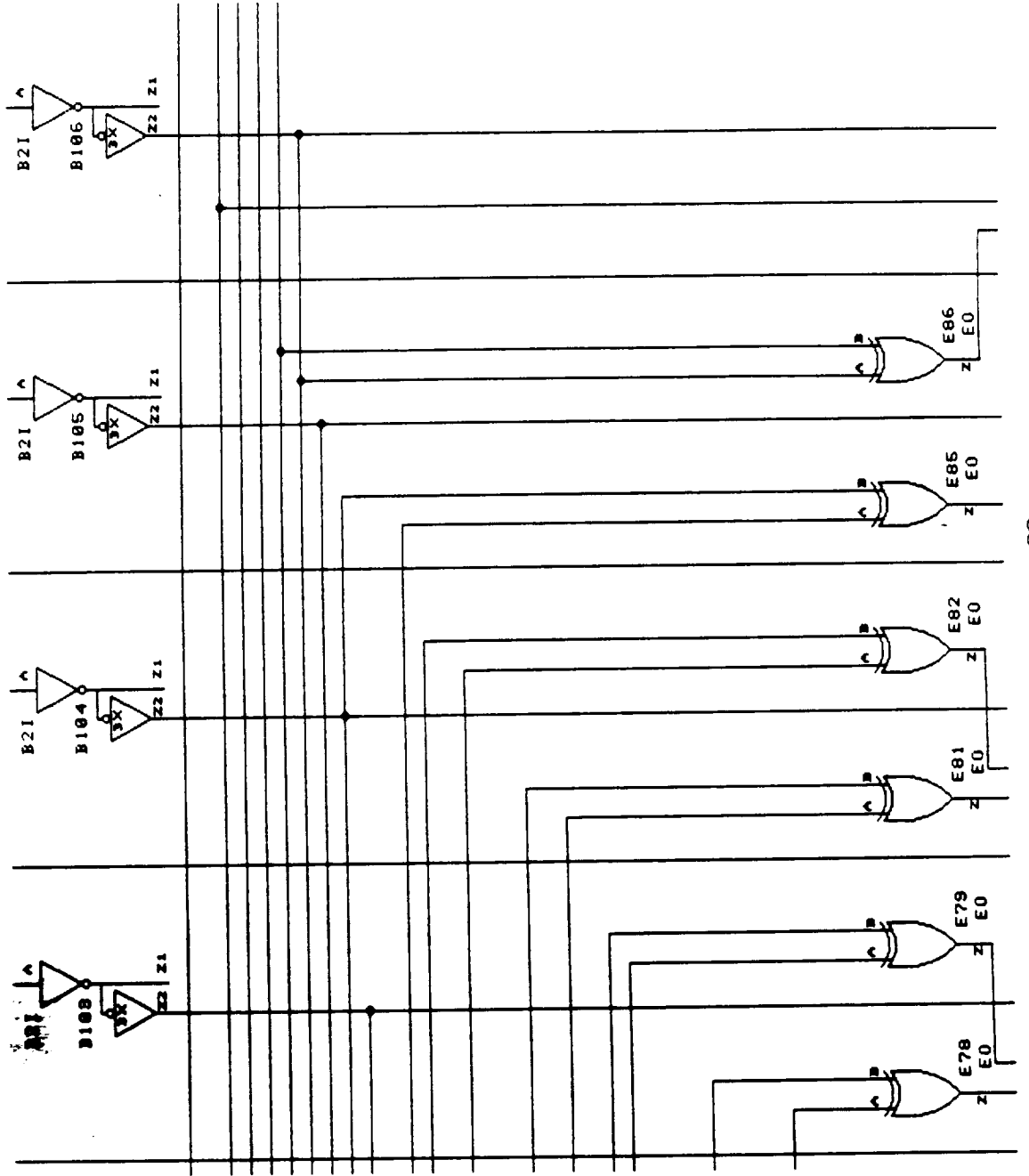


Figure 28

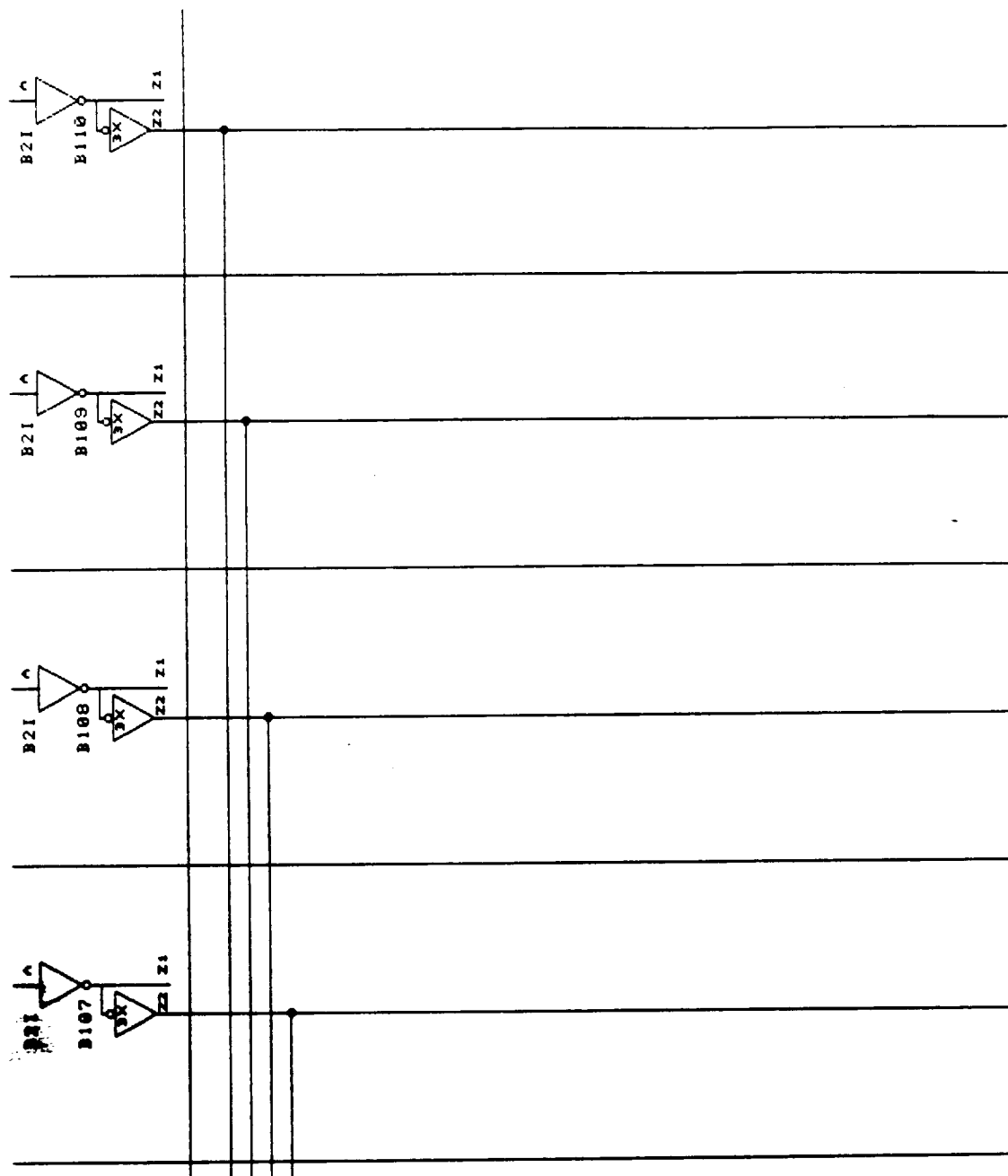


Figure 21

Figure 22

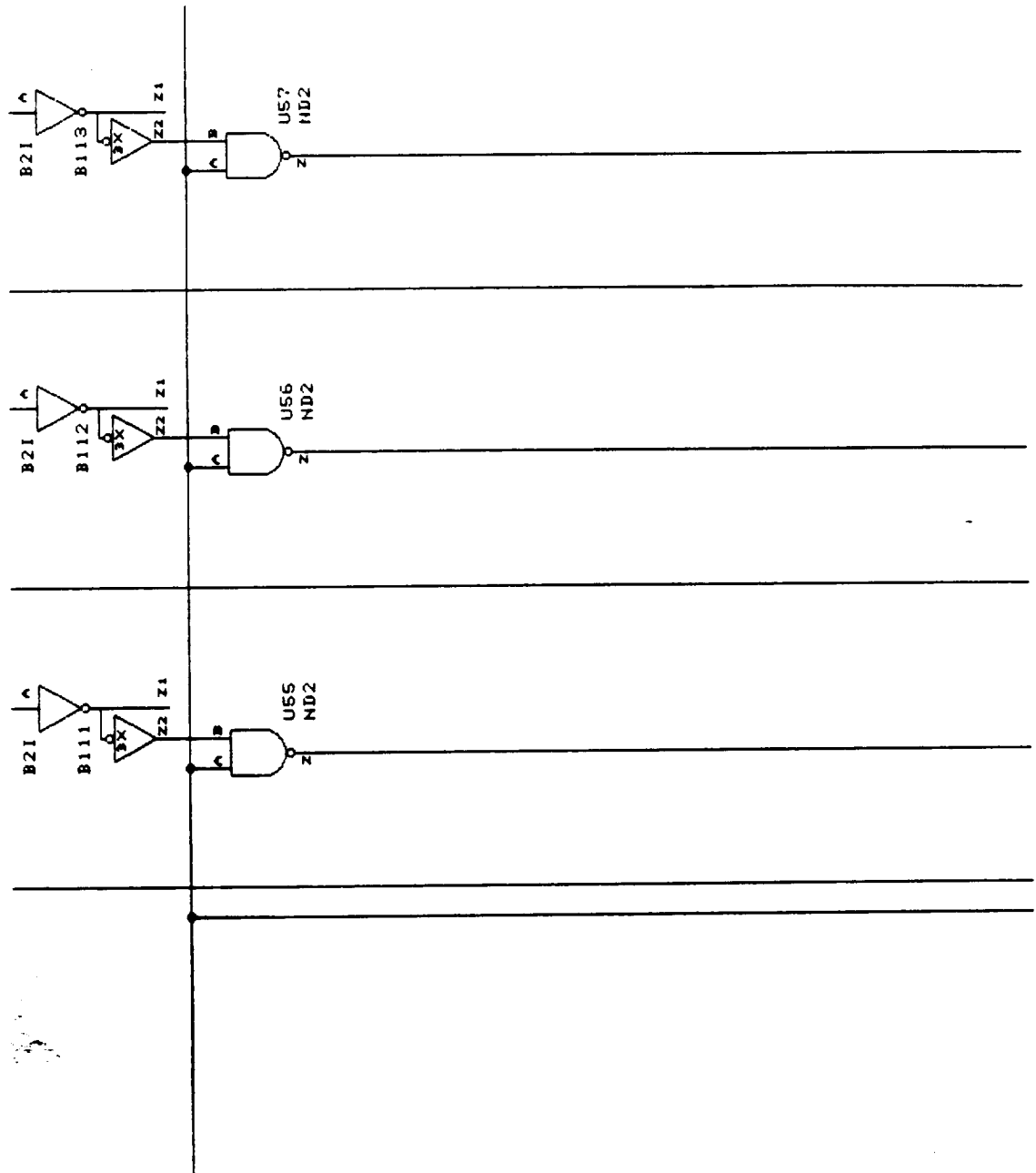


Figure 23

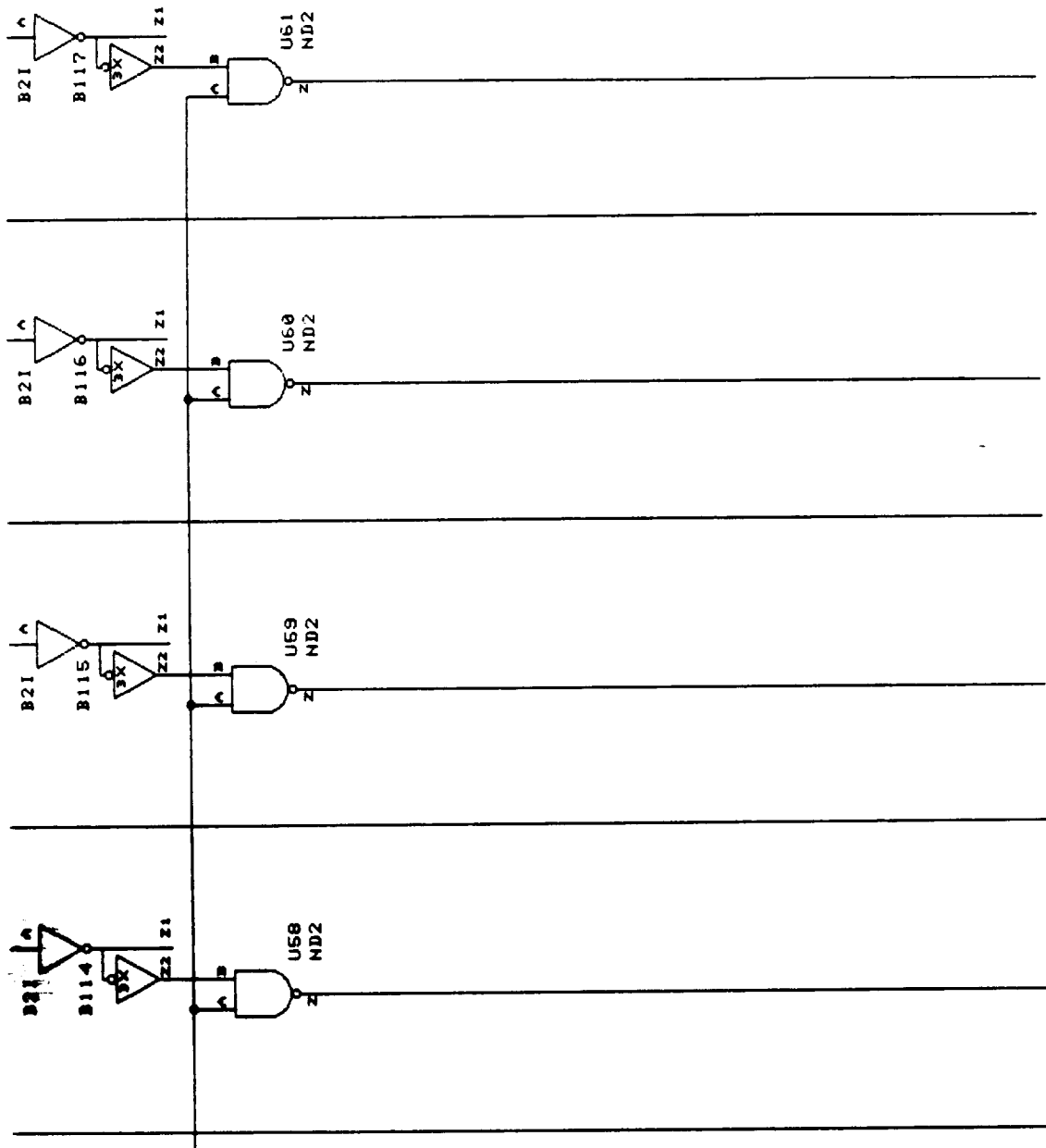


Figure 74

Figure 25

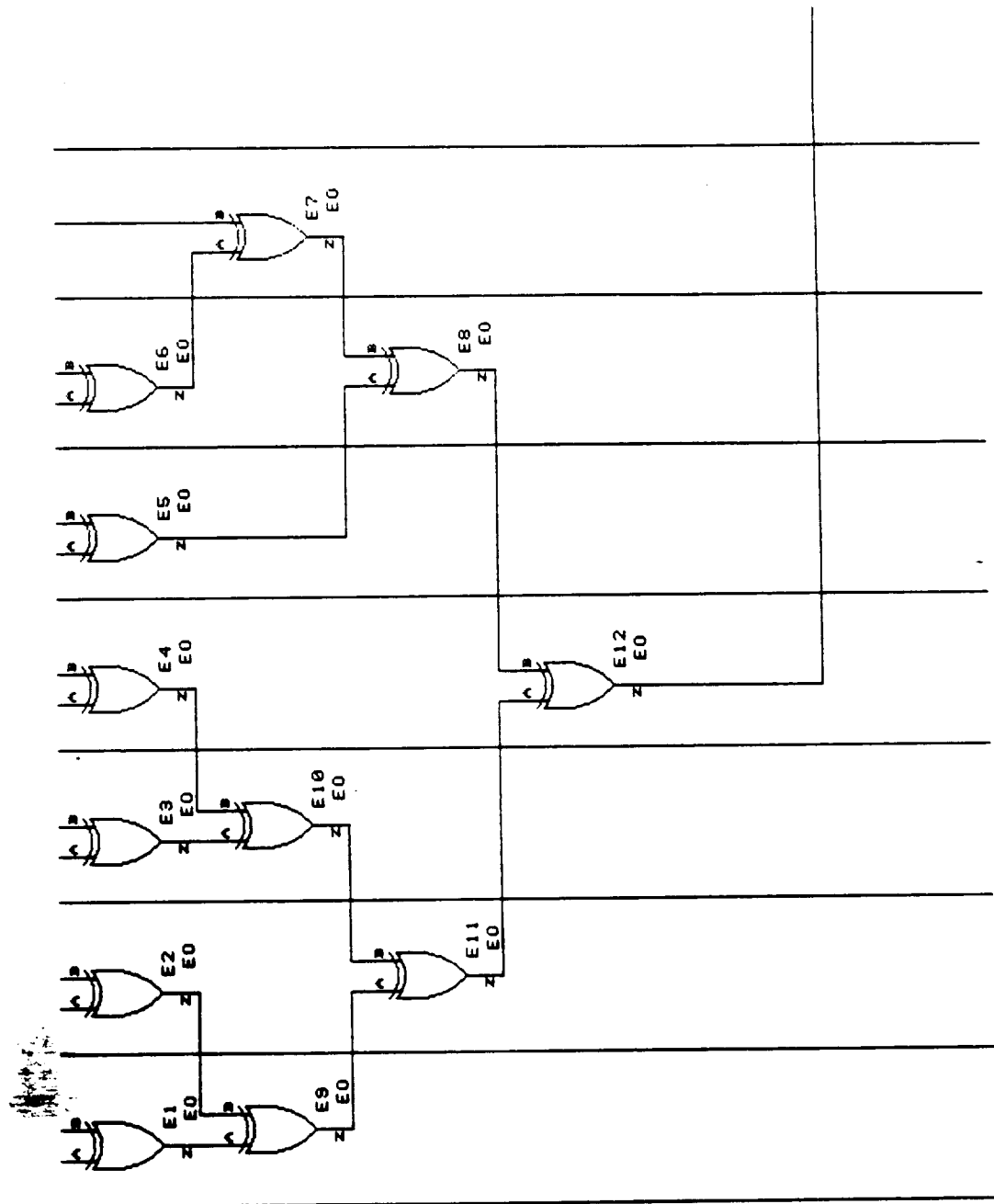


Figure 26

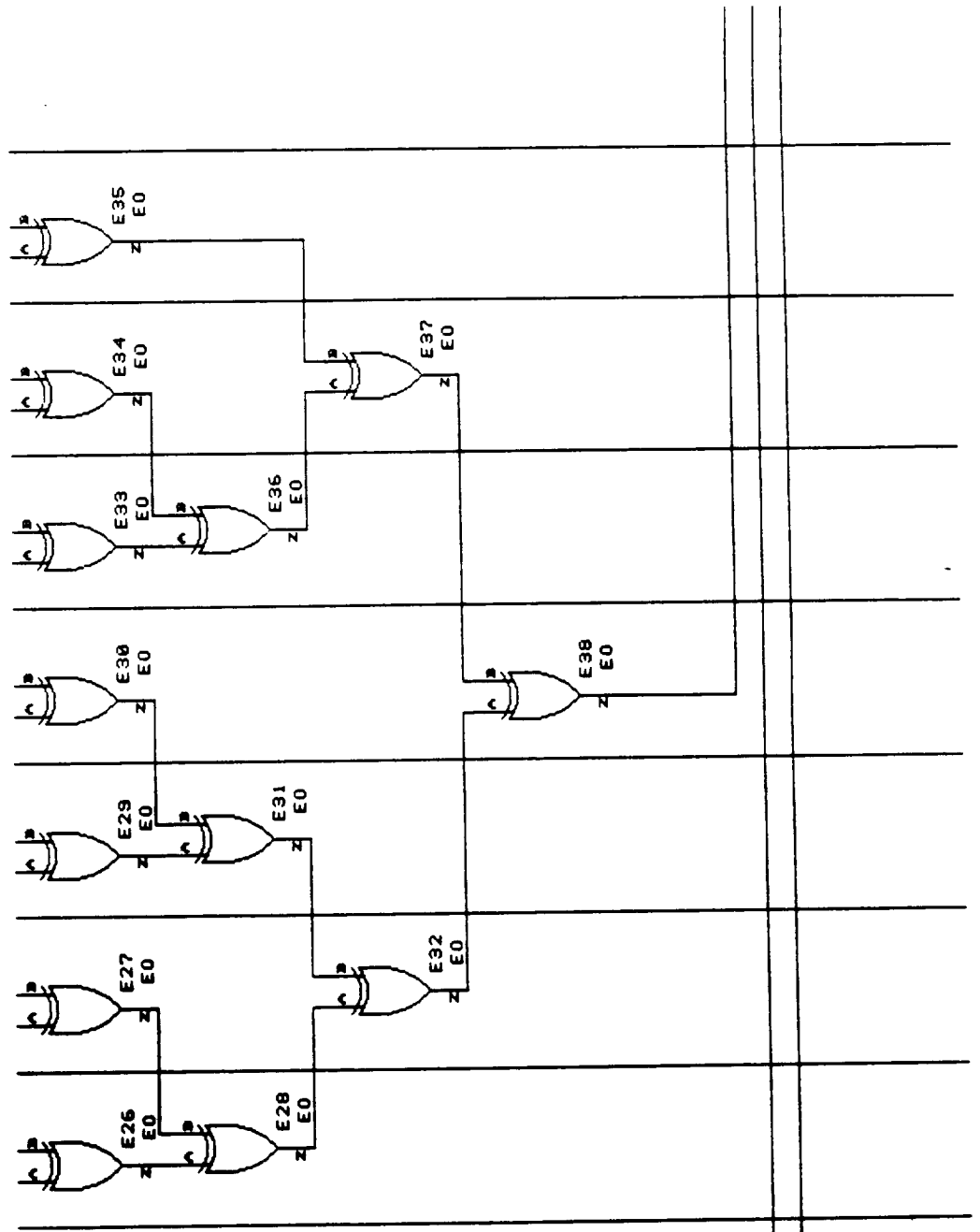


Figure 28

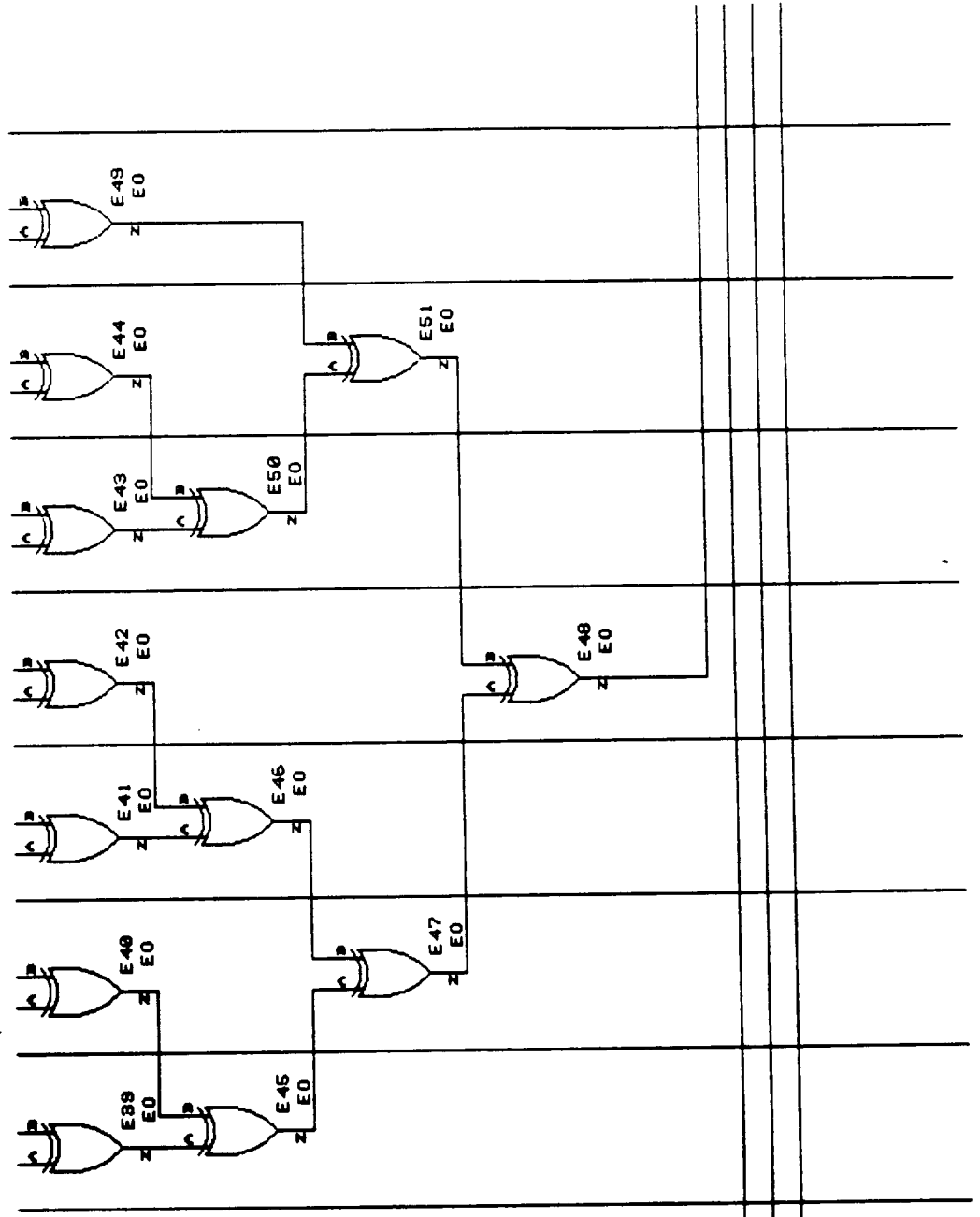


Figure 29

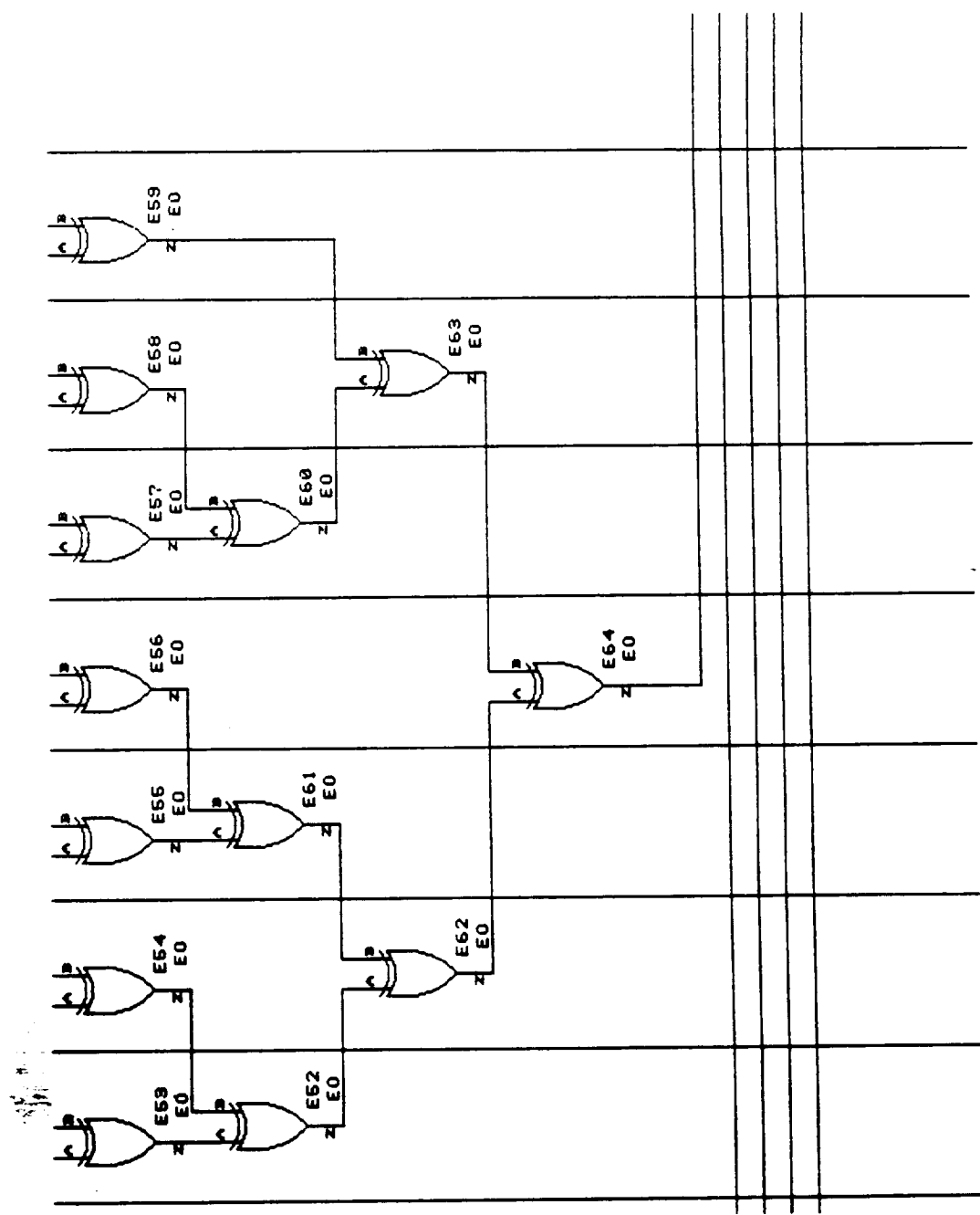


Figure 30

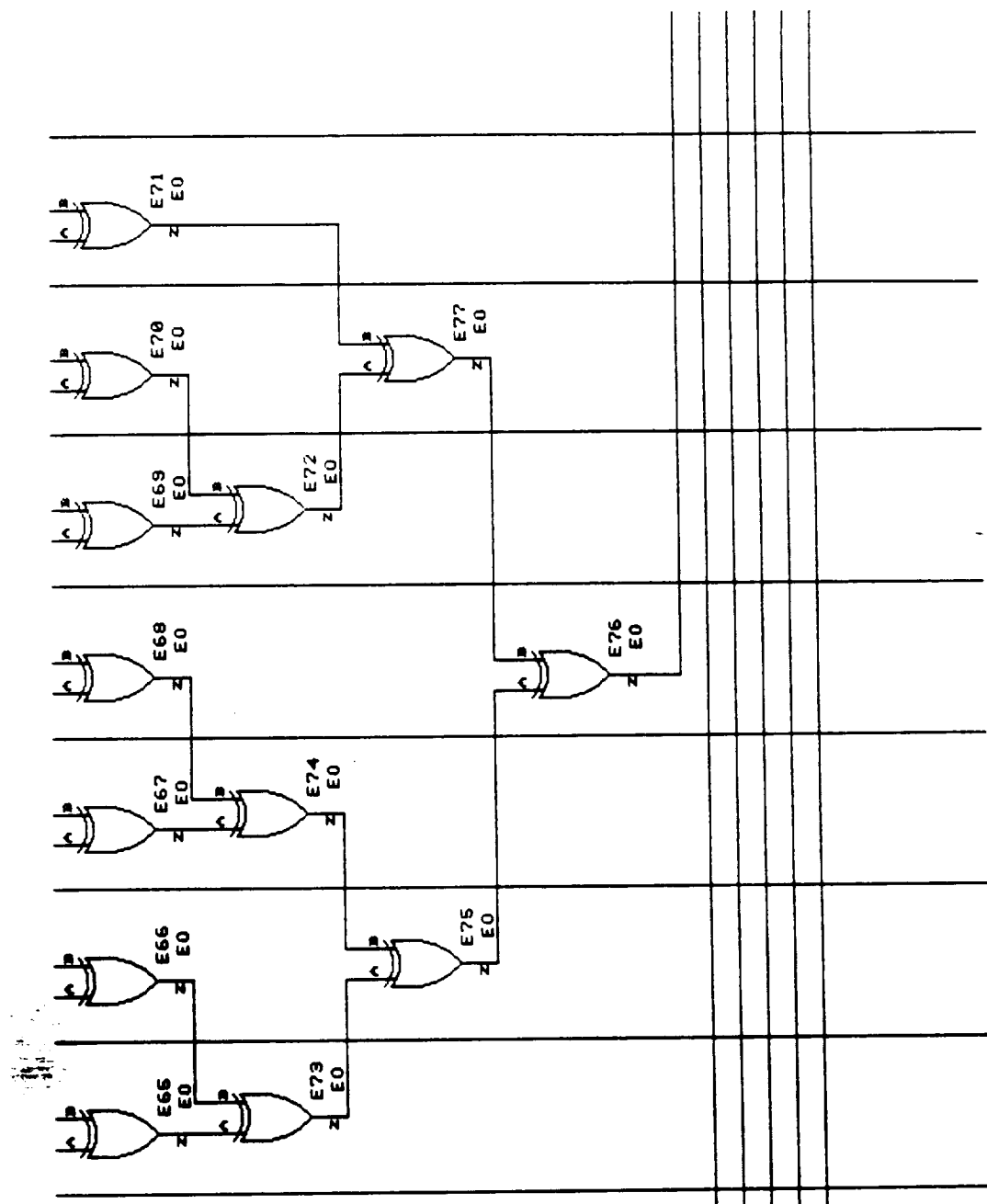


Figure 31

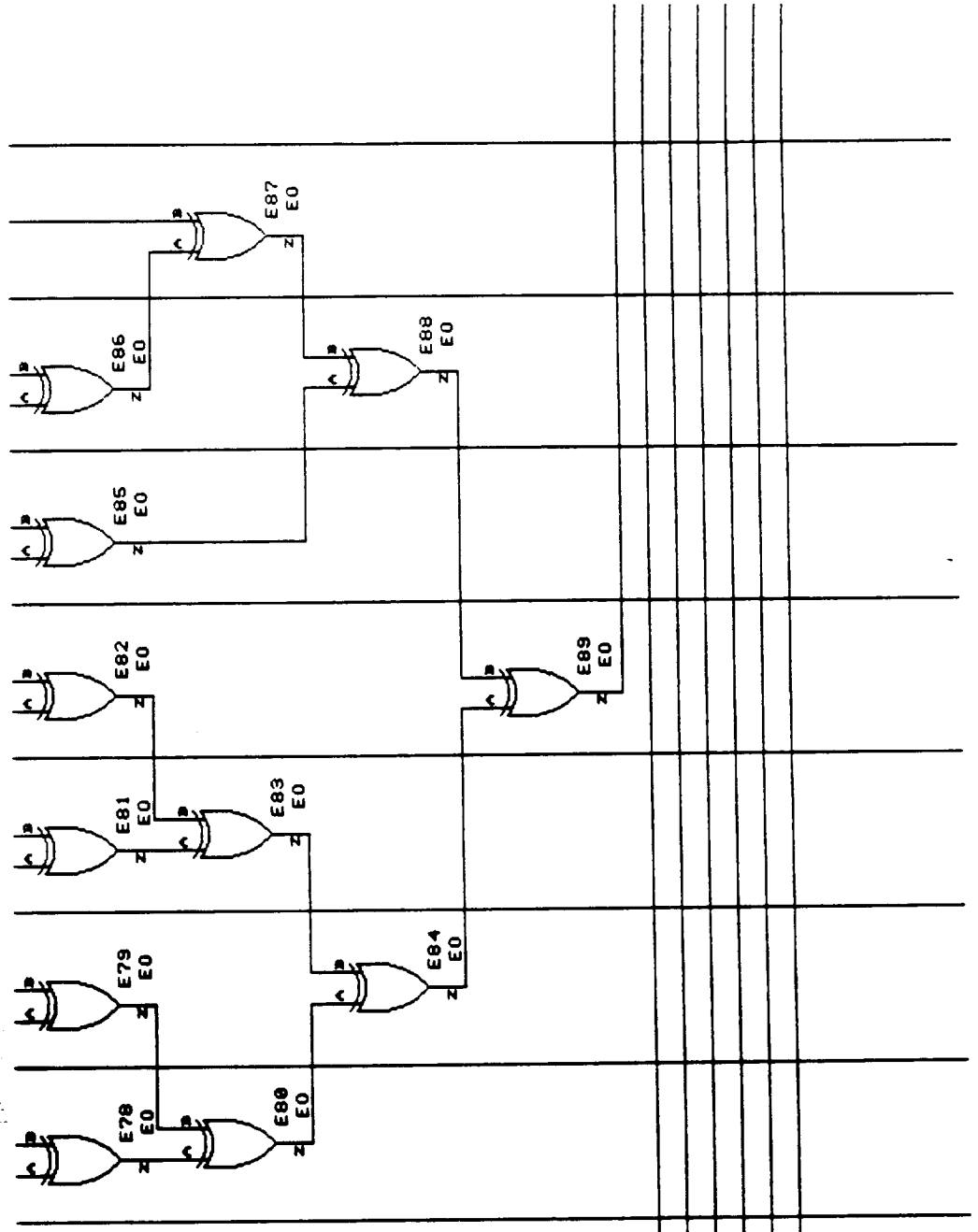


Figure 32

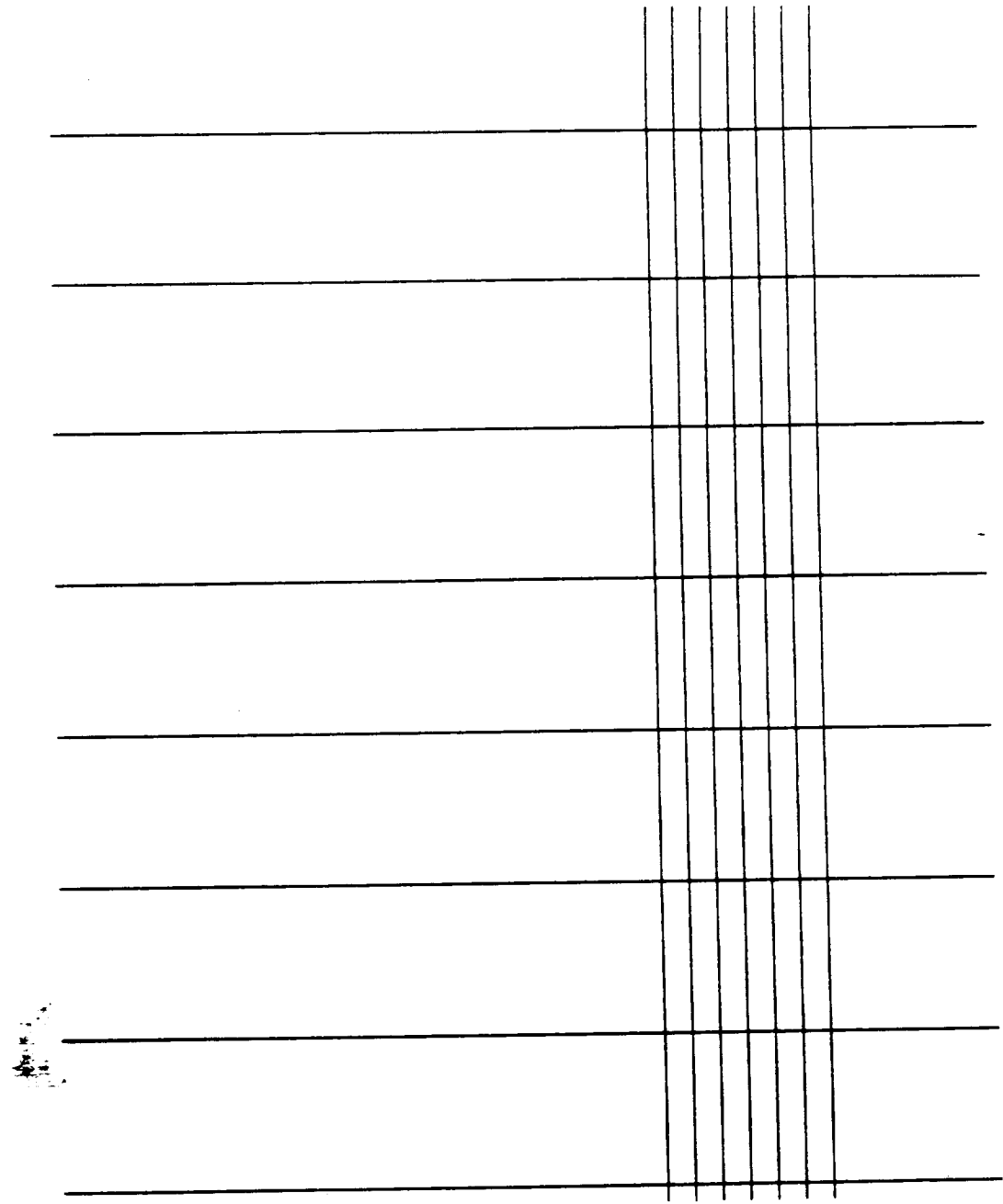


Figure 33

09/04/00

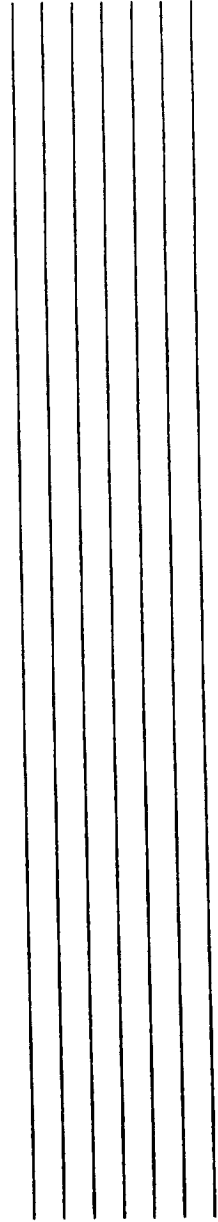


Figure 34

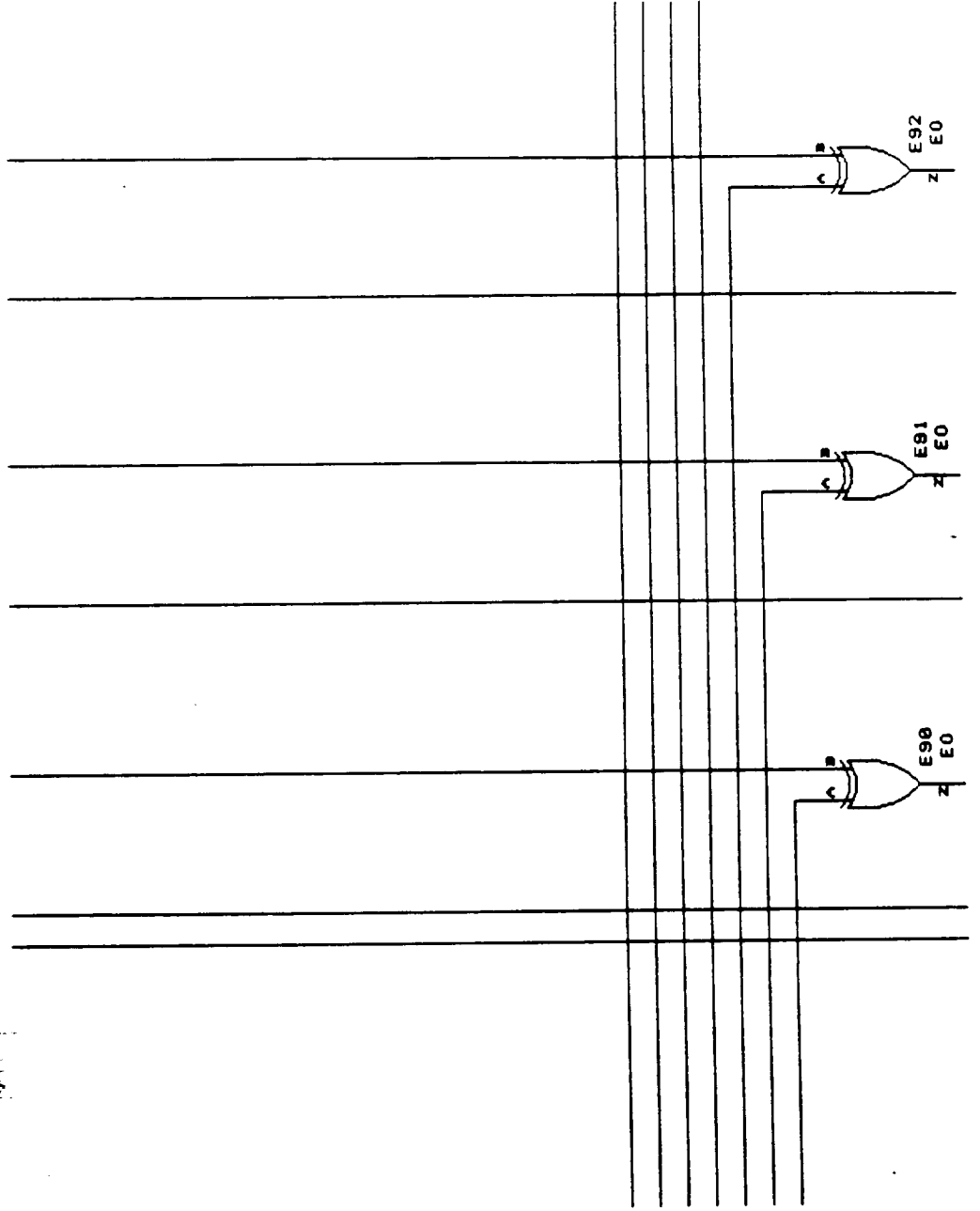


Figure 35

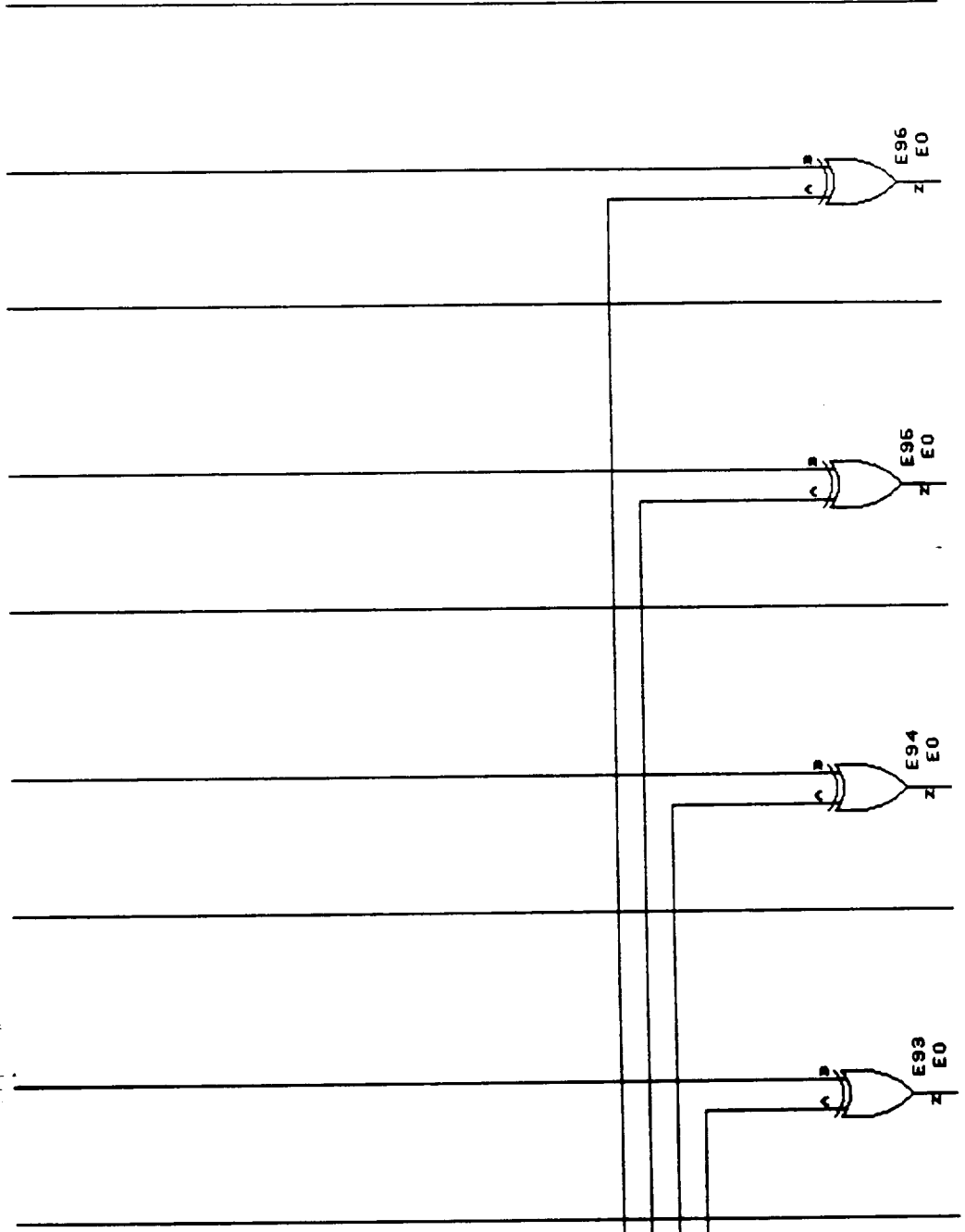


Figure 36

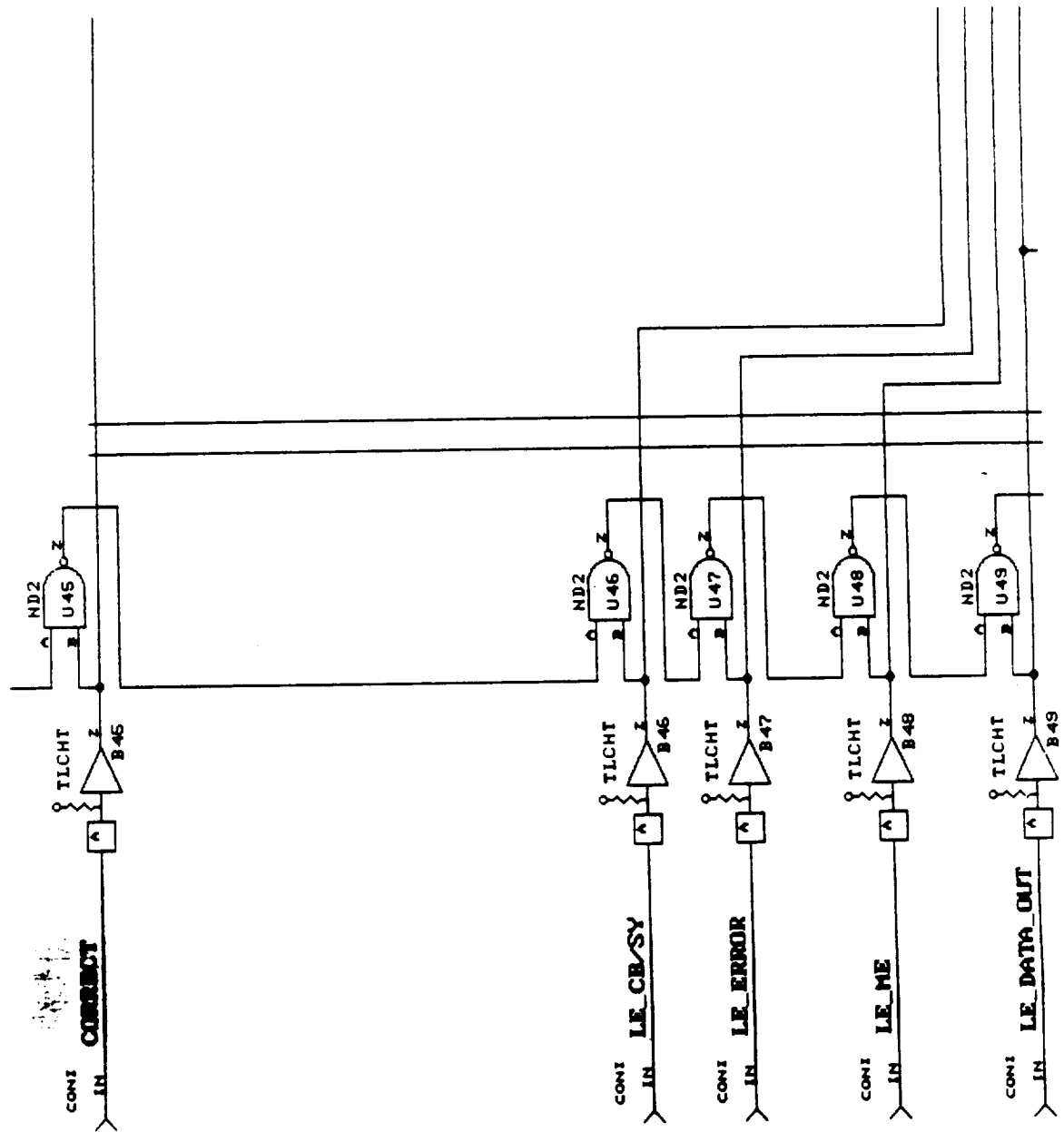


Figure 37

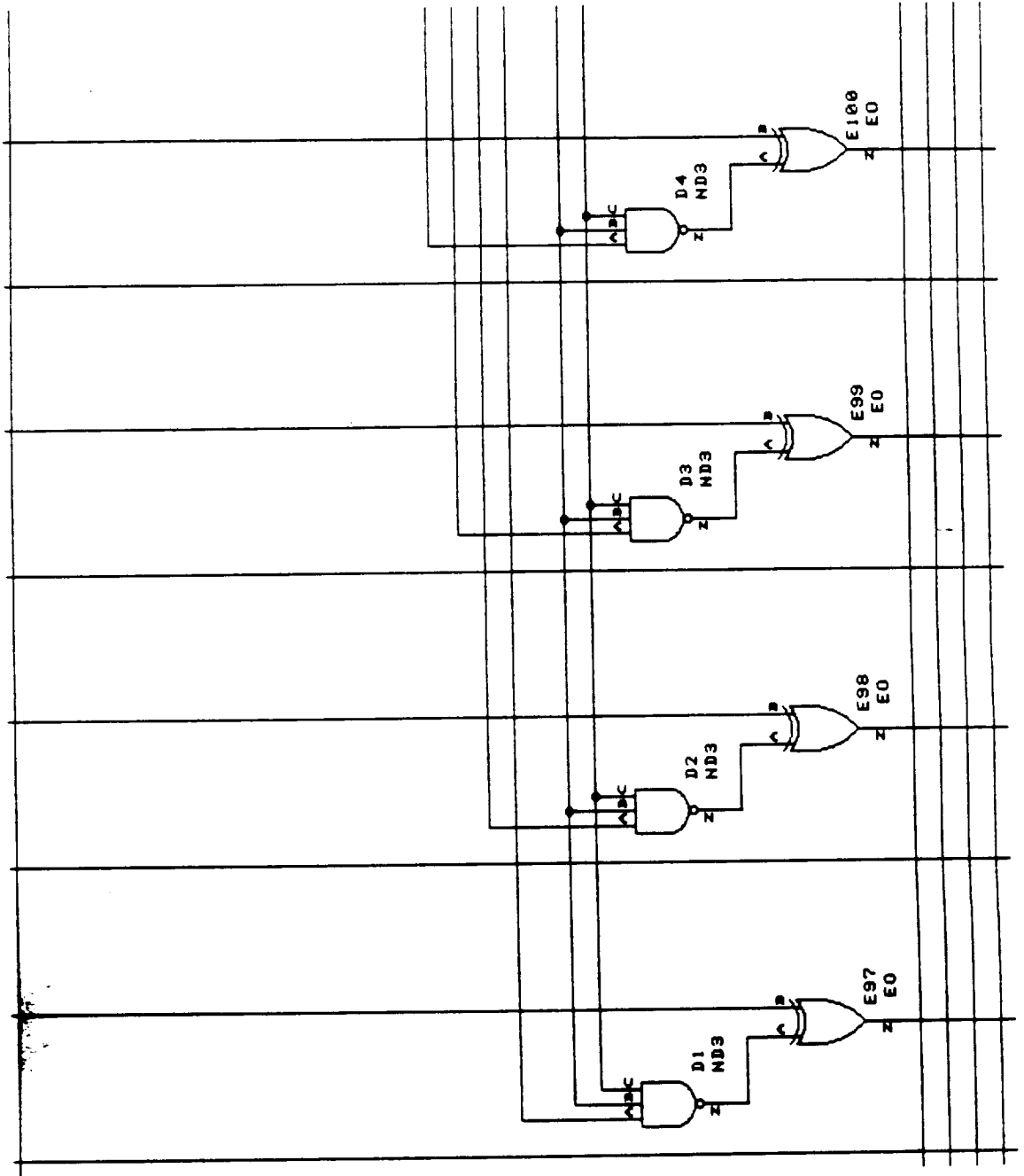


Figure 38

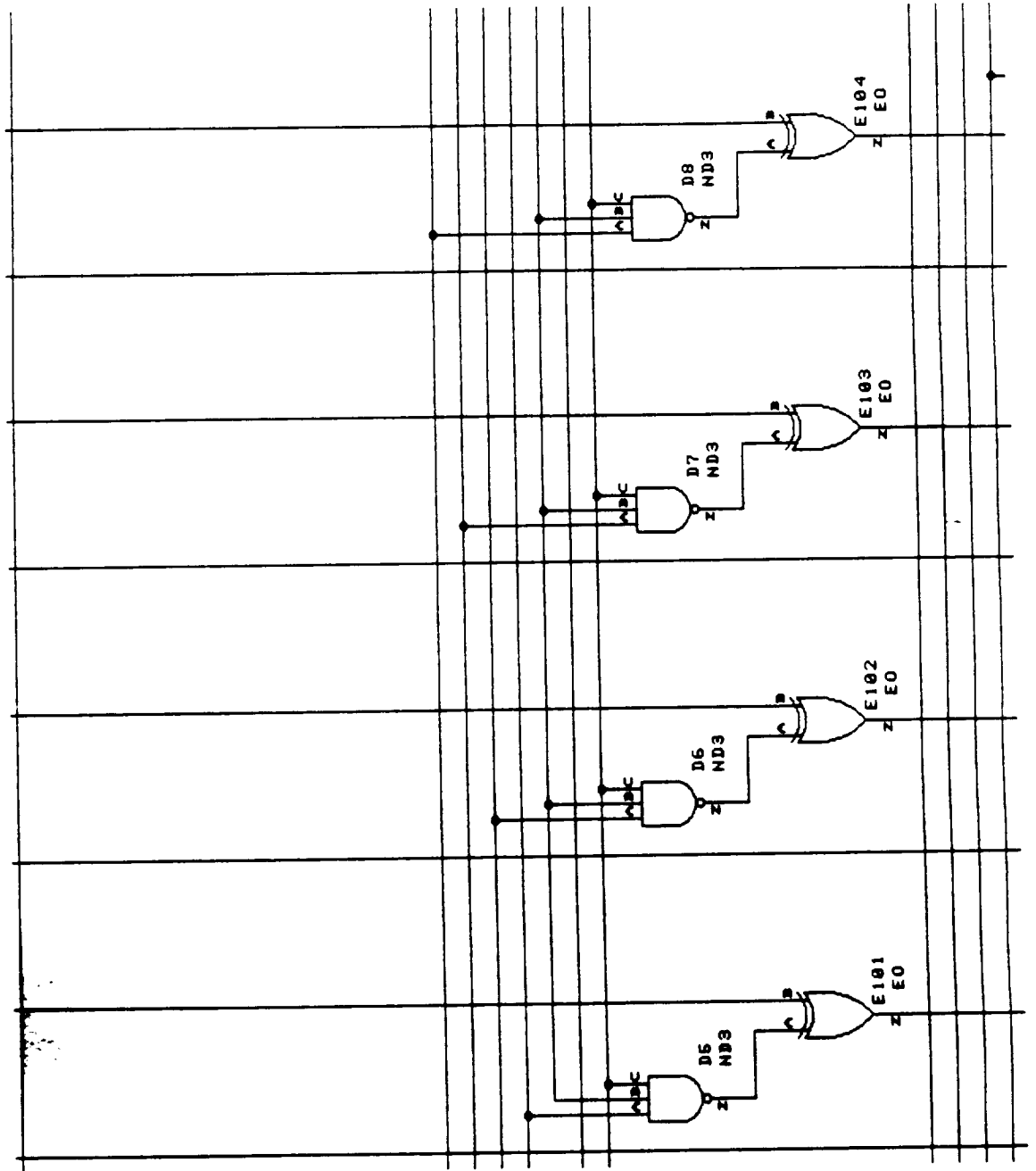


Figure 39

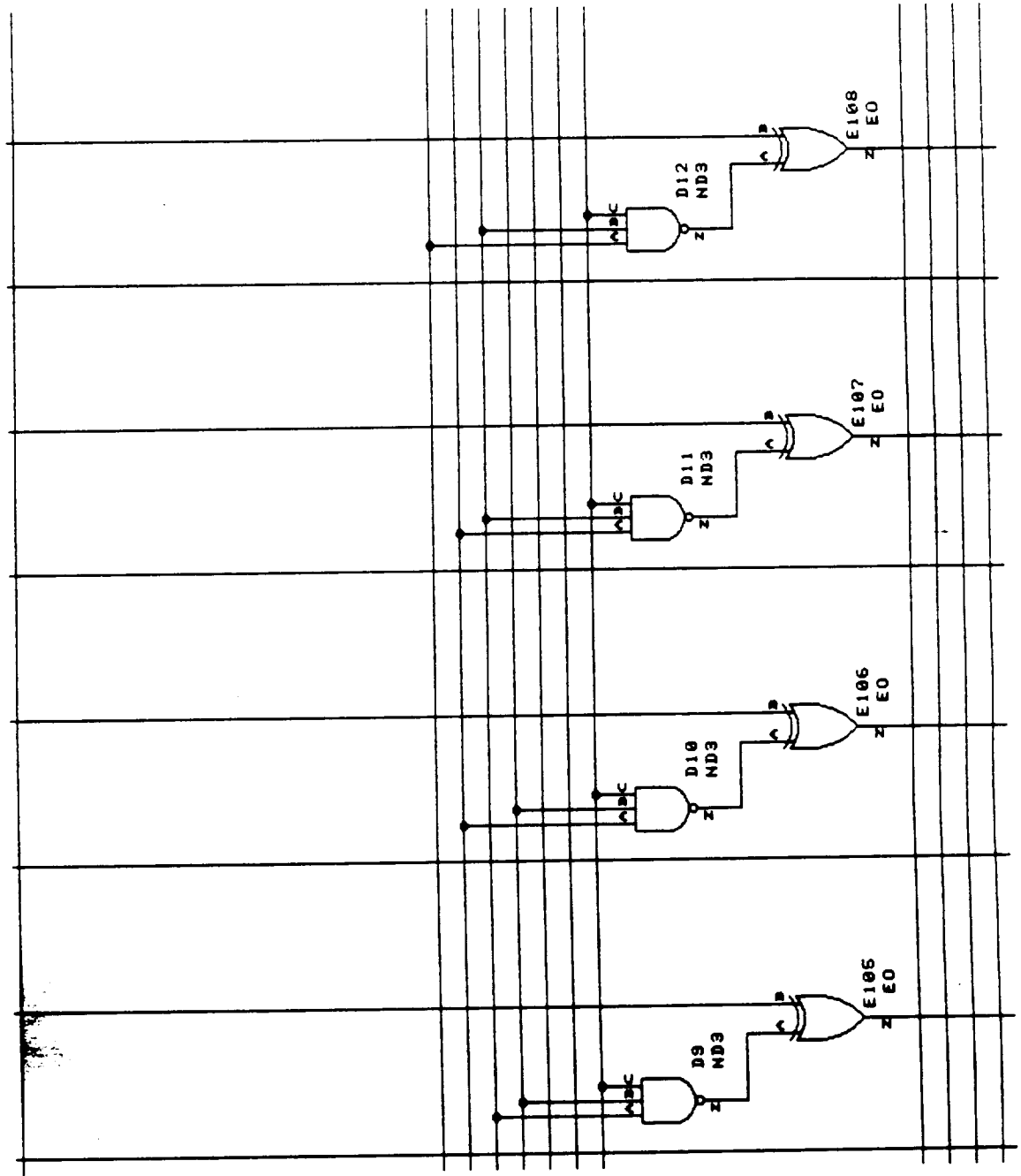


Figure 40

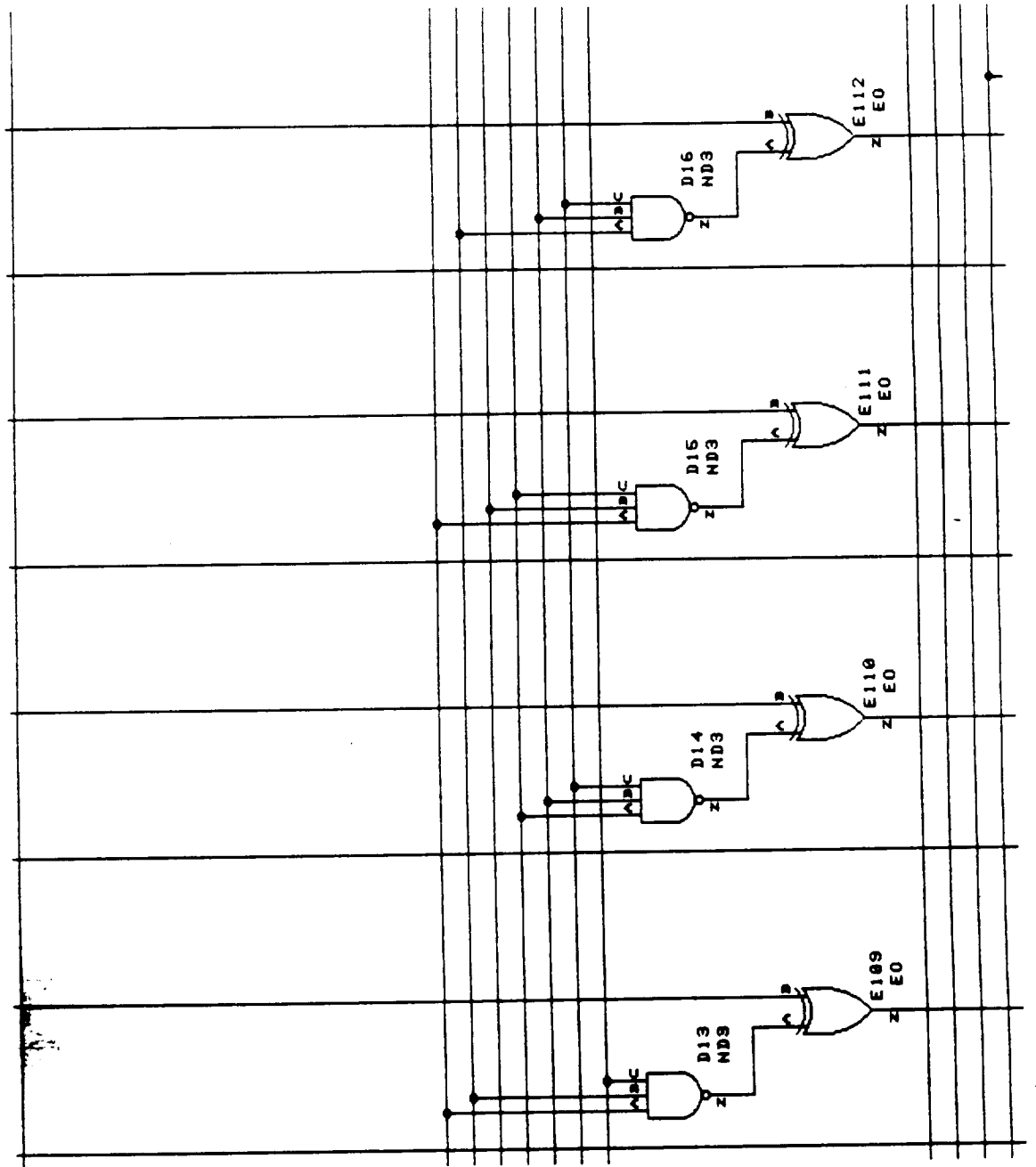


Figure 41

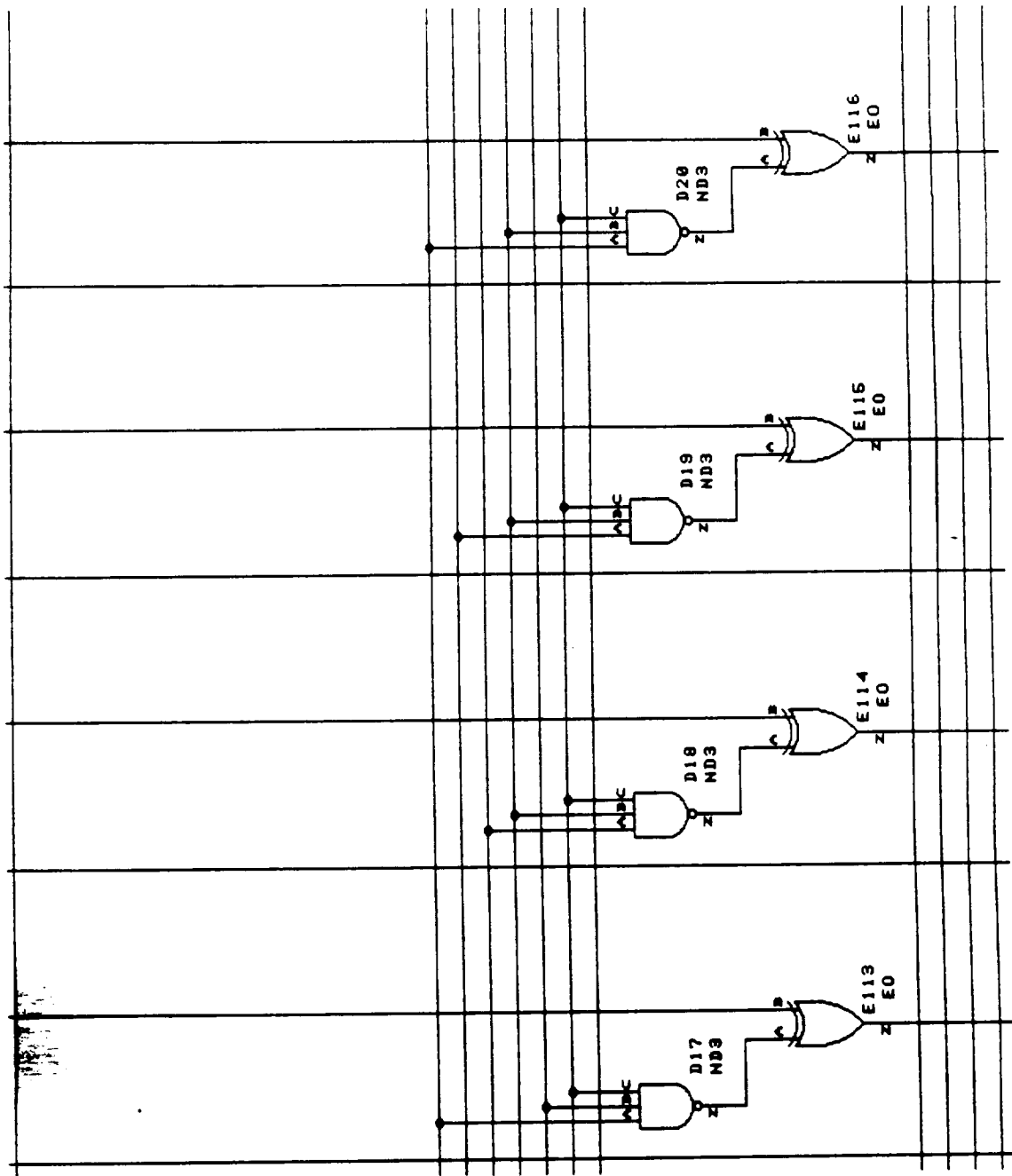


Figure 42

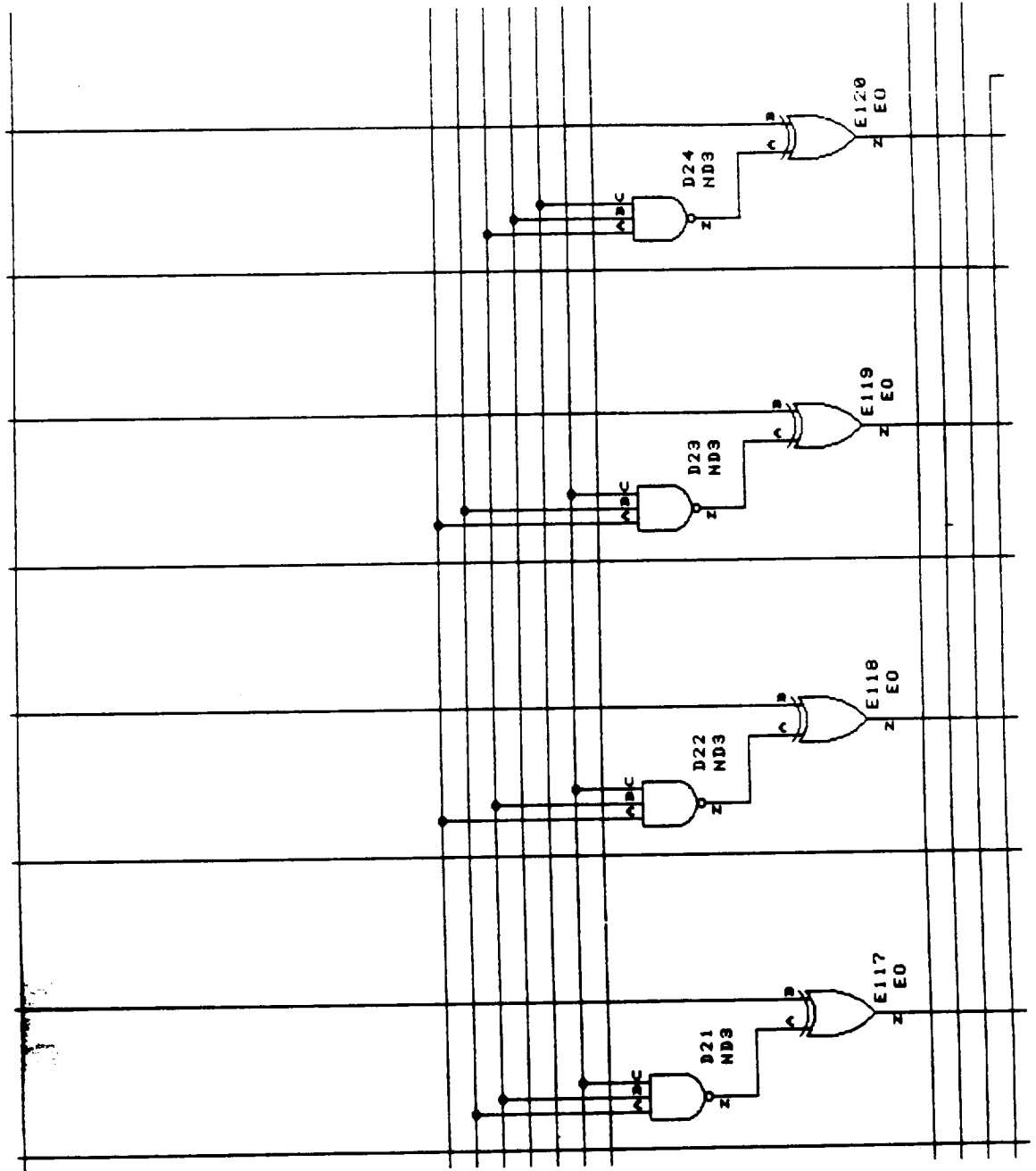


Figure 43

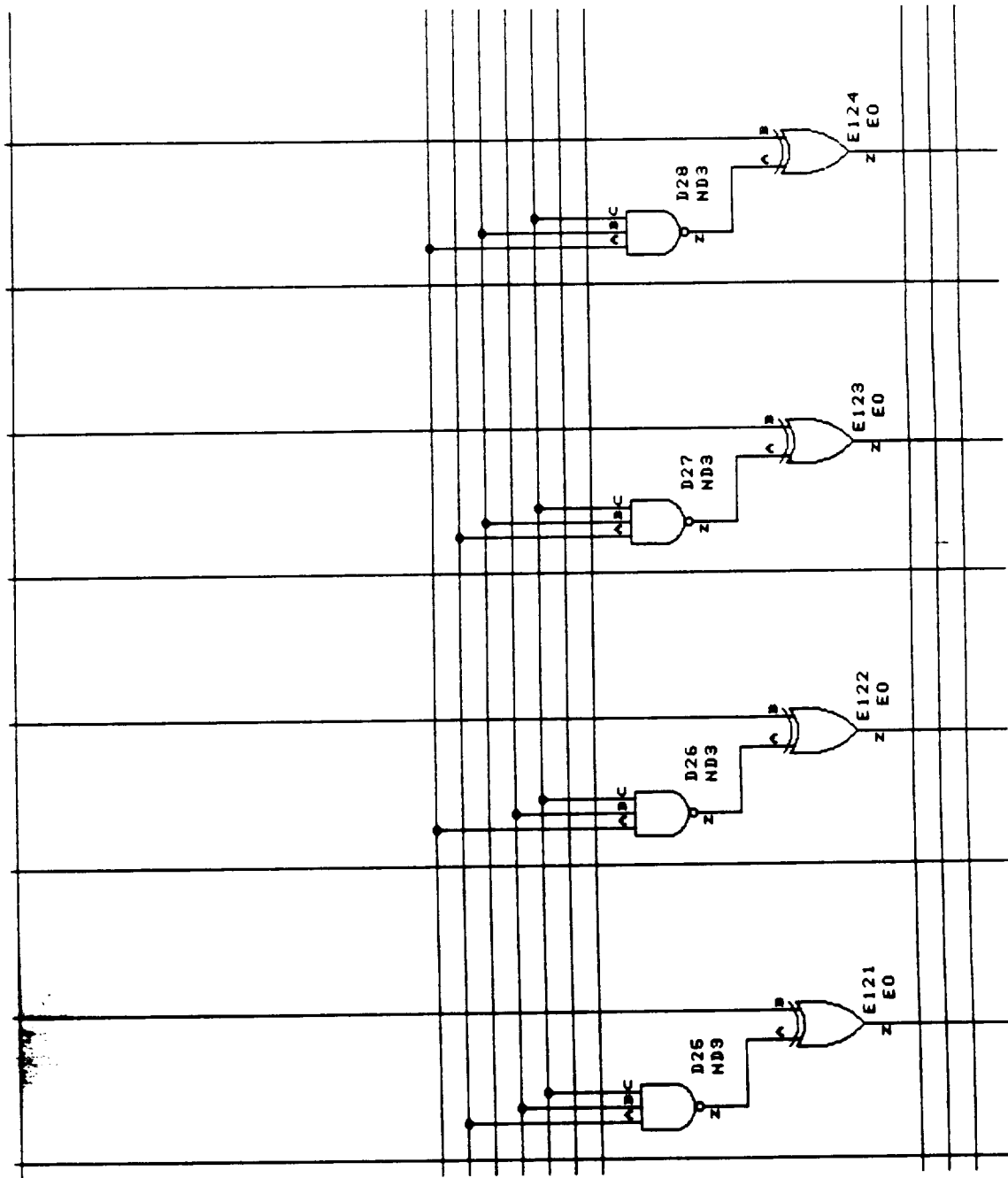


Figure 44

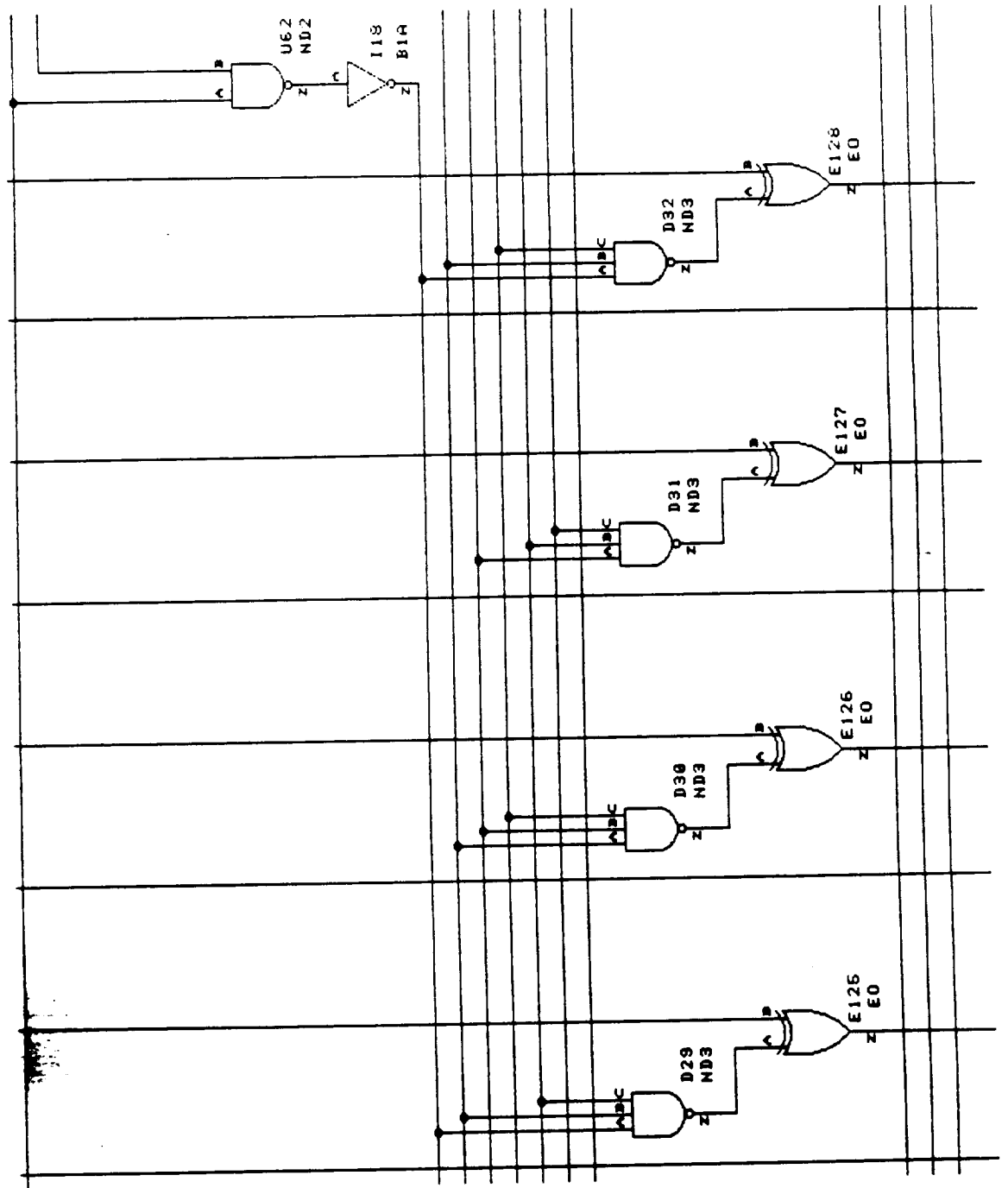


Figure 45

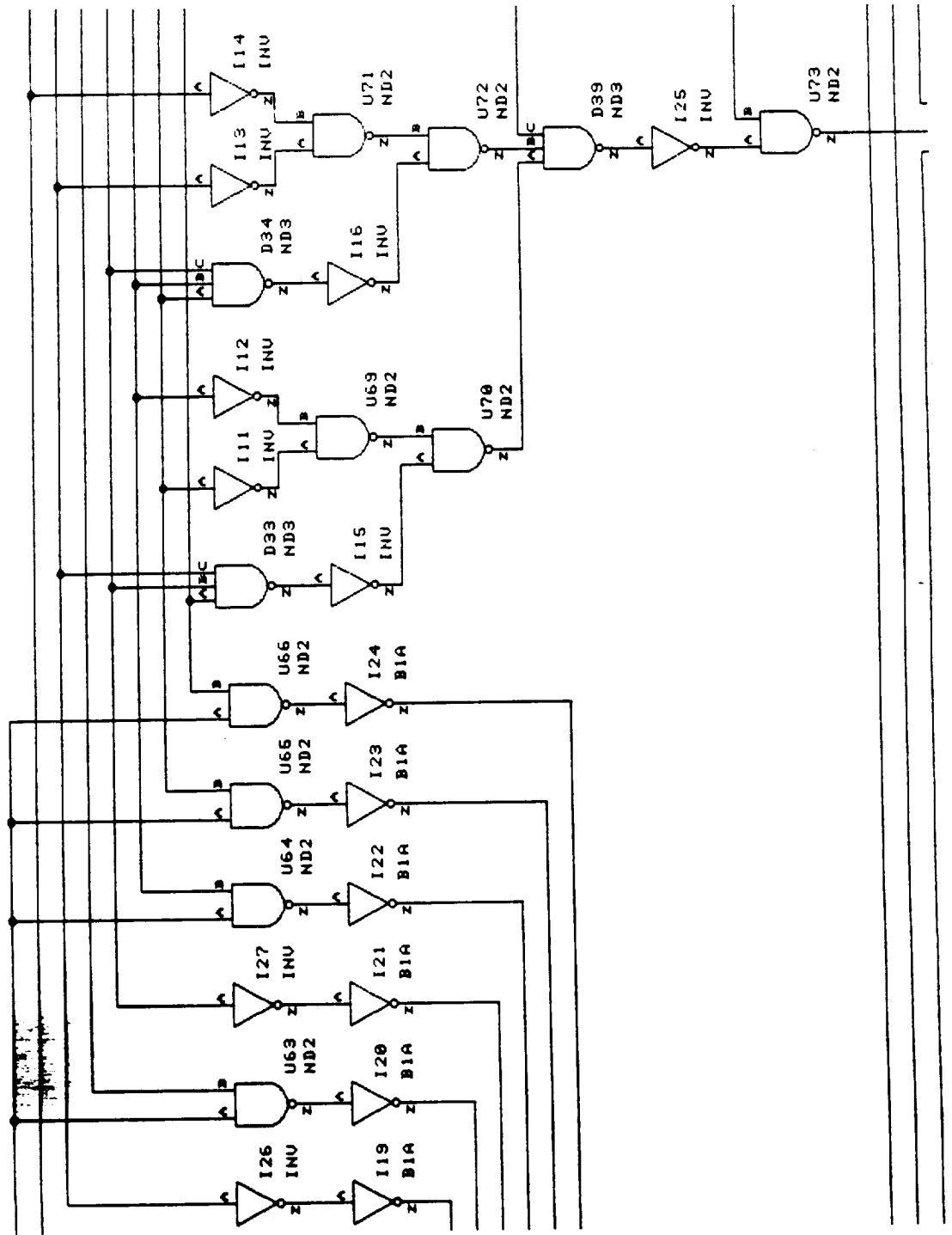


Figure 46

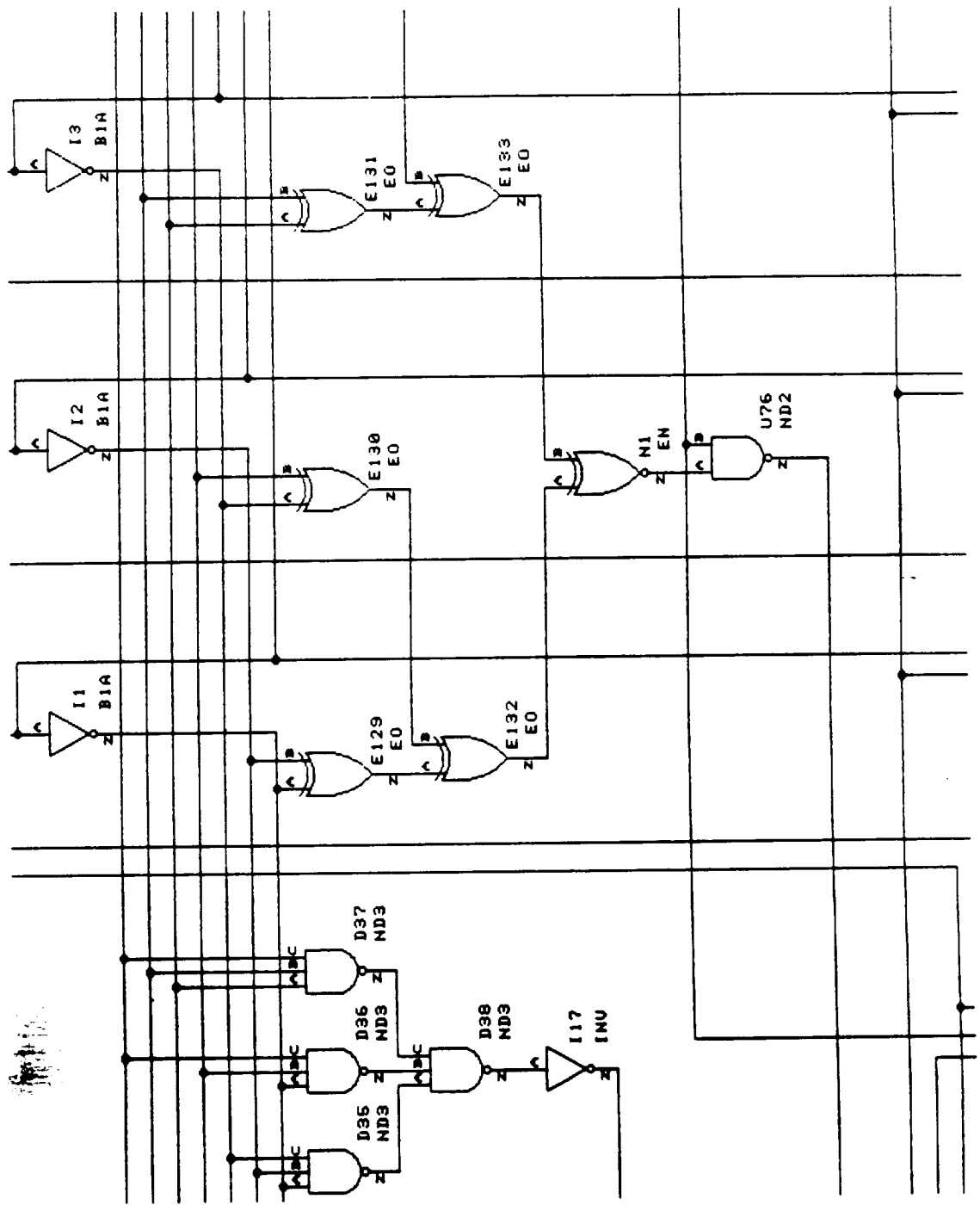


Figure 47

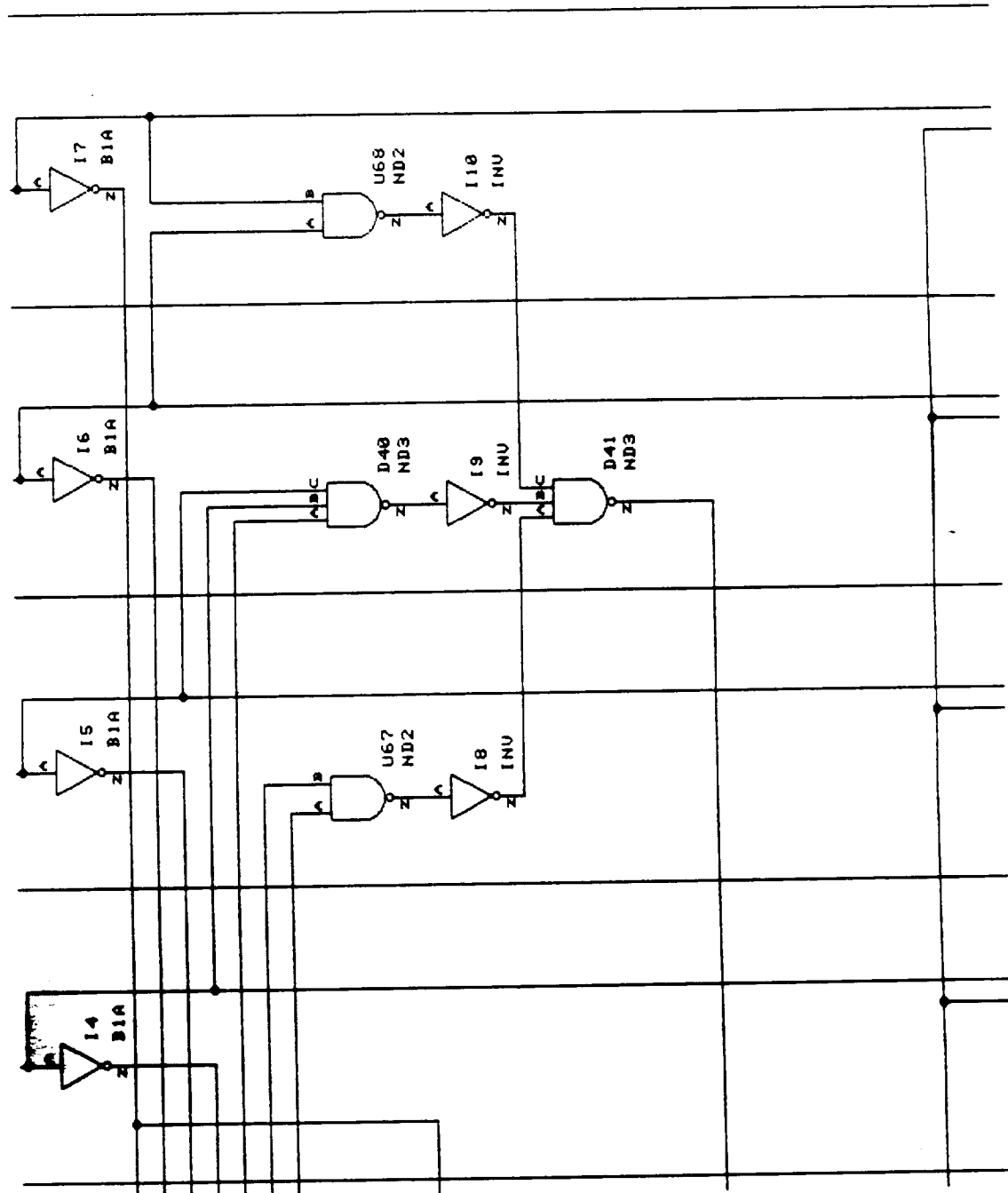


Figure 48

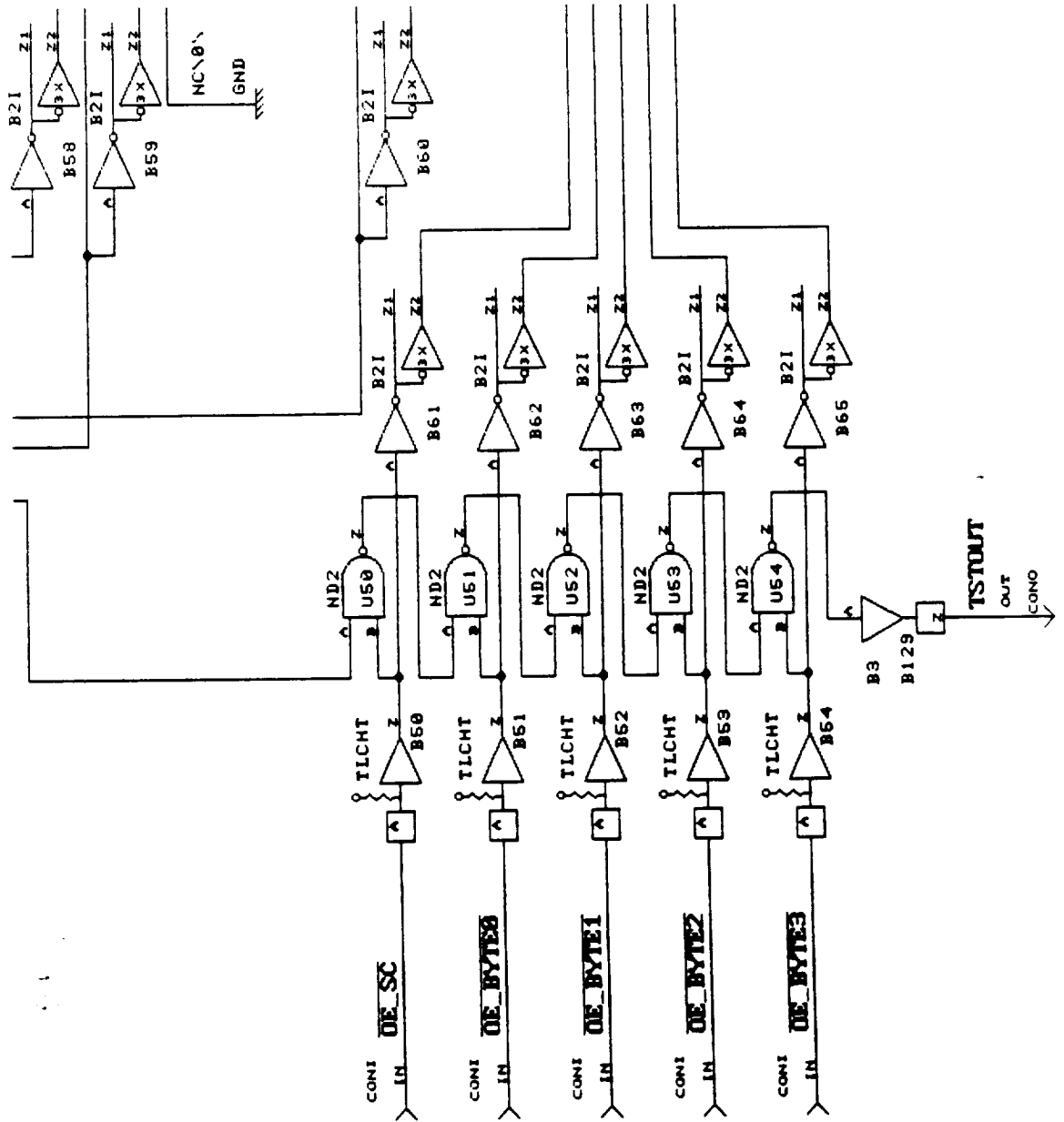


Figure 49

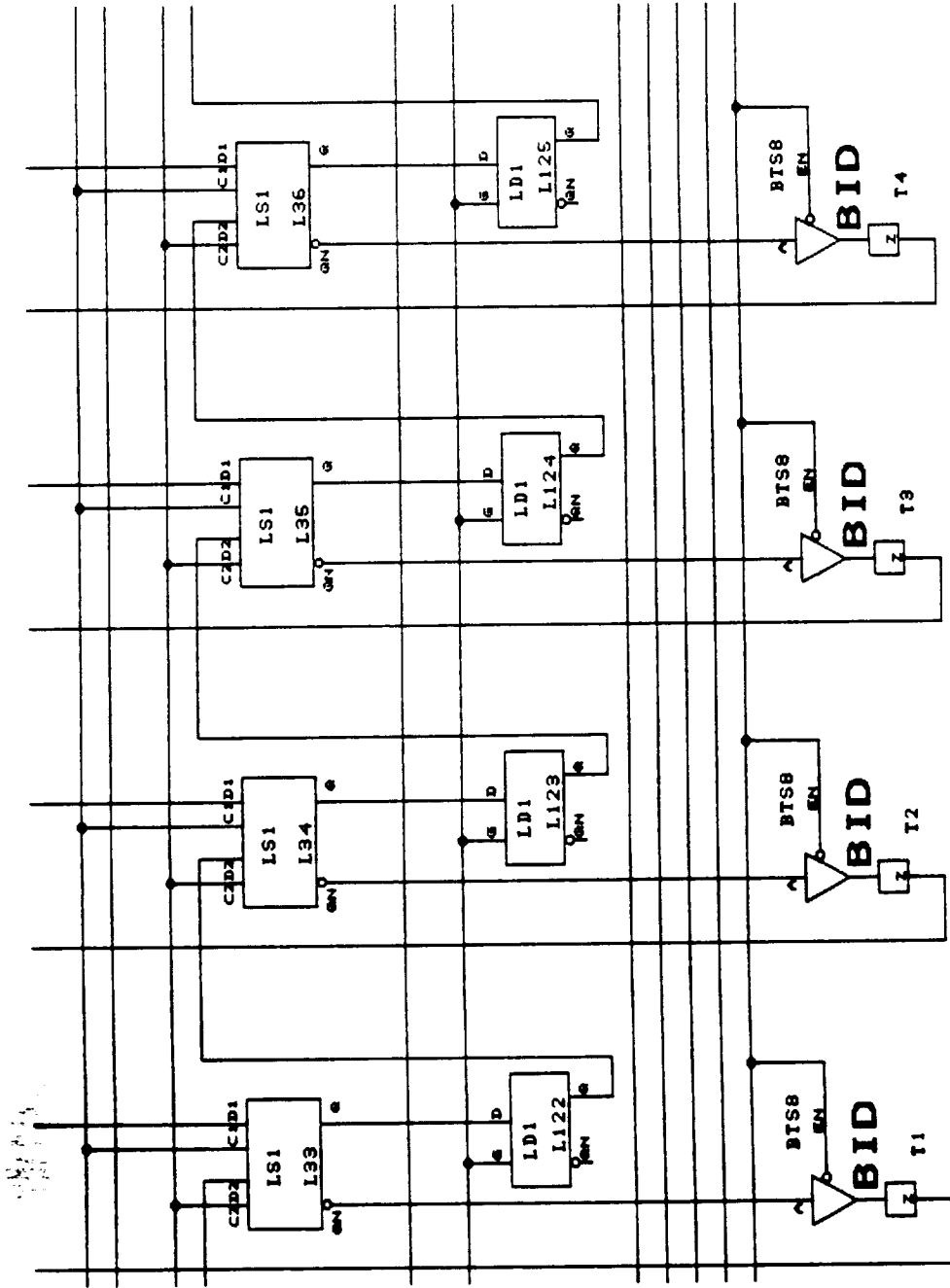


Figure 50

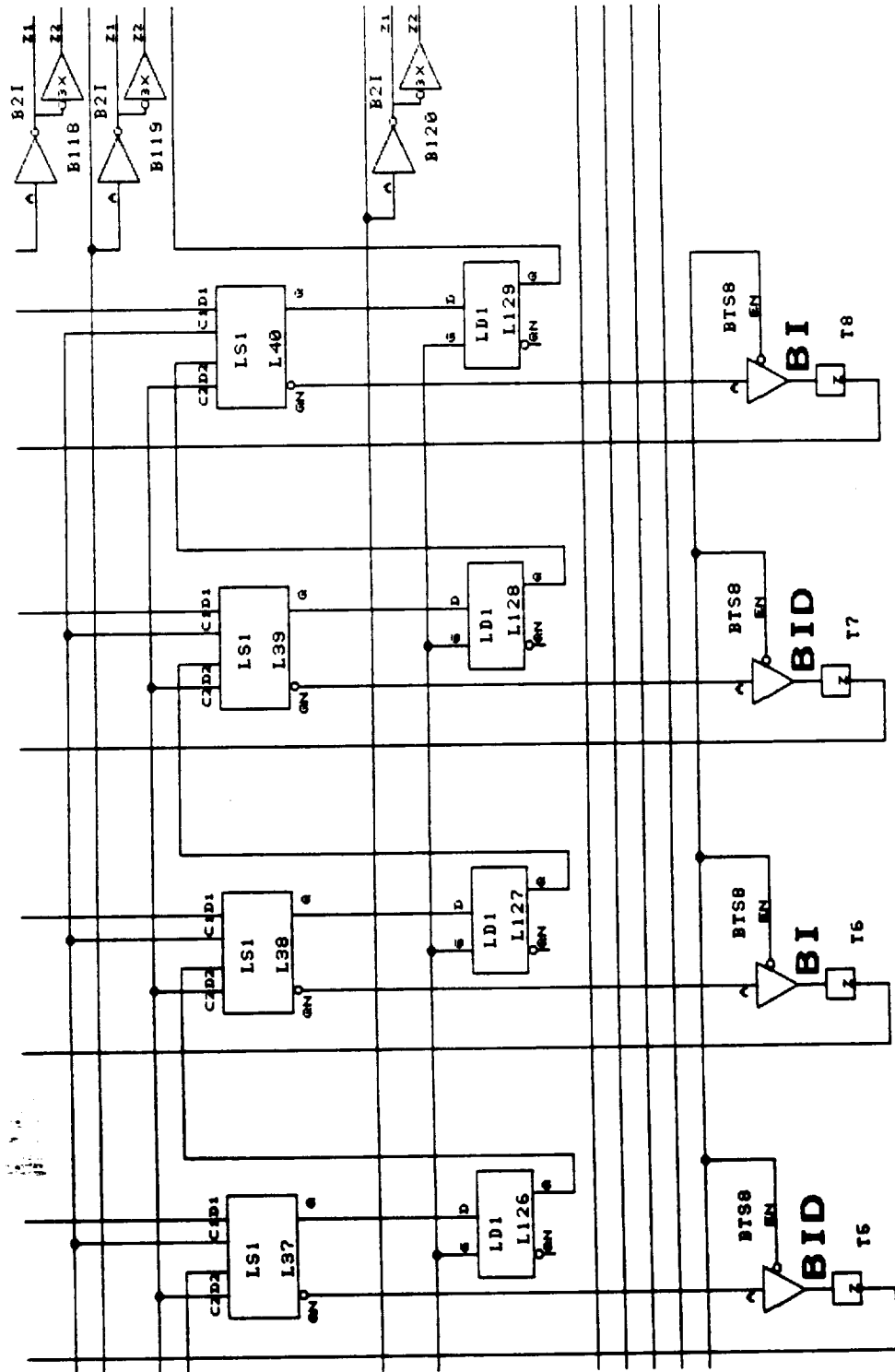


Figure 51

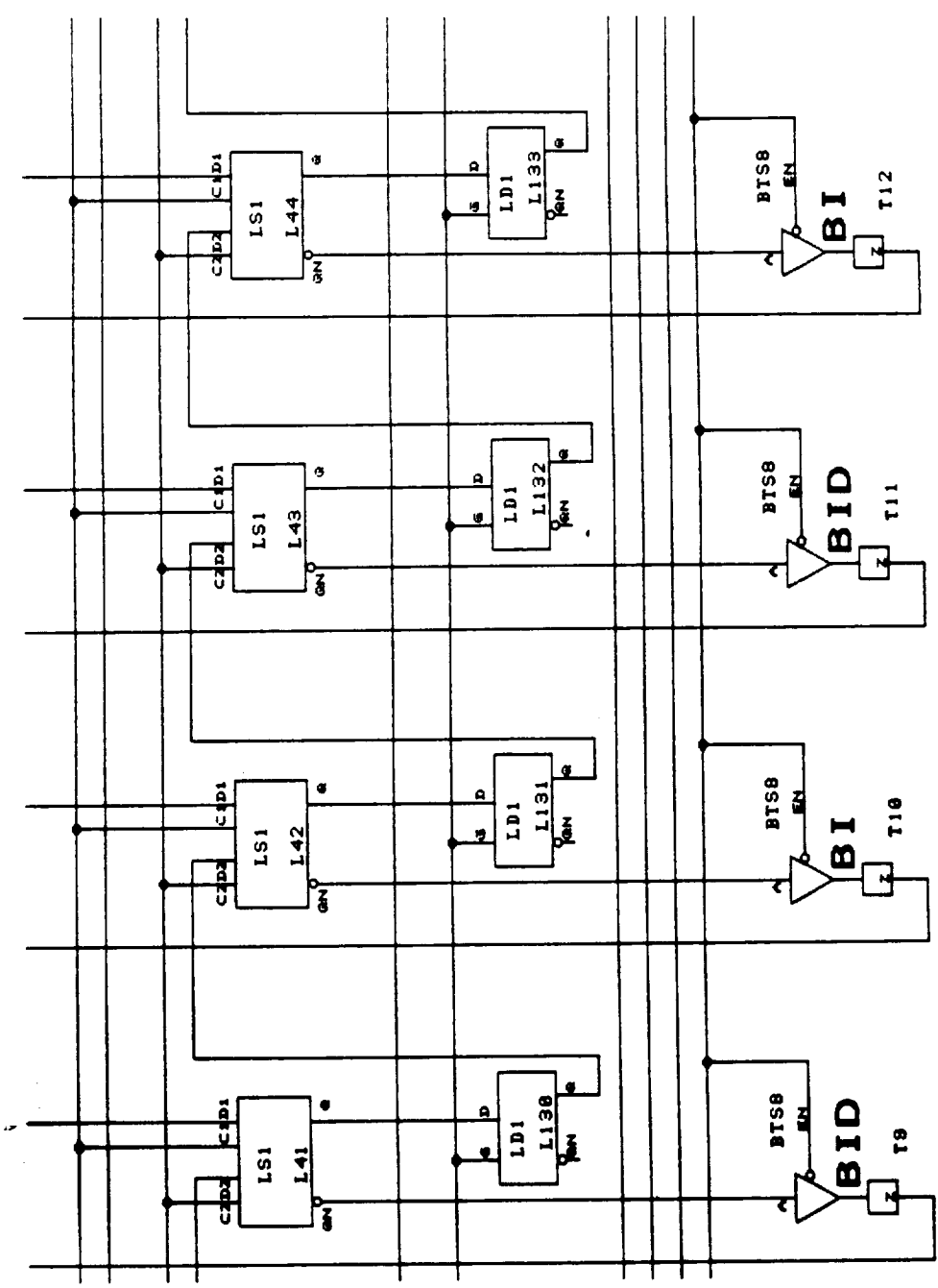


Figure 52

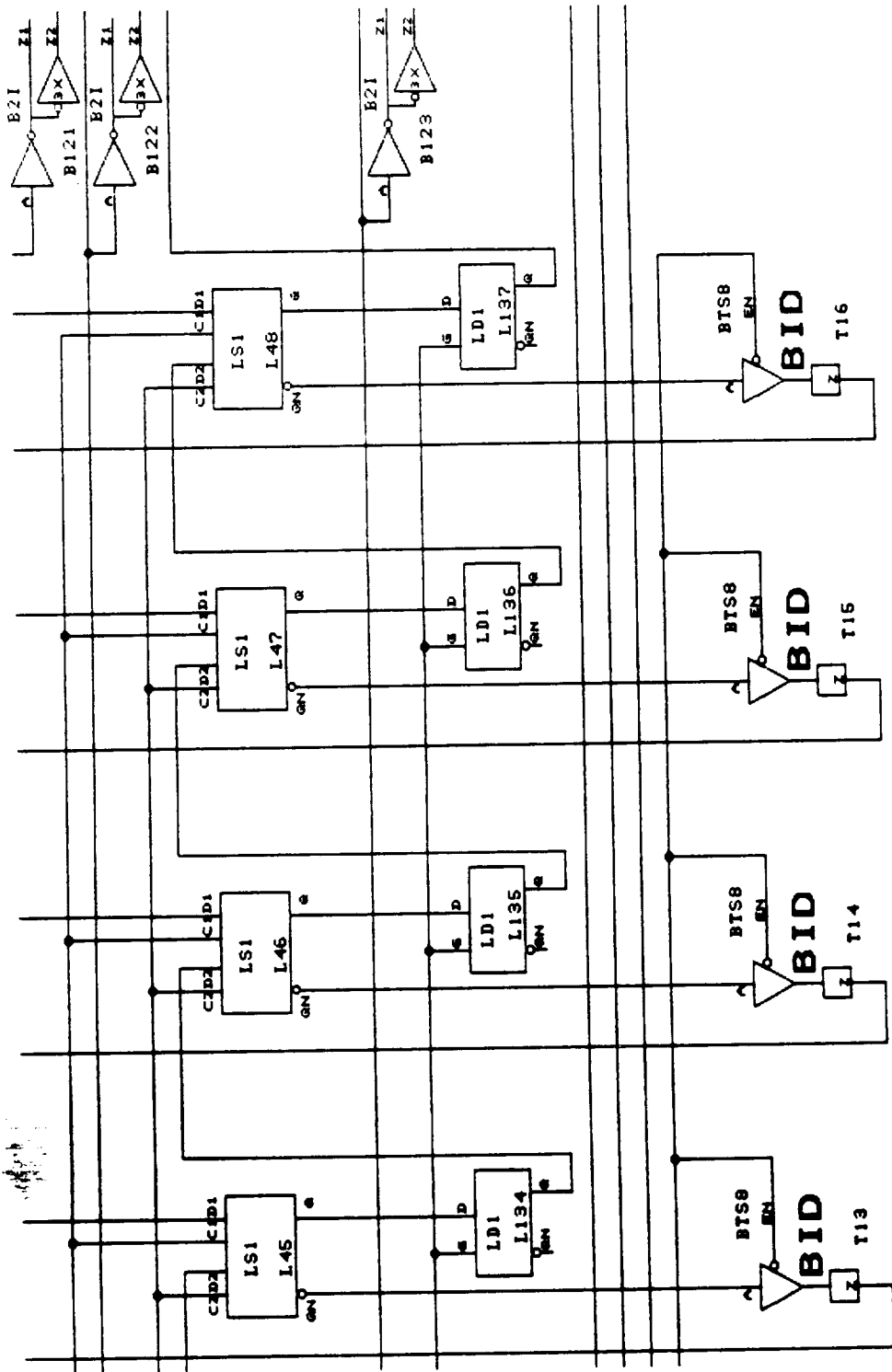


Figure 53

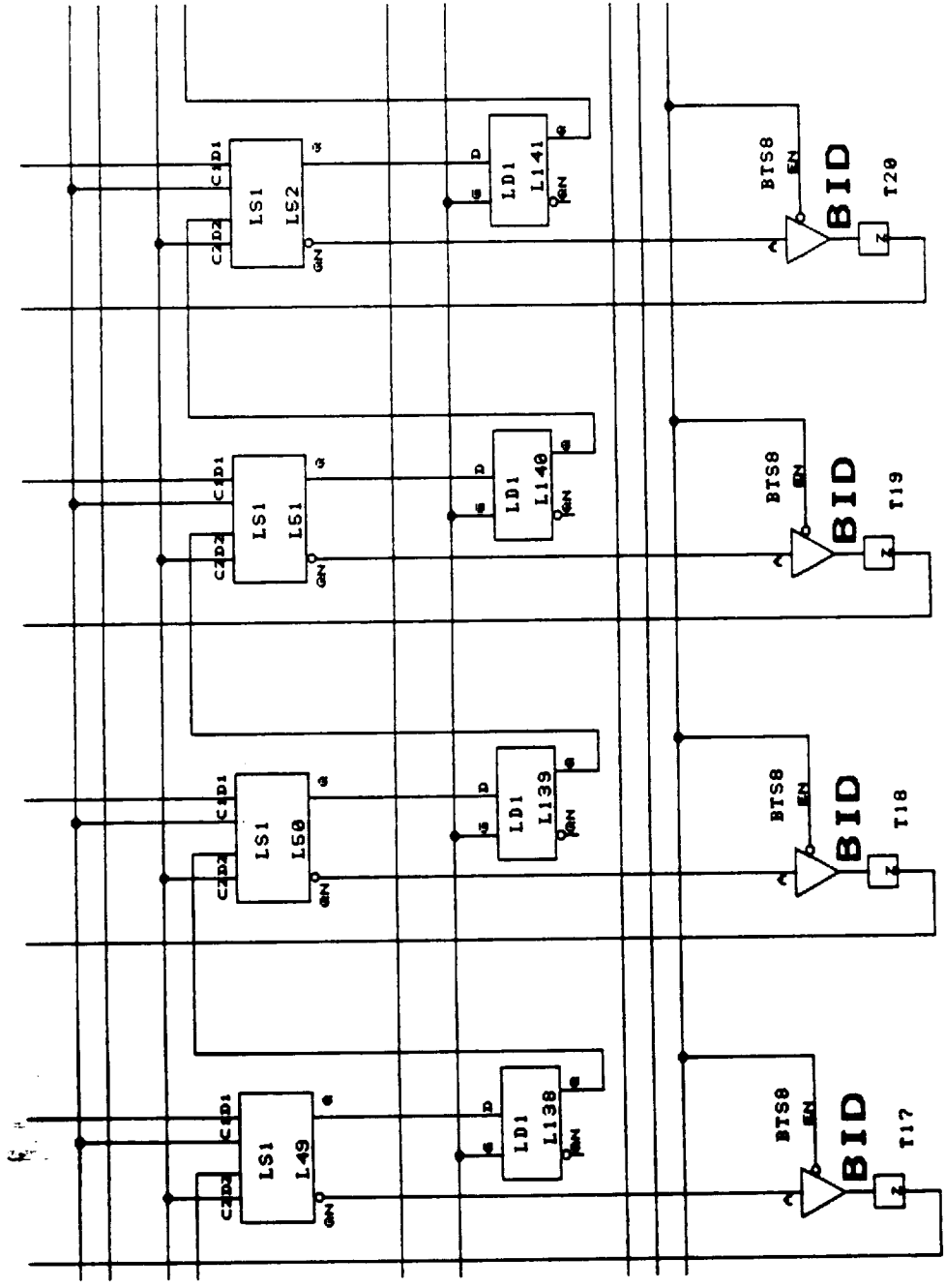
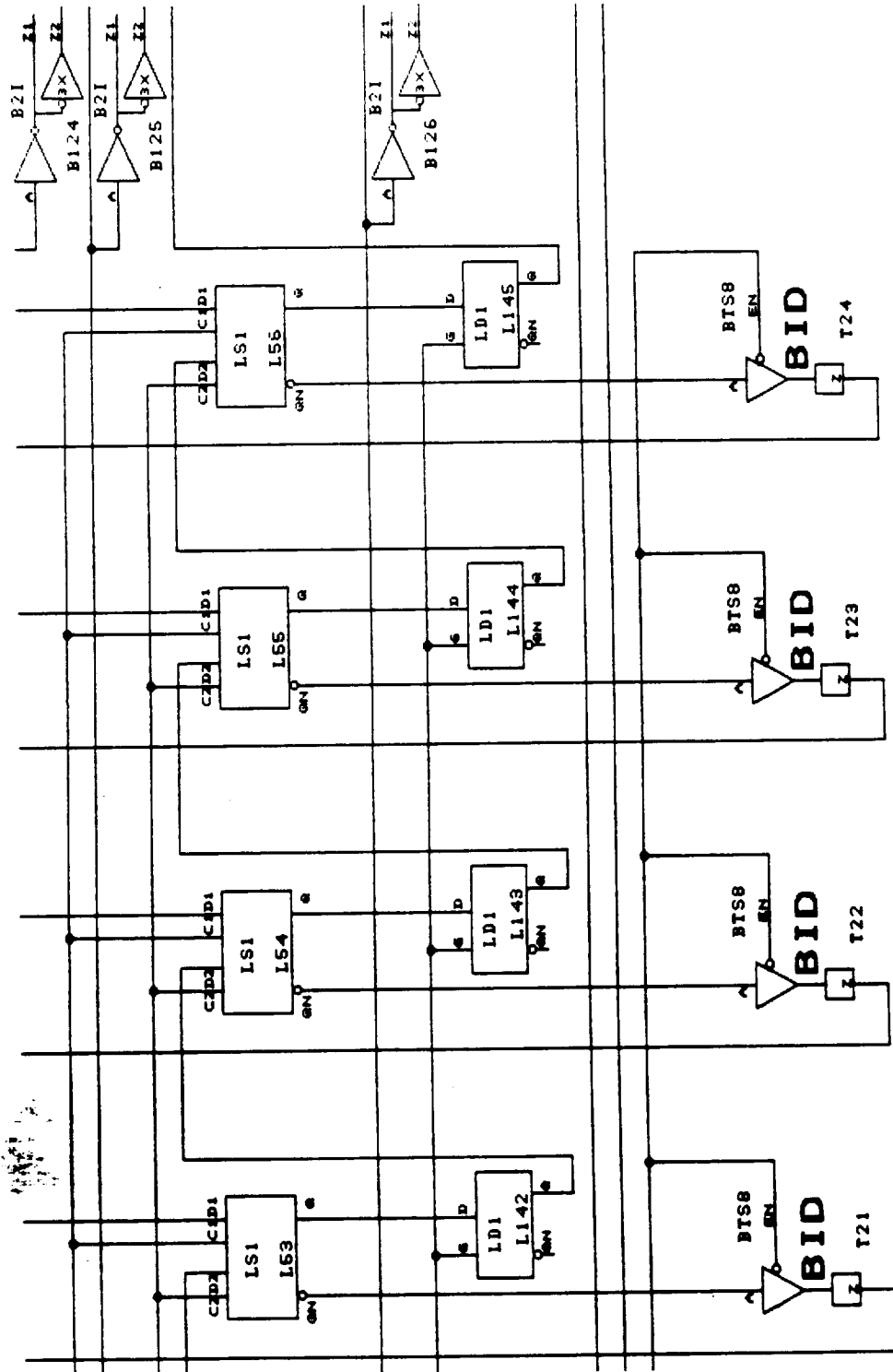


Figure 54



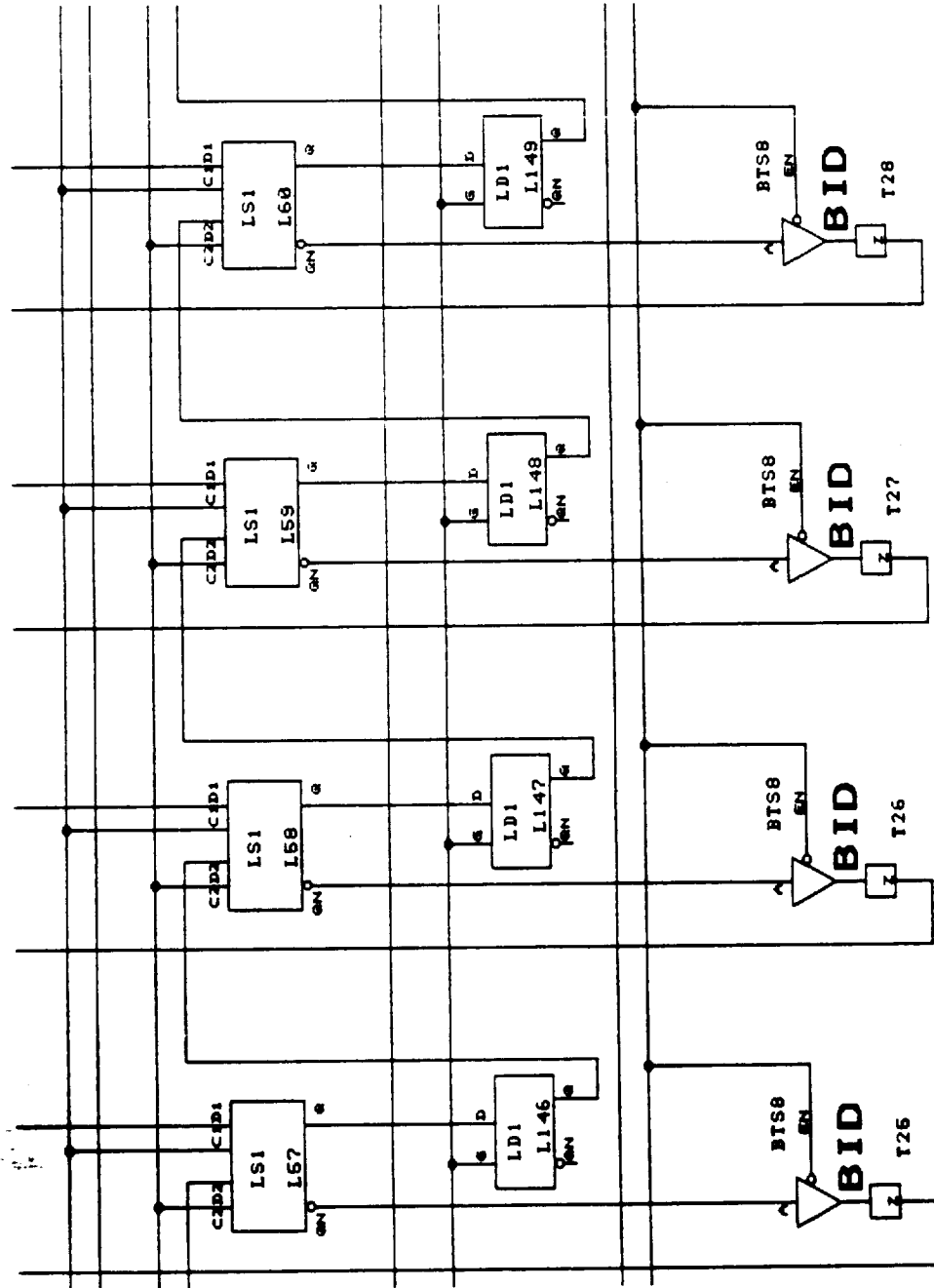


Figure 56

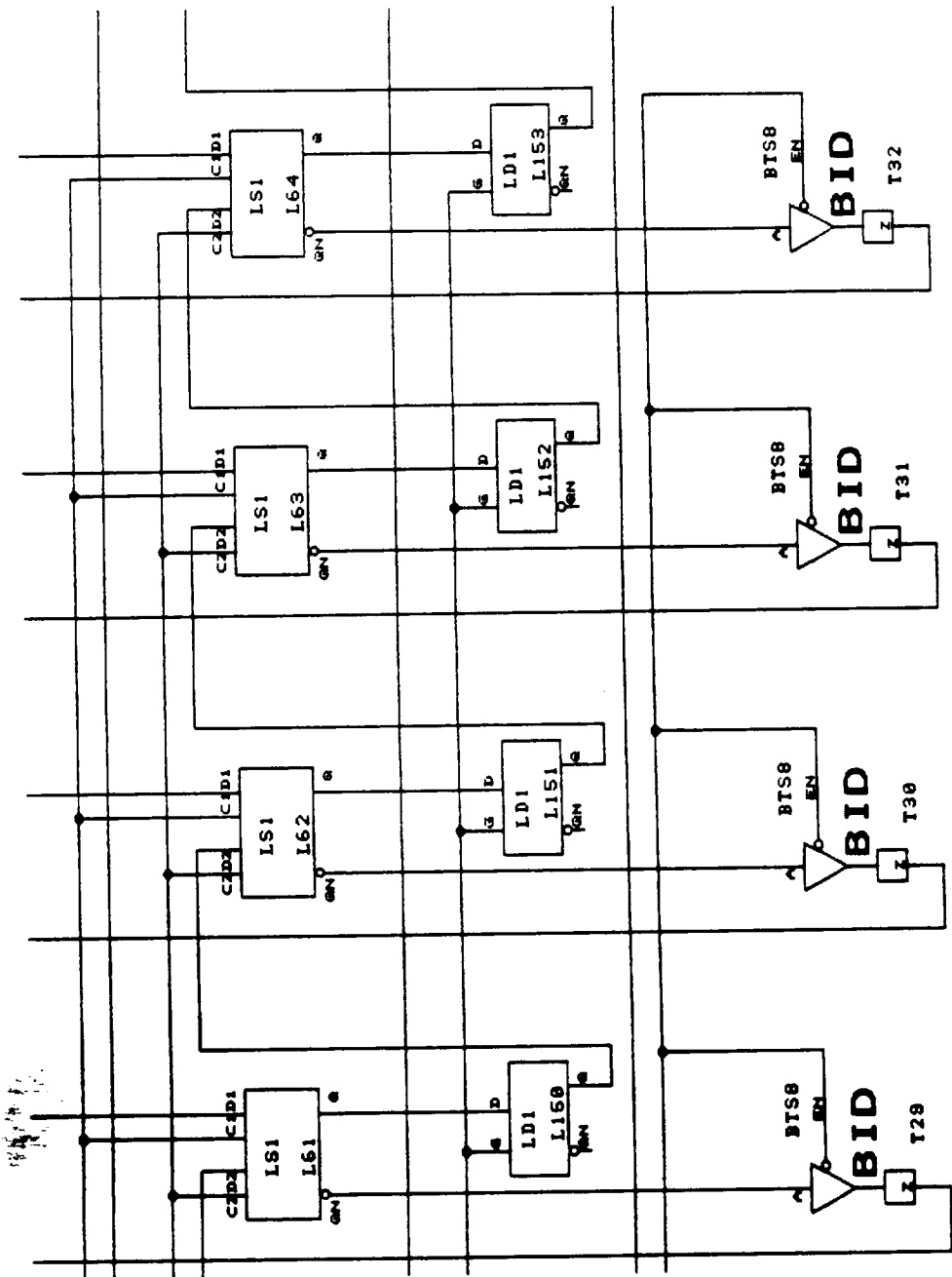


Figure 57

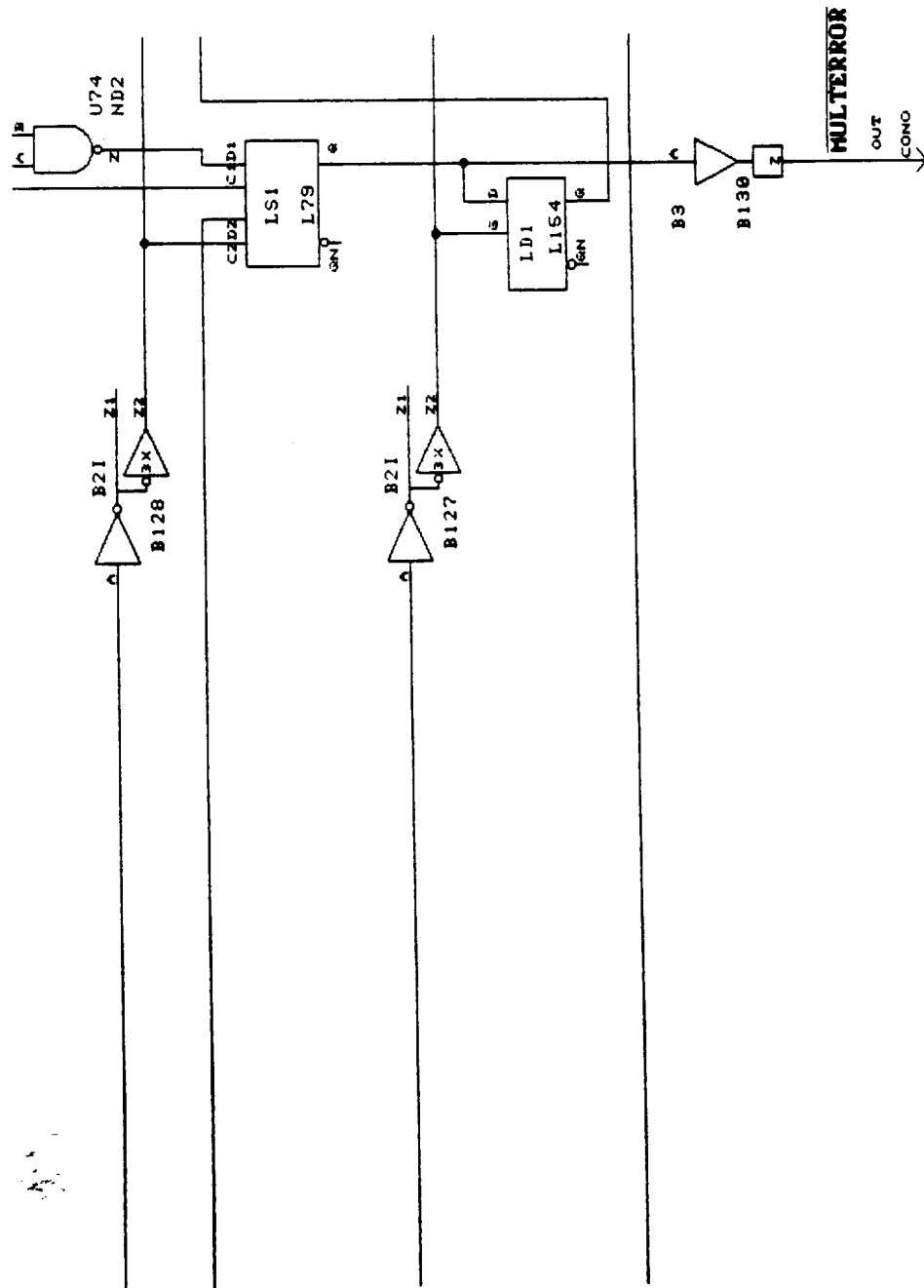


Figure 58

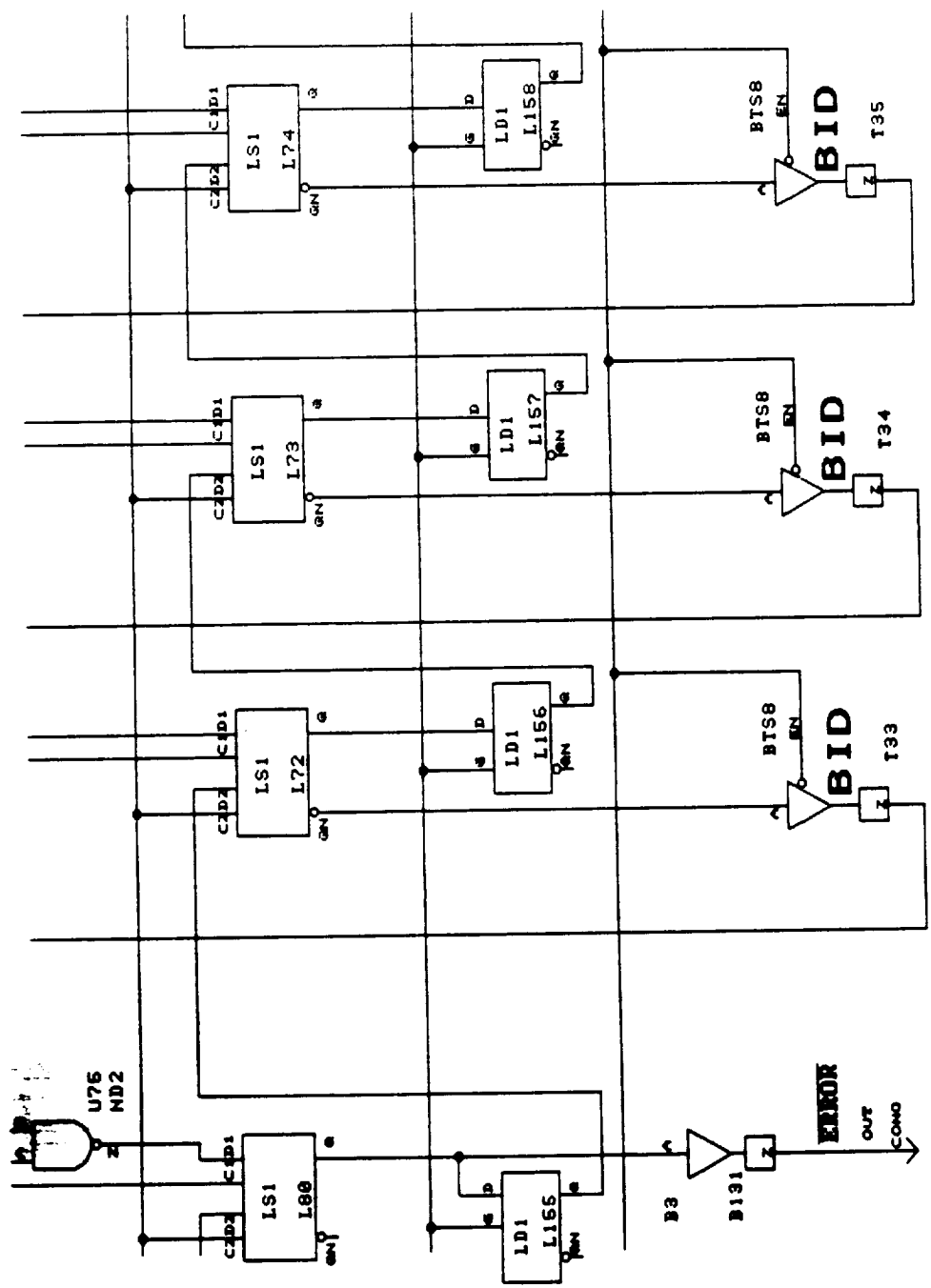
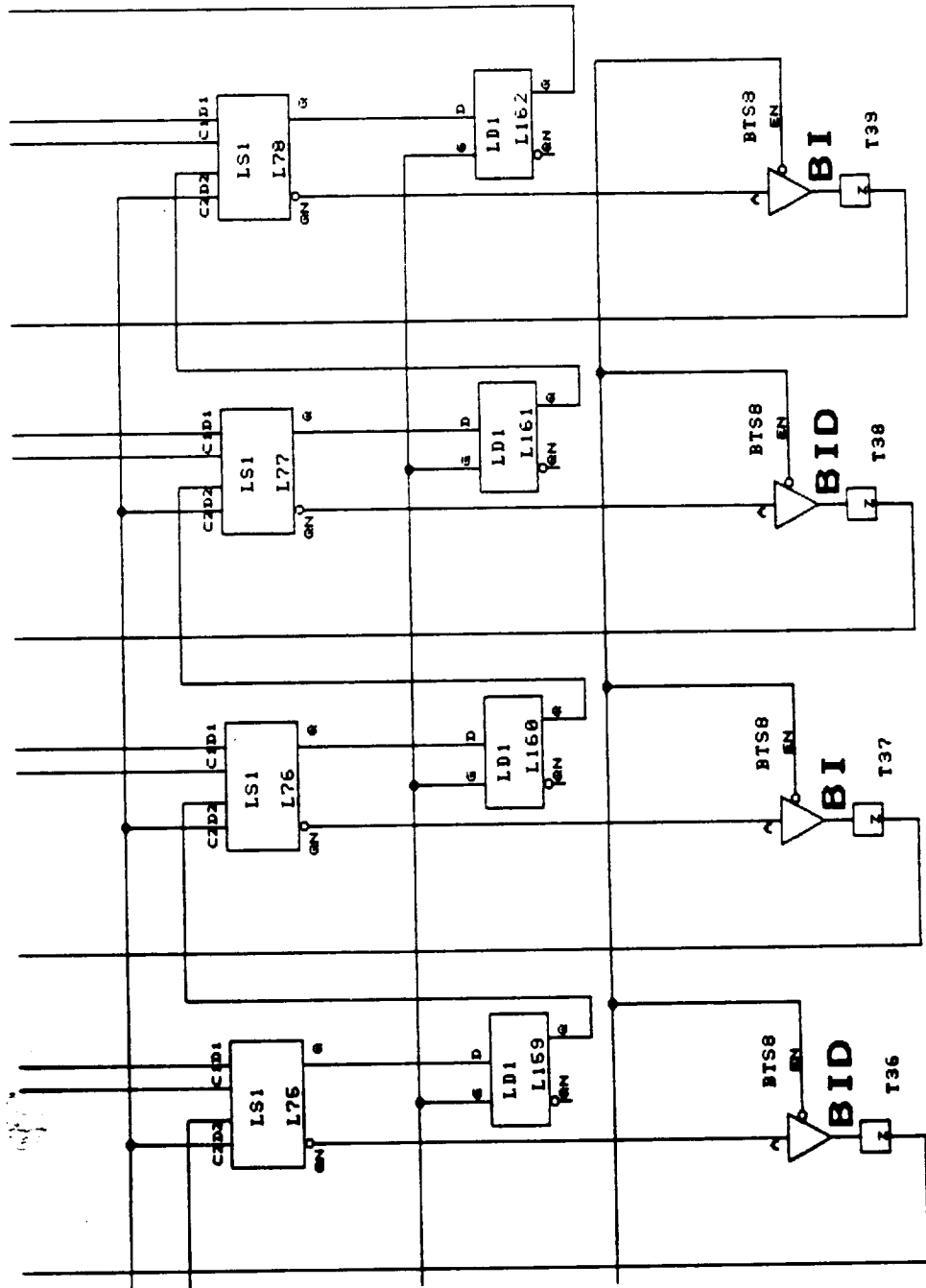


Figure 59

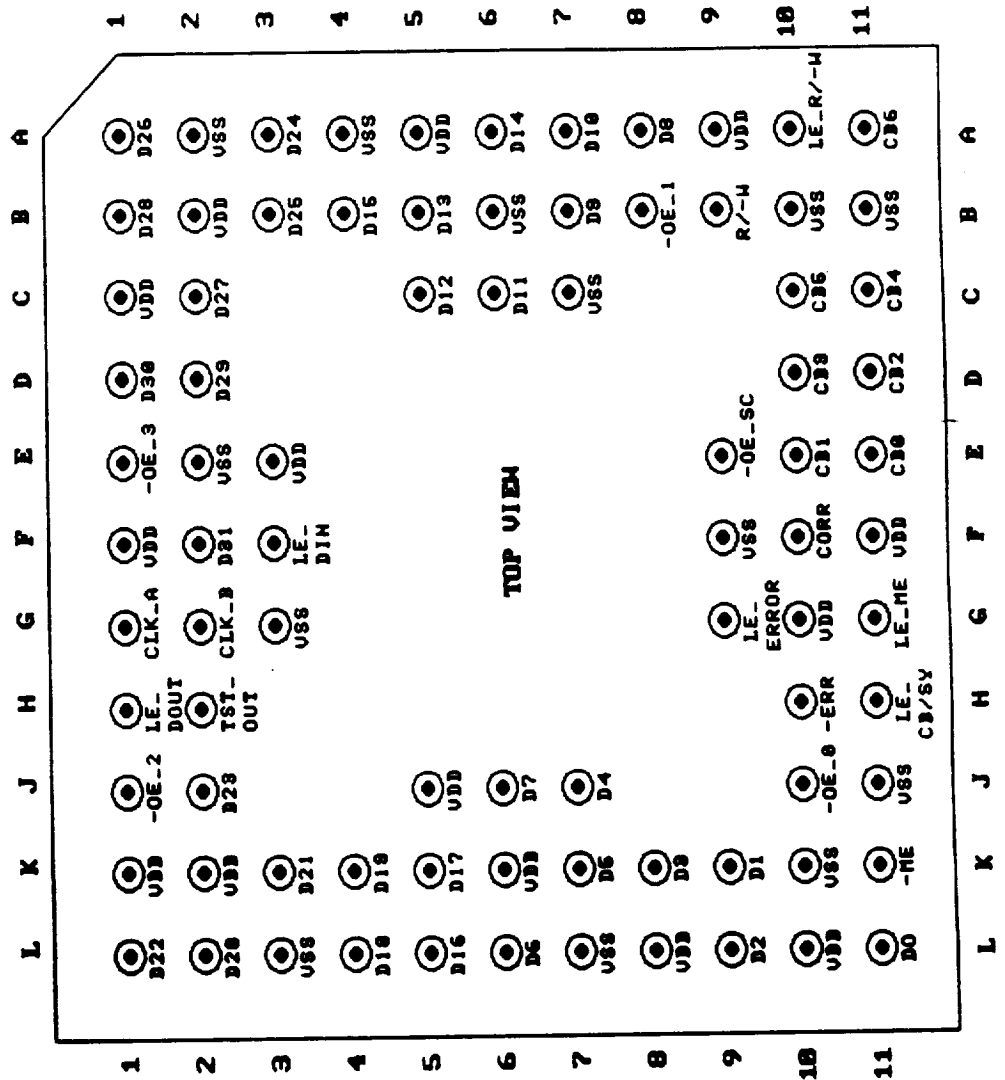


APR 12 1989

EDC.DMG

Figure 60

PIN ASSIGNMENT



PIN DESCRIPTION

PIN NAME	I/O	DESCRIPTION
DATA ₀₋₃₁	I/O	32 bidirectional data lines. They provide input to the Data Input Latch and receive output from the Data Output Latch. DATA is the LSB; DATA ₃₁ is the MSB.
CB/SY ₀₋₄	I/O	Seven bidirectional checkbit lines. They provide input to the Check Bit Input Latch and receive output from the Check Bit / Syndrome Output Latch.
R/W	I	Selects between Read and Write Mode. When LOW (Write Mode), check bits are generated from a set of 32 data bits. When HIGH (Read Mode), the device detects single and double errors, and generates syndrome bits based upon the contents of the Data Input Latch. In Correct mode (CORRECT-HIGH) single bit errors are also automatically corrected, with the corrected data placed at the inputs of the Data Output Latches.
LE R/W	I	Latch Enable for R/W signal. This signal is kept HIGH during system modes (Read, Write, Pass Thru). When HIGH, the R/W latch is transparent for the R/W signal. During Self-Test this signal must be LOW.
LE DATA IN	I	Latch Enable for Data Input Latch. Controls latching of the input data. Data Input Latch and Check Bit / Syndrome Input Latch are latched to their previous state when LOW. When HIGH, the Data Input Latch and Check Bit / Syndrome Input Latch follow the input data and input check bits.
LE DATA OUT	I	Latch Enable for Data Output Latch. Controls latching of the output data.
LE CB/SY	I	Latch Enable for CB/SY Output Latch. Controls latching of the CB/SY output.
LE ERROR	I	Latch Enable for ERROR Output Latch.
LE HE	I	Latch Enable for HULT-ERROR Output Latch.
CORRECT	I	The correct input which, when HIGH, allows the correction network to correct any single bit error in the Data Input Latch (by complementing the bit-in-error) before putting it into the Data Output Latch. When LOW, the device will drive data directly from the Data Input Latch to the Data Output Latch without correction.

PIN DESCRIPTION (cont'd)

OE BYTE ₀₋₃	1	Output Enable for Byte 0, 1, 2, and 3. Control the three-state output buffers for each of the four bytes of the Data Output Latch. When IOM, they enable the output buffer of the Data Output Latch. When HIGH, they force the Data Output Latch buffer into the high impedance mode.
OE SC	1	Output Enable for Syndrome or Check Bits. In the HIGH condition, the CB/SY outputs are in the high impedance state. When IOM, all CB/SY output lines are enabled.
ERROR	0	In the Detect or Correct Mode, this output will go LOW if one or more data or check bits contain an error. When HIGH, no errors have been detected. This pin is forced HIGH in Write Mode.
MULTI_ERROR	0	In the Detect or Correct Mode, this output will go LOW if two or more errors have been detected. A HIGH level indicates that either one or no errors have been detected. This pin is forced HIGH in Write Mode.
CIK_A, CIK_B	1	Shift clocks for Built-In Self-Test (BIST). These non-overlapping clocks are used to shift data through the Linear Feedback Shift Registers (LFSRs) in Self-Test Mode.
TSOUT	0	Test pin for parametric test.

EDAC PROPAGATION DELAYS						
Mode	WRITE		READ			
From input	De-3:	LE_CB/SY	De-3:		CB/SYe-4	
To output	CB/SYe-4 (nsec)		ME/ (nsec)	ERR/ (nsec)	ME/ (nsec)	ERR/ (nsec)
Output Load	0pf		47pf	47pf	47pf	47pf
Part #						
1	54		68	72	57	68
4	54		68	72	57	61
5	55		70	73	60	62
6	54		68	72	57	61
7	56		71	76	61	64
8	55		70	75	60	63
9	54		68	73	57	61
10	53		65	69	55	58

ERROR DETECTION AND CORRECTION UNIT WITH BUILT-IN SELF
TEST CAPABILITY FOR SPACECRAFT APPLICATIONS

NAS7-1028

PART 4

APPENDIX F: PROGRAM LISTINGS

F.1 Design Verification

- F.1.1 EDAC.C: Random Test.....F-1
- F.1.2 PT-TEST.C: Parity Tree Test.....F-13
- F.1.3 ED-TEST.C: Error Detector Test.....F-24
- F.1.4 EC-TEST.C: Error Corrector Test.....F-28

F.2 Fault Simulation

- F.2.1 FAULT.C: Combinational Logic.....F-32
- F.2.2 FAULT1.C: Scannable Latches.....F-48

```

/*
File: EDAC.C
Date: March 22, 1989

This program is designed to work with our EDAC design and with two
commercial EDAC chips (i.e., SN 74ALS632 from Texas Instruments
and IDT 49C460 from Integrated Device Technology). The program
enables you to write any data to the EDAC and to read the response
from the EDAC. It also allows you to inject faults into the written
data to determine the circuits behavior in regard to faulty data.
Last, but not least, the program has a feature that enables you to
test the EDAC automatically (Function key F8-Test).

*/

#include <stdio.h>
#include <graph.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>

const long black=0;
const long blue=1;
const long green=2;
const long cyan=3;
const long red=4;
const long magenta=5;
const long white=7;
const long yellow=14;
const long bright=15;

const false=0;
const true=1;

char *line="
";
char buffer[255];
char *text[4]={" ", "Spaceborne_EDAC", "TI-EDAC", "IDT-EDAC"};

int control1=0x00;
int control2=0xE7;
int write_mode=0xA7;
int read_mode=0xE7;
int CLK_A=0x08;
int CLK_B=0x10;
int mode=3;
int byte[4], data[4], goodbye[4], goodcb, check, cb, syndr, error, multerror, cbit;
int error_bit, me_bit, fdo, control_in, looping, error_count, ti;
int single, doubl, endloop;
char binbyte[8];
int finish=0;
long count;
int Power_of_2(int x);

main()
{
int i, j, key, key1;
time_t start, stop;
cb=0;
device_screen();

```

```

edac_screen();
initialize();
byte[0]=byte[1]=byte[2]=byte[3]=0;
display_data_write();
write_all();
display_mode();
cb=check;
display_cb(9,cb);
display_error();
do
{
  if (kbhit())
  {
    key=getch();
    if (key==0)
    {
      key=getch();
      switch (key)
      {
        case 59:
          write_all();
          display_mode();
          clr_data(12);
          cb=check;
          display_cb(9,cb);
          display_error(16,5);
          break;
        case 60:
          read_all();
          display_mode();
          display_data_read();
          display_cb(12,syindr);
          display_error(16,5);
          break;
        case 61:
          for (i=0;i<=3;i++)
          {
            for (j=0;j<=7;j++)
            {
              byte[i]=Power_of_2(j);
              display_data_write();
              write_all();
              display_mode();
              clr_data(12);
              cb=check;
              display_cb(9,cb);
              display_error(16,5);
              getch();
              byte[i]=0;
            }
          }
          break;
        case 62:
          data_input();
          display_data_write(10,1);
          break;
        case 63:
          checkbit_input();
          display_cb(9,cb);
          break;
      }
    }
  }
}

```



```

case 64:
    write_err();
    write_out();
    write_in();
    read_in();
    display_mode();
    display_data_read();
    display_cb(12, syndr);
    display_error(16, 5);
    do
    {
        outp(773, control2 ; CLK_B);
        outp(773, control2);
        probe();
        _settextposition(24, 1);
        _outtext("B");
        read_in();
        display_mode();
        display_data_read();
        display_cb(12, syndr);
        display_error(16, 5);
        getch();
        outp(773, control2 ; CLK_A);
        outp(773, control2);
        /*probe();
        _settextposition(24, 1);
        _outtext("A");
        read_in();
        display_mode();
        display_data_read();
        display_cb(12, syndr);
        display_error(16, 5);
        getch();*/
    }
    while (!kbhit());
    break;
case 65:
    probe();
    break;
case 66:
    clr_lines(11, 24, blue);
    test_screen();
    single=doubl=true;
    endloop=false;
    do
    {
        if (kbhit())
        {
            key=getch();
            if (key==0)
            {
                key=getch();
                if (key==59)
                {
                    time(&start);
                    srand(start);
                    count=error_count=0;
                    looping=true;
                    do
                    {

```

```

count+=1;
display_count(15,5);
for (i=0;i<=3;i++)
{
    byte[i]=goodbyte[i]=rand()&0xff;
}
display_data_write();
write_all();
cb=goodcb=check;
display_cb(9,cb);
no_error();
if (single) single_error();
if (doubl) double_error();
display_error_count(15,50);
if (kbhit())
{
    key=getch();
    if (key==0)
    {
        key=getch();
        if (key==67)
        {
            looping=false;
        }
        if (key==69)
        {
            looping=false;
            endloop=true;
        }
    }
}
while (looping);
}
if (key==62)
{
    single='single';
    display_test(3);
}
if (key==63)
{
    doubl='doubl';
    display_test(3);
}
if (key==68)
{
    endloop=true;
    break;
}
}
}
}
while (!endloop);
edac_screen();
byte[0]=byte[1]=byte[2]=byte[3]=0;
display_data_write();
display_mode();
write_all();
cb=check;
display_cb(9,cb);

```

```

        display_error();
        break;
    case 67:
        do
        {
            outp(773,control2 ; CLK_B);
            outp(773,control2);
            outp(773,control2 ; CLK_A);
            outp(773,control2);
        }
        while (!kbhit());
        getch();
        break;
    case 68:
        finish=true;
        break;
    }
}
}
}
while (!finish);
_clearscreen(_GCLEARSCREEN);
}

data_input(void)
{
    int i,x,y;
    _setbkcolor(cyan);
    _settextcolor(blue);
    for (i=18; i<=23; i++)
    {
        _settextposition(i,5);
        _outtext(" ");
    }
    _settextposition(19,6);
    sprintf (buffer,"Byte 0: "); _outtext(buffer); scanf ("%x",&byte[0]);
    _settextposition(20,6);
    sprintf (buffer,"Byte 1: "); _outtext(buffer); scanf ("%x",&byte[1]);
    _settextposition(21,6);
    sprintf (buffer,"Byte 2: "); _outtext(buffer); scanf ("%x",&byte[2]);
    _settextposition(22,6);
    sprintf (buffer,"Byte 3: "); _outtext(buffer); scanf ("%x",&byte[3]);
}

checkbit_input(void)
{
    int i;
    _setbkcolor(cyan);
    _settextcolor(blue);
    for (i=20; i<=22; i++)
    {
        _settextposition(i,40);
        _outtext(" ");
    }
    _settextposition(21,40);
    sprintf (buffer," Checkbits: "); _outtext(buffer); scanf ("%x",&cb);
    check=cb;
}

display_byte(int x, int y, int dbyte)

```

c-4

```

    {
    int i,j;
    _setbkcolor(blue);
    _settextcolor(yellow);
    _settextposition(x-1,y+3);
    sprintf (buffer,"%02x",dbyte);
    _outtext(buffer);
    _setbkcolor(white);
    _settextcolor(black);
    conv_byte_to_bin(dbyte);
    j=0;
    for (i=7; i>=0; i--)
        {
            j+=sprintf(buffer+j,"%d",binbyte[i]);
        }
    _settextposition(x,y);
    _outtext(buffer);
    }

display_cb(int x,int dbyte)
{
    int i,j;
    _setbkcolor(blue);
    _settextcolor(yellow);
    _settextposition(x-1,63);
    sprintf (buffer,"%02x",dbyte);
    _outtext(buffer);
    _setbkcolor(white);
    _settextcolor(black);
    conv_byte_to_bin(dbyte);
    j=0;
    for (i=6; i>=0; i--)
        {
            j+=sprintf(buffer+j,"%d",binbyte[i]);
        }
    _settextposition(x,60);
    _outtext(buffer);
}

display_data_write(void)
{
    int i,j;
    int x,y;
    x=9;
    for (j=0; j<=3; j++)
        {
            y=5+j*9;
            display_byte(x,y,byte[3-j]);
        }
}

display_data_read(void)
{
    int j;
    int x,y;
    x=12;
    for (j=0; j<=3; j++)
        {
            y=5+j*9;
            display_byte(x,y,data[3-j]);
        }
}

```

ORIGINAL PAGE IS
OF POOR QUALITY

```

    }
}

display_error(void)
{
    _setbkcolor(white);
    _settextcolor(red);
    _settextposition(15,60);
    sprintf (buffer,"%d",!error_bit);
    _outtext(buffer);
    _settextposition(15,70);
    sprintf (buffer,"%d",!me_bit);
    _outtext(buffer);
}

display_mode(void)
{
    char *modus;
    if (control2==write_mode)
        modus=" WRITE ";
    else
        modus=" READ ";
    _setbkcolor(white);
    _settextcolor(red);
    _settextposition(15,5);
    _outtext(modus);
}

display_count(int x, int y)
{
    _settextcolor(black);
    _setbkcolor(white);
    _settextposition(x,y);
    sprintf(buffer,"%6ld",count);
    _outtext(buffer);
}

display_error_count(int x, int y)
{
    _settextcolor(black);
    _setbkcolor(white);
    _settextposition(x,y);
    sprintf(buffer,"%5d",error_count);
    _outtext(buffer);
}

display_test(int x)
{
    _settextcolor(white);
    _setbkcolor(blue);
    _settextposition(x,5);
    _outtext("Testing for: Errors");
    _settextcolor(black);
    _setbkcolor(white);
    _settextposition(x,20);
    _outtext(" No ");
    if (single)
    {
        _settextcolor(black);
        _setbkcolor(white);
    }
}

```

```

        _settextposition(x,26);
        _outtext(" Single ");
    }
    else
    {
        _setbkcolor(blue);
        _settextposition(x,26);
        _outtext(" ");
    }
    if (doubl)
    {
        _settextcolor(black);
        _setbkcolor(white);
        _settextposition(x,36);
        _outtext(" Double ");
    }
    else
    {
        _setbkcolor(blue);
        _settextposition(x,36);
        _outtext(" ");
    }
}

display_probe(int probel,int probe2)
{
    int i,j;
    char *field=" ";
    _setbkcolor(white);
    for (i=17;i<=23;i++)
    {
        _settextposition(i,50);
        _outtext(field);
    }
    _settextcolor(black);
    _settextposition(19,50);
    _outtext(" 1 2 3 4 5 6 7 8");
    _settextposition(22,50);
    _outtext(" 9 10 11 12 13 14 15 16");
    conv_byte_to_bin(probel);
    _settextcolor(red);
    j=0;
    for (i=0;i<=7;i++)
    {
        j+=sprintf(buffer+j," %d ",binbyte[i]);
    }
    _settextposition(18,51);
    _outtext(buffer);
    conv_byte_to_bin(probe2);
    j=0;
    for (i=0;i<=7;i++)
    {
        j+=sprintf(buffer+j," %d ",binbyte[i]);
    }
    _settextposition(21,51);
    _outtext(buffer);
}

initialize(void)

```

```

/* initializes the EDAC */
{
int i;
outp(772,control1); /* set port 772 to 0 */
outp(773,control2); /* set port 773 to E7hex */
mode=3; /* set microsimulator to */
outp(774,mode); /* mode 3 */
for (i=0; i<3000; i++)
{
outp(772,control1); /* shift in 0s */
outp(772,control1 | 0x40);
}
mode=0; /* set microsimulator to */
outp(774,mode); /* mode 0 */
}

write_all(void)
{
int wait;
control2=write_mode;
outp(773,control2 & 0xFE); /* output enable data */
outp(772,control1 | 0x02); /* latch R/W signal */
outp(772,control1);
outp(788,byte[0]); /* write databyte 0 */
outp(789,byte[1]); /* 1 */
outp(790,byte[2]); /* 2 */
outp(791,byte[3]); /* 3 */
outp(772,control1 | 0x01); /* latch data into data */
outp(772,control1); /* input registers */
outp(773,control2 & 0xFA); /* output enable checkbits */
outp(772,control1 | 0x38); /* latch CB, Error into data */
outp(772,control1); /* output registers */
outp(770,0); /* latch CB/SY into interface */
check=inp(785) & 0x7f; /* read checkbit */
outp(768,0); /* latch Error into interface */
control_in=inp(769); /* read error signals */
me_bit=(control_in & 1); /* MULT ERROR */
error_bit=(control_in & 2)/2; /* ERROR */
outp(773,control2); /* disable output */
}

read_all(void)
{
control2=read_mode;
outp(773,control2 & 0x7E); /* enable output data, cb */
outp(772,control1 | 0x02); /* latch R/W signal */
outp(772,control1);
outp(788,byte[0]); /* write databyte 0 */
outp(789,byte[1]); /* 1 */
outp(790,byte[2]); /* 2 */
outp(791,byte[3]); /* 3 */
outp(779,cb); /* write checkbit */
outp(772,control1 | 0x01); /* latch in data, cb */
if (ti)
{
outp(772,control1 | 0x3d); /* latch data, cb, error */
outp(772,control1 | 0x01); /* into output register */
outp(773,control2); /* disable output */
outp(773,control2 & 0xF9); /* output enable data, cb */
outp(787,0); /* latch data into interface */
}
}

```

```

        outp(770,0);
        data[0]=inp(794);
        data[1]=inp(795);
        data[2]=inp(796);
        data[3]=inp(797);
        syndr=inp(785);
        outp(768,0);
        control_in=inp(769);
        outp(772,control1);
    }
else
    {
        outp(772,control1);
        outp(773,control2);
        outp(773,control2 & 0xF9);
        outp(772,control1 | 0x3c);
        outp(772,control1);
        outp(787,0);
        outp(770,0);
        data[0]=inp(794);
        data[1]=inp(795);
        data[2]=inp(796);
        data[3]=inp(797);
        syndr=inp(785);
        outp(768,0);
        control_in=inp(769);
    }
me_bit=(control_in & 1);
error_bit=(control_in & 2)/2;
outp(773,control2);
}

/* latch cb into interface */
/* read databyte 0 */
/*      1 */
/*      2 */
/*      3 */
/* read syndrom */
/* latch error into IF */
/* read error signals */

/* disable output */
/* output enable data, cb */
/* latch data, cb, error */
/* into output register */
/* latch data into interface */
/* latch cb into interface */
/* read databyte 0 */
/*      1 */
/*      2 */
/*      3 */
/* read syndrom */
/* latch error into IF */
/* read error signals */

/* MULT ERROR */
/* ERROR */
/* disable output */

write_out(void)
{
    control2=write_mode;
    outp(773,control2 & 0xFE);
    outp(772,control1 | 0x02);
    outp(772,control1);
    outp(788,0);
    outp(789,0);
    outp(790,0);
    outp(791,0);
    outp(772,control1 | 0x01);
    outp(772,control1);
    outp(772,control1 | 0x0C);
    outp(772,control1);
    outp(773,control2);
}

/* output enable data */
/* latch R/W signal */

/* write databyte 0 */
/*      1 */
/*      2 */
/*      3 */

/* latch data into data */
/* input registers */
/* latch CB, data into data */
/* output registers */
/* disable output */

write_err(void)
{
    control2=read_mode;
    outp(773,control2 & 0x7E);
    outp(772,control1 | 0x02);
    outp(772,control1);
    outp(788,0);
    outp(789,0);
}

/* enable output data, cb */
/* latch R/W signal */

/* write databyte 0 */
/*      1 */

```



```

    outp(790,0);
    outp(791,0);
    outp(779,0xff);
    outp(772,control1 | 0x01);
    outp(772,control1);
    outp(772,control1 | 0x30);
    outp(772,control1);
    outp(773,control2);
    }

    /*          2          */
    /*          3          */
    /* write checkbit */
    /* latch in data, cb */
    /* latch data, cb, error */
    /* into output register */
    /* disable output */

write_in(void)
{
    control2=write_mode;
    outp(773,control2 & 0x7E);
    outp(772,control1 | 0x02);
    outp(772,control1);
    outp(788,0);
    outp(789,0);
    outp(790,0);
    outp(791,0);
    outp(779,0);
    outp(772,control1 | 0x01);
    outp(772,control1);
    outp(773,control2);
    }

    /* output enable data */
    /* latch R/W signal */
    /* write databyte 0 */
    /*          1          */
    /*          2          */
    /*          3          */
    /* write checkbit */
    /* latch data into data */
    /* input registers */
    /* disable output */

read_in(void)
{
    outp(773,control2 & 0xF9);
    outp(787,0);
    outp(770,0);
    data[0]=inp(794);
    data[1]=inp(795);
    data[2]=inp(796);
    data[3]=inp(797);
    syndr=inp(785);
    outp(768,0);
    control_in=inp(769);
    me_bit=(control_in & 1);
    error_bit=(control_in & 2)/2;
    outp(773,control2);
    }

    /* output enable data, cb */
    /* latch data into interface */
    /* latch cb into interface */
    /* read databyte 0 */
    /*          1          */
    /*          2          */
    /*          3          */
    /* read syndrom */
    /* latch error into IF */
    /* read error signals */
    /* MULT ERROR */
    /* ERROR */
    /* disable output */

probe(void)
{
    int probe1,probe2;
    outp(768,0);
    probe1=inp(769);
    probe2=inp(771);
    display_probe(probe1,probe2);
}

conv_byte_to_bin(int byte)
{
    int i;
    for (i=0; i<8; i++)
    {
        binbyte[i]=(byte >> i) & 1;
    }
}

```

```

    }

Power_of_2(int x)
{
    int i;
    int pow2=1;
    for (i=0;i<x;i++)
    {
        pow2*=2;
    }
    return pow2;
}

no_error(void)
{
    int i;
    read_all();
    for(i=0;i<=3;i++)
    {
        if (goodbyte[i]!=data[i])
        {
            error_count+=1;
        }
    }
    if (me_bit==0 || error_bit==0)
    {
        error_count+=1;
    }
}

single_error(void)
{
    int i,j;
    for(i=0;i<=6;i++)
    {
        cb^=Power_of_2(i);
        test_single();
    }
    for(j=0;j<=3;j++)
    {
        for(i=0;i<=7;i++)
        {
            byte[j]^=Power_of_2(i);
            test_single();
        }
    }
}

test_single(void)
{
    int i;
    read_all();
    for(i=0;i<=3;i++)
    {
        if (goodbyte[i]!=data[i])
        {
            error_count+=1;
        }
    }
}

```

ORIGINAL PAGE IS
OF POOR QUALITY

```

if (me_bit==0 || error_bit!=0)
{
    error_count+=1;
}
restore_data();
}

double_error(void)
{
    int i,j,k,l;
    for(i=0;i<=6;i++)
    {
        for(k=i+1;k<=6;k++)
        {
            cb^=Power_of_2(i) | Power_of_2(k);
            read_all();
            if (me_bit!=0 || error_bit!=0)
            {
                error_count+=1;
            }
            restore_data();
        }
        for(l=0;l<=3;l++)
        {
            for(k=0;k<=7;k++)
            {
                cb^=Power_of_2(i);
                byte[l]^=Power_of_2(k);
                read_all();
                if (me_bit!=0 || error_bit!=0)
                {
                    error_count+=1;
                }
                restore_data();
            }
        }
    }
    for(j=0;j<=3;j++)
    {
        for(i=0;i<=7;i++)
        {
            for(k=i+1;k<=7;k++)
            {
                byte[j]^=Power_of_2(i);
                byte[j]^=Power_of_2(k);
                read_all();
                if (me_bit!=0 || error_bit!=0)
                {
                    error_count+=1;
                }
                restore_data();
            }
        }
        for(l=j+1;l<=3;l++)
        {
            for(k=0;k<=7;k++)
            {
                byte[j]^=Power_of_2(i);
                byte[l]^=Power_of_2(k);
                read_all();
                if (me_bit!=0 || error_bit!=0)

```

```

        {
            error_count+=1;
        }
        restore_data();
    }
}
}
}

restore_data(void)
{
    int i;
    for(i=0;i<=3;i++)
    {
        byte[i]=goodbyte[i];
    }
    cb=goodcb;
}

device_screen(void)
{
    int position,not_end,key;
    _settextcolor(white);
    _setbkcolor(black);
    _clearscreen(_GCLEARSCREEN);
    _settextposition(10,20);
    _outtext("Error Detection and Correction");
    _settextposition(15,20);
    _outtext("Select Device:");
    position=1;
    invert(1);
    normal(2);
    normal(3);
    not_end=true;
    do
    {
        if (kbhit())
        {
            key=getch();
            if (key==13)
            {
                if (position==2) ti=true; else ti=false;
                not_end=false;
            }
            if (key==0)
            {
                key=getch();
                switch (key)
                {
                    case 72:
                        if (position>1)
                        {
                            normal(position);
                            position-=1;
                            invert(position);
                        }
                        break;
                    case 80:
                        if (position<3)

```

```

        {
            normal(position);
            position+=1;
            invert(position);
        }
        break;
    }
}
}
while (not_end);
}
normal(int pos)
{
    _settextcolor(white);
    _setbkcolor(black);
    _settextposition(14+pos,40);
    _outtext(text[pos]);
}

invert(int pos)
{
    _settextcolor(black);
    _setbkcolor(white);
    _settextposition(14+pos,40);
    _outtext(text[pos]);
}

edac_screen()
{
    int i,j;
    char *titel="          EDAC - Error Detection and Correction Unit
";
    char *menu=" F1-Write  F2-Read   F4-Data  F5-Checkbits   F7-Probe  F8-Tes
t   F10-Exit ";
    _wrapon(_GWRAPOFF);
    _settextcolor(black);
    _setbkcolor(black);
    _clearscreen(_GCLEARSCREEN);
    _setbkcolor(green);
    _settextposition(1,1);
    _outtext(titel);
    _settextposition(25,1);
    _outtext(menu);
    _setbkcolor(blue);
    for (i=2; i<=24; i++)
    {
        _settextposition(i,1);
        _outtext(line);
    }
    _settextcolor(yellow);
    _settextposition(6,5);
    _outtext(" Byte 3   Byte 2   Byte 1   Byte 0           Checkbits");
    _setbkcolor(white);
    for (j=0; j<=1; j++)
    {
        for (i=0; i<=3; i++)
        {
            _settextposition(9+3*j,5+9*i);
            _outtext(" ");
        }
    }
}

```

```

        }
        _settextposition(9+3*j,60);
        _outtext(" ");
    }
    _settextposition(15,5);
    _outtext(" ");
    _settextposition(15,60);
    _outtext(" ");
    _settextposition(15,70);
    _outtext(" ");
    _setbkcolor(blue);
    _settextposition(8,45);
    _outtext("data");
    _settextposition(9,45);
    _outtext("written");
    _settextposition(11,45);
    _outtext("data");
    _settextposition(12,45);
    _outtext("read");
    _settextposition(15,13);
    _outtext("- Mode");
    _settextposition(15,52);
    _outtext("Error:");
    _settextposition(15,66);
    _outtext("ME:");
}

test_screen()
{
    char *menu= " F1-Start      F4-Single   F5-Double           F9-Stop
                F10-Exit ";
    single=doubl=true;
    _settextcolor(black);
    _setbkcolor(green);
    _settextposition(25,1);
    _outtext(menu);
    display_test(3);
    _setbkcolor(blue);
    _settextcolor(white);
    _settextposition(15,13);
    _outtext("vectors");
    _settextposition(15,56);
    _outtext("discrepancies");
    display_count(15,5);
    display_error_count(15,50);
}

clr_data(int x)
{
    int i;
    _setbkcolor(blue);
    _settextposition(x-1,5);
    _outtext(" ");
    _settextposition(x-1,60);
    _outtext(" ");
    _setbkcolor(white);
    for (i=0; i<=3; i++)
    {
        _settextposition(x,5+9*i);
        _outtext(" ");
    }
}

```

```
    }
    _settextposition(x,60);
    _outtext(" ");
}

clr_lines(int start, int stop, int color)
/* clear lines from start to stop and set to color */
{
    int i;
    _setbkcolor(color);
    for (i=start; i<=stop; i++)
    {
        _settextposition(i,1);
        _outtext(line);
    }
}
```

```

/*
File: PT-TEST.C
Date: March 23, 1989

This program generates the test pattern for an exhaustive test of the
parity networks. Each parity network (13 or 14 inputs) generates one
checkbit. The test pattern are applied to the EDAC and the generated
checkbits are read in and compared with the computed checkbits.
*/

#include<stdio.h>
#include<graph.h>
#include<conio.h>
#include<time.h>
#include<math.h>

int input[7][14]=
{
    {0,3,4,6,9,10,12,15,17,19,20,24,28,99},
    {0,2,3,5,7,9,11,13,16,19,21,22,25,29},
    {1,2,4,5,8,10,11,14,17,20,21,23,26,30},
    {0,2,6,7,8,12,13,14,17,18,22,23,27,31},
    {1,3,4,5,6,7,8,15,16,18,24,25,26,27},
    {1,9,10,11,12,13,14,15,16,18,28,29,30,31},
    {19,20,21,22,23,24,25,26,27,28,29,30,31,99},
};

const long black=0;
const long blue=1;
const long green=2;
const long red=4;
const long white=7;
const long yellow=14;
const long bright=15;
const false=0;
const true=-1;

time_t start,stop;
int inputs,vector,k;
int CBcount=7;
int control1=0;
int control2=0xE7;
int write_mode=0xA7;
int read_mode=0xE7;
int mode=3;
int byte[4],data[4],check,checkbit,cb,cbit;
int control_in,error_bit,me_bit,syndr;
char binbyte[8];

main()
{
    long Power_of_2(int x);
    int i,key,mask,error,ges_error;
    int bit[14];
    error=false;
    ges_error=false;
    pt_screen();
    initialize();
    start_time();
    /* initialize screen */
    /* initialize EDAC */
    /* start timer */
}

```



```

/* mainloop:
   i.....number of checkbit tested
   vector...number of vector applied
   k.....number of input to parity tree
*/
for (i=0; i<CBcount; i++)
{
  if (i==0 ;; i==6) inputs=12; else inputs=13;
  for (vector=0; vector<Power_of_2(inputs+1); vector++)
  {
    byte[0]=byte[1]=byte[2]=byte[3]=0;
    for (k=0; k<=inputs; k++)
    {
      mask=Power_of_2(inputs-k);
      bit[k]=(vector & mask)/mask;
    }
    for (k=0; k<=inputs; k++)
    {
      if (input[i][k]<8)
      {
        byte[0] +=bit[inputs-k]*(Power_of_2(input[i][k]));
      }
      if (input[i][k]>=8 && input[i][k]<16)
      {
        byte[1] +=bit[inputs-k]*(Power_of_2(input[i][k]-8));
      }
      if (input[i][k]>=16 && input[i][k]<24)
      {
        byte[2] +=bit[inputs-k]*(Power_of_2(input[i][k]-16));
      }
      if (input[i][k]>=24 && input[i][k]<32)
      {
        byte[3] +=bit[inputs-k]*(Power_of_2(input[i][k]-24));
      }
    }
    _settextposition(10+i,50);
    printf ("%5d",vector+1);
    write_all();
    calculate_checkbit();
    read_all();
    mask=Power_of_2(i);
    checkbit=(check & mask)/mask;
    if (checkbit != cbit)
    {
      error=true;
      ges_error=true;
    }
    if (kbhit())
    {
      key=getch();
      key=getch();
      switch (key)
      {
        case 67: goto finish;
        case 68: goto exit;
      }
    }
  }
  _settextposition(10+i,20);
  if (error) printf ("Error"); else printf ("ok");
}

```

```

        error=false;
    }
finish:
    stop_time();                /* stop timer          */
    _settextposition(20,1);
    if (ges_error)
    {
        printf ("Test failed! Errors found in Checkbits!");
    }
    else
    {
        printf ("Test completed successfully! No Errors detected!");
    }
    while (!kbhit() );          /* wait for key          */
exit:
    _clearscreen(_GCLEARSCREEN);
}

long Power_of_2(int x)
/* procedure to calculate the power of 2 */
{
    int j;
    long Pow2=1;
    for (j=0; j<x; j++)
    {
        Pow2 *= 2;
    }
    return Pow2;
}

initialize(void)
/* initializes the EDAC */
{
    int i;
    outp(772,control1);        /* set port 772 to 0     */
    outp(773,control2);        /* set port 773 to E7hex */
    mode=3;                    /* set microsimulator to */
    outp(774,mode);            /* mode 3                 */
    for (i=0; i<3000; i++)
    {
        outp(772,control1);    /* shift in 0s           */
        outp(772,control1 | 0x40);
    }
    mode=0;                    /* set microsimulator to */
    outp(774,mode);            /* mode 0                 */
}

write_all(void)
{
    int wait;
    control2=write_mode;
    outp(773,control2 & 0xFE); /* output enable data    */
    outp(772,control1 | 0x02); /* latch R/W signal      */
    outp(772,control1);
    outp(788,byte[0]);         /* write databyte 0      */
    outp(789,byte[1]);         /*                       1 */
    outp(790,byte[2]);         /*                       2 */
    outp(791,byte[3]);         /*                       3 */
    outp(772,control1 | 0x01); /* latch data into data  */
    outp(772,control1);        /* input registers       */
}

```

```

    outp(773,control2 & 0xFA);          /* output enable checkbits */
    outp(772,control1 | 0x38);         /* latch CB, Error into data */
    outp(772,control1);                /* output registers */
    outp(770,0);                       /* latch CB/SY into interface */
    check=inp(785) & 0x7f;             /* read checkbit */
    outp(768,0);                       /* latch Error into interface */
    control_in=inp(769);                /* read error signals */
    me_bit=(control_in & 1);           /* MULT ERROR */
    error_bit=(control_in & 2)/2;      /* ERROR */
    outp(773,control2);                /* disable output */
}

read_all(void)
{
    control2=read_mode;
    outp(773,control2 & 0x7E);         /* enable output data, cb */
    outp(772,control1 | 0x02);         /* latch R/W signal */
    outp(772,control1);
    outp(788,byte[0]);                 /* write databyte 0 */
    outp(789,byte[1]);                 /*           1 */
    outp(790,byte[2]);                 /*           2 */
    outp(791,byte[3]);                 /*           3 */
    outp(779,cb);                      /* write checkbit */
    outp(772,control1 | 0x01);         /* latch in data, cb */
    outp(772,control1);
    outp(773,control2);                /* disable output */
    outp(773,control2 & 0xF9);         /* output enable data, cb */
    outp(772,control1 | 0x3c);         /* latch data, cb, error */
    outp(772,control1);                /* into output register */
    outp(787,0);                       /* latch data into interface */
    outp(770,0);                       /* latch cb into interface */
    data[0]=inp(794);                  /* read databyte 0 */
    data[1]=inp(795);                  /*           1 */
    data[2]=inp(796);                  /*           2 */
    data[3]=inp(797);                  /*           3 */
    syndr=inp(785);                    /* read syndrom */
    outp(768,0);                       /* latch error into IF */
    control_in=inp(769);                /* read error signals */
    me_bit=(control_in & 1);           /* MULT ERROR */
    error_bit=(control_in & 2)/2;      /* ERROR */
    outp(773,control2);                /* disable output */
}

int calculate_checkbit(void)
/* procedure to calculate the checkbit */
{
    int cbyte,i,bit;
    cbyte=byte[0]^byte[1]^byte[2]^byte[3];
    cbit=0;
    for (i=0; i<8; i++)
    {
        bit=(cbyte>>i)&1;
        cbit^=bit;
    }
}

start_time(void)
/* starts timer and displays starttime */
{
    char starttime [9];

```

```

        _strtime(starttime);
        time(&start);
        _settextposition(23,12);
        printf ("%s",starttime);
    }

stop_time(void)
/* stops timer and displays stoptime and testtime */
{
    char stoptime [9];
    int sec,min,hour;
    double x,y,n,times;
    _strtime(stoptime);
    time(&stop);
    _settextposition(23,35);
    printf ("%s",stoptime);
    x=(difftime(stop,start)+.5)/60;
    sec=modf(x,&n)*60;
    x=n/60;
    min=modf(x,&n)*60;
    x=n/60;
    hour=modf(x,&n)*60;
    _settextposition(23,62);
    printf ("%02d:%02d:%02d",hour,min,sec);
}

pt_screen(void)
/* initializes and displays screen */
{
    int line;
    _settextcolor(white);
    _setbkcolor(blue);
    _clearscreen(_GCLEARSCREEN);
    _settextposition(3,10);
    _outtext("EDAC - Error Detection and Correction Unit");
    _settextposition(5,1);
    _outtext ("Testing of the Parity Tree Network - exhaustive");
    _settextposition(23,1);
    _outtext ("Teststart:");
    _settextposition(23,25);
    _outtext ("Teststop:");
    _settextposition(23,50);
    _outtext ("Testtime:");
    _setbkcolor(green);
    _settextcolor(black);
    _settextposition(25,1);
    _outtext ("
top F10-Exit ");
    _settextcolor(yellow);
    _setbkcolor(blue);
    _settextposition(8,18);
    _outtext("Status");
    _settextposition(8,45);
    _outtext("# vectors tested");
    for (line=0; line<=6; line++)
    {
        _settextposition(10+line,3);
        printf ("Checkbit %d",line);
        _settextposition(10+line,50);
        printf ("%5d",0);
    }
}

```

F9-S

} }

```

/*
File: ED-TEST.C
Date: March 23, 1989

This program generates the test pattern for an exhaustive test of
the Error Detector. This part of the EDAC detects single and multiple
errors and generates the error signals ERROR and MULT ERROR. The test
pattern are applied to the EDAC in READ mode and the generated error
signals are compared with the calculated error signals.
*/

#include<stdio.h>
#include<graph.h>
#include<conio.h>
#include<time.h>
#include<math.h>

const long black=0;
const long blue=1;
const long green=2;
const long red=4;
const long white=7;
const long yellow=14;
const long bright=15;
const false=0;
const true=-1;

time_t start,stop;
int controll=0;
int control2=0xE7;
int write_mode=0xA7;
int read_mode=0xE7;
int mode=3;
int byte[4],cb,control_in,error_bit,me_bit,sydr,fdo;
int data[4],check;

main()
{
int i,vector;
int error_flag,me_flag;
int sy[7];
int TOME,ERROR,ME,even;
int _ERROR,_ME;
error_flag=false;
me_flag=false;
ed_screen(); /* initialize screen */
initialize(); /* initialize EDAC */
start_time(); /* start timer */
byte[0]=byte[1]=byte[2]=byte[3]=0;
for (vector=0; vector<128; vector++)
{
cb=vector & 127; /* write data to the EDAC */
write_all();
for (i=0; i<8; i++)
{
sy[i]=(vector>>i) & 1;
}
TOME= (sy[0]&sy[3]&sy[6]) |
(sy[0]&sy[1]&sy[2]) |
(sy[4]&sy[5]&sy[6]) |

```

```

        (sy[0]&sy[3]&sy[4]&sy[5]) ;
        (sy[1]&sy[3]&sy[4]&sy[5]) ;
        (sy[1]&sy[3]&sy[4]&sy[6]) ;
        (sy[1]&sy[3]&sy[5]&sy[6]);
ERROR= sy[0] | sy[1] | sy[2] | sy[3] | sy[4] | sy[5] | sy[6];
even= sy[0] ^ sy[1] ^ sy[2] ^ sy[3] ^ sy[4] ^ sy[5] ^ sy[6];
ME= (!even & ERROR) | TOME;
_ERROR=!ERROR;
_ME=!ME;
read_all(); /* read error signals */
if (_ERROR !=error_bit) error_flag=true;
if (_ME !=me_bit) me_flag=true;
_settextposition(10,50);
printf ("%d",vector+1);
_settextposition(1,1);
}
stop_time(); /* stop timer */
if (!error_flag & !me_flag)
{
_settextposition(10,20);
printf ("ok");
_settextposition(18,1);
printf ("Test completed successfully! No Errors detected!");
}
else
{
_settextposition(10,20);
printf ("Error");
if (error_flag)
{
_settextposition(18,1);
printf ("Hardware Fault in ERROR Signal detected!");
}
if (me_flag)
{
_settextposition(19,1);
printf ("Hardware Fault in MULT ERROR Signal detected!");
}
}
while (!kbhit() ); /* wait for key */
_clearscreen(_GCLEARSCREEN);
}

initialize(void)
/* initializes the EDAC */
{
int i;
outp(772,control1); /* set port 772 to 0 */
outp(773,control2); /* set port 773 to E7hex */
mode=3; /* set microsimulator to */
outp(774,mode); /* mode 3 */
for (i=0; i<3000; i++)
{
outp(772,control1); /* shift in 0s */
outp(772,control1 | 0x40);
}
mode=0; /* set microsimulator to */
outp(774,mode); /* mode 0 */
}

```

```

write_all(void)
{
    int wait;
    control2=write_mode;
    outp(773,control2 & 0xFE);          /* output enable data */
    outp(772,control1 | 0x02);          /* latch R/W signal */
    outp(772,control1);
    outp(788,byte[0]);                  /* write databyte 0 */
    outp(789,byte[1]);                  /*           1 */
    outp(790,byte[2]);                  /*           2 */
    outp(791,byte[3]);                  /*           3 */
    outp(772,control1 | 0x01);          /* latch data into data */
    outp(772,control1);                 /* input registers */
    outp(773,control2 & 0xFA);          /* output enable checkbits */
    outp(772,control1 | 0x38);          /* latch CB, Error into data */
    outp(772,control1);                 /* output registers */
    outp(770,0);                         /* latch CB/SY into interface */
    check=inp(785) & 0x7f;              /* read checkbit */
    outp(768,0);                         /* latch Error into interface */
    control_in=inp(769);                 /* read error signals */
    me_bit=(control_in & 1);             /* MULT ERROR */
    error_bit=(control_in & 2)/2;        /* ERROR */
    outp(773,control2);                 /* disable output */
}

read_all(void)
{
    control2=read_mode;
    outp(773,control2 & 0x7E);          /* enable output data, cb */
    outp(772,control1 | 0x02);          /* latch R/W signal */
    outp(772,control1);
    outp(788,byte[0]);                  /* write databyte 0 */
    outp(789,byte[1]);                  /*           1 */
    outp(790,byte[2]);                  /*           2 */
    outp(791,byte[3]);                  /*           3 */
    outp(779,cb);                       /* write checkbit */
    outp(772,control1 | 0x01);          /* latch in data, cb */
    outp(772,control1);
    outp(773,control2);
    outp(773,control2 & 0xF9);          /* disable output */
    outp(772,control1 | 0x3c);          /* output enable data, cb */
    outp(772,control1);                 /* latch data, cb, error */
    outp(772,control1);                 /* into output register */
    outp(787,0);                         /* latch data into interface */
    outp(770,0);                         /* latch cb into interface */
    data[0]=inp(794);                   /* read databyte 0 */
    data[1]=inp(795);                   /*           1 */
    data[2]=inp(796);                   /*           2 */
    data[3]=inp(797);                   /*           3 */
    syndr=inp(785);                     /* read syndrom */
    outp(768,0);                         /* latch error into IF */
    control_in=inp(769);                 /* read error signals */
    me_bit=(control_in & 1);             /* MULT ERROR */
    error_bit=(control_in & 2)/2;        /* ERROR */
    outp(773,control2);                 /* disable output */
}

start_time(void)
/* starts timer and displays starttime */
{

```



```

char starttime [9];
_sftime(starttime);
time(&start);
_settextposition(23,12);
printf ("%s",starttime);
}

stop_time(void)
/* stops timer and displays stoptime and testtime */
{
char stoptime [9];
int sec,min,hour;
double x,y,n,times;
_sftime(stoptime);
time(&stop);
_settextposition(23,35);
printf ("%s",stoptime);
x=(difftime(stop,start)+.5)/60;
sec=modf(x,&n)*60;
x=n/60;
min=modf(x,&n)*60;
x=n/60;
hour=modf(x,&n)*60;
_settextposition(23,62);
printf ("%02d:%02d:%02d",hour,min,sec);
}

ed_screen(void)
/* initialize and display screen */
{
int line;
_settextcolor(white);
_setbkcolor(blue);
_clearscreen(_GCLEARSCREEN);
_settextposition(3,10);
_outtext("EDAC - Error Detection and Correction Unit");
_settextposition(5,1);
_outtext ("Testing of the Error Detector - exhaustive");
_settextposition(23,1);
_outtext ("Teststart:");
_settextposition(23,25);
_outtext ("Teststop:");
_settextposition(23,50);
_outtext ("Testtime:");
_setbkcolor(green);
_settextcolor(black);
_settextposition(25,1);
_outtext ("
F10-Exit ");
_settextcolor(yellow);
_setbkcolor(blue);
_settextposition(8,18);
_outtext("Status");
_settextposition(8,45);
_outtext("# vectors tested");
}

```

```

/*
File: EC-TEST.C
Date: March 23, 1989

This program generates the test pattern for an exhaustive test of
the Error Locator. In the case of a single error this part of the
EDAC locates the faulty bit and corrects it. The test pattern are
applied to the EDAC in READ mode. The corrected data is checked to
make sure that the faulty bit and only the faulty bit has been
flipped.
*/

#include<stdio.h>
#include<graph.h>
#include<conio.h>
#include<time.h>
#include<math.h>

const long black=0;
const long blue=1;
const long green=2;
const long red=4;
const long white=7;
const long yellow=14;
const long bright=15;
const false=0;
const true=-1;

time_t start,stop;
int control1=0;
int control2=0xE7;
int write_mode=0xA7;
int read_mode=0xE7;
int mode=3;
int byte[4],data[4],cb,check,binbyte[8];
int control_in,error_bit,me_bit,syndr;
int syndrom[32] = {11,52,14,19,21,22,25,26,28,35,37,38,41,42,44,49,50,13,56,
67,69,70,74,76,81,82,84,88,97,98,100,104};

main()
{
int i,j,vector;
int error;
error=false;
ec_screen();
initialize();
start_time();
byte[0]=byte[1]=byte[2]=byte[3]=0;
write_all();
control2=read_mode;
for (vector=0; vector<32; vector++)
{
cb=syndrom[vector];
read_all();
for (i=0; i<4; i++)
{
conv_byte_to_bin(data[i]);
for (j=0; j<8; j++)
{
if (vector==(8*i+j))
/* initialize screen */
/* initialize EDAC */
/* start timer */
/* write data to the EDAC */
}
}
}
}

```

```

        {
            if (binbyte[j] != 1) error=true;
        }
        else
        {
            if (binbyte[j] == 1) error=true;
        }
    }
}
_settextposition(10,50);
printf ("%d",vector+1);
}
stop_time(); /* stop timer */
if (!error)
{
    _settextposition(10,20);
    printf ("ok");
    _settextposition(18,1);
    printf ("Test completed successfully! No Errors detected!");
}
else
{
    _settextposition(10,20);
    printf ("Error");
    _settextposition(18,1);
    printf ("Hardware Fault in Error Locator detected!");
}
while (!kbhit() ); /* wait for key */
_clearscreen(_GCLEARSCREEN);
}

```

```

initialize(void)
/* initializes the EDAC */
{
    int i;
    outp(772,control1); /* set port 772 to 0 */
    outp(773,control2); /* set port 773 to E7hex */
    mode=3; /* set microsimulator to */
    outp(774,mode); /* mode 3 */
    for (i=0; i<3000; i++)
    {
        outp(772,control1); /* shift in 0s */
        outp(772,control1 | 0x40);
    }
    mode=0; /* set microsimulator to */
    outp(774,mode); /* mode 0 */
}

```

```

write_all(void)
{
    int wait;
    control2=write_mode;
    outp(773,control2 & 0xFE); /* output enable data */
    outp(772,control1 | 0x02); /* latch R/W signal */
    outp(772,control1);
    outp(788,byte[0]); /* write databyte 0 */
    outp(789,byte[1]); /* 1 */
    outp(790,byte[2]); /* 2 */
    outp(791,byte[3]); /* 3 */
}

```

```

    outp(772,control1 | 0x01);          /* latch data into data */
    outp(772,control1);                /* input registers */
    outp(773,control2 & 0xFA);         /* output enable checkbits */
    outp(772,control1 | 0x38);         /* latch CB, Error into data */
    outp(772,control1);                /* output registers */
    outp(770,0);                        /* latch CB/SY into interface */
    check=inp(785) & 0x7f;             /* read checkbit */
    outp(768,0);                        /* latch Error into interface */
    control_in=inp(769);                /* read error signals */
    me_bit=(control_in & 1);            /* MULT ERROR */
    error_bit=(control_in & 2)/2;       /* ERROR */
    outp(773,control2);                /* disable output */
}

read_all(void)
{
    control2=read_mode;                /* enable output data, cb */
    outp(773,control2 & 0x7E);         /* latch R/W signal */
    outp(772,control1 | 0x02);         /* write databyte 0 */
    outp(772,control1);                /* write databyte 1 */
    outp(788,byte[0]);                 /* write databyte 2 */
    outp(789,byte[1]);                 /* write databyte 3 */
    outp(790,byte[2]);                 /* write checkbit */
    outp(791,byte[3]);                 /* latch in data, cb */
    outp(779,cb);
    outp(772,control1 | 0x01);         /* disable output */
    outp(772,control1);                /* output enable data, cb */
    outp(773,control2);                /* latch data, cb, error */
    outp(773,control2 & 0xF9);         /* into output register */
    outp(772,control1 | 0x3c);         /* latch data into interface */
    outp(772,control1);                /* latch cb into interface */
    outp(787,0);                        /* read databyte 0 */
    outp(770,0);                        /* read databyte 1 */
    data[0]=inp(794);                  /* read databyte 2 */
    data[1]=inp(795);                  /* read databyte 3 */
    data[2]=inp(796);                  /* read syndrom */
    data[3]=inp(797);                  /* latch error into IF */
    syndr=inp(785);                    /* read error signals */
    outp(768,0);                        /* MULT ERROR */
    control_in=inp(769);                /* ERROR */
    me_bit=(control_in & 1);            /* disable output */
    error_bit=(control_in & 2)/2;
    outp(773,control2);
}

conv_byte_to_bin(int byte)
/* converts 1 byte into 8 bits */
{
    int i;
    for (i=0; i<8; i++)
    {
        binbyte[i]=(byte >> i) & 1;
    }
}

start_time(void)
/* starts timer and displays starttime */
{
    char starttime [9];
    _strtime(starttime);
}

```

```

time(&start);
_settextposition(23,12);
printf ("%s",starttime);
}

stop_time(void)
/* stops timer and displays stoptime and testtime */
{
char stoptime [9];
int sec,min,hour;
double x,y,n,times;
_strtime(stoptime);
time(&stop);
_settextposition(23,35);
printf ("%s",stoptime);
x=(difftime(stop,start)+.5)/60;
sec=modf(x,&n)*60;
x=n/60;
min=modf(x,&n)*60;
x=n/60;
hour=modf(x,&n)*60;
_settextposition(23,62);
printf ("%02d:%02d:%02d",hour,min,sec);
}

ec_screen(void)
/* initialize and display screen */
{
int line;
_settextcolor(white);
_setbkcolor(blue);
_clearscreen(_GCLEARSCREEN);
_settextposition(3,10);
_outtext("EDAC - Error Detection and Correction Unit");
_settextposition(5,1);
_outtext ("Testing of the Error Locator - exhaustive");
_settextposition(23,1);
_outtext ("Teststart:");
_settextposition(23,25);
_outtext ("Teststop:");
_settextposition(23,50);
_outtext ("Testtime:");
_setbkcolor(green);
_settextcolor(black);
_settextposition(25,1);
_outtext ("
F10-Exit ");
_settextcolor(yellow);
_setbkcolor(blue);
_settextposition(8,18);
_outtext("Status");
_settextposition(8,45);
_outtext("# vectors tested");
}

```

```

/*
File: FAULT.C
Date: September 7, 1989

This program performs fault simulation. It generates the good
machine signature, then a fault is injected and the generated
signature is compared with the good machine signature.
*/

#include <stdio.h>
#include <graph.h>
#include <conio.h>

const long black=0;
const long blue=1;
const long green=2;
const long cyan=3;
const long red=4;
const long magenta=5;
const long white=7;
const long yellow=11;
const long bright=15;

const false=0;
const true=1;
const no_fault=0;
const stuck_at=2;
const stuck_open=3;
const write_mode=0xA7;
const read_mode=0xE7;
const correct=0x20;
const detect=0xEF;
const CLK_A=0x08;
const CLK_B=0x10;
const max_fault=1464;
const min_fault=1;
const not_used=19;
const all=1;

char *line="
";
char binbyte[8];
char buffer[255];
char ext[2];
char filename[12]="FAULT.L";
FILE *fault_write,*fault_read,*stream;

int control1=0;
int control2=0xE7;
int mode=0;
int corr=1;
int fault_mode=1;
int fault_num;
int fdo_error=0;
int pass=0;
int finish=0;
int cb=0xaa;
int endloop; shift_num;
int check,error,multerror,error_bit,me_bit,cbit,fdo,error_flag,valid;
int not_detected[10],detectable,detected,pass_num,fault_alt,result;

```

```

int data[1], sign[6], good[6];
int byte[4] = {0xFF, 0xFF, 0xFF, 0xFF};
int test_cycle[11] = {0, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000};
int unused[20] = {1197, 1198, 1199, 1200, 1354, 1355, 1356,
                  1375, 1376, 1377, 1378, 1379, 1380, 1408,
                  1409, 1410, 1414, 1415, 1416};

int fault[2500];
long cycle[10];
float coverage[10];

main()
{
    int i, j, key;
    int detected;
    fault_num=min_fault;
    edac_screen(); /* initialize display */
    initialize(); /* initialize EDAC */
    signature(); /* initial signature */
    display_signature(11, 20);
    display_mode();
    display_pass(3, 70);
    display_data_write(8, 20);
    display_cb(8, 65, cb);
    display_status(13);
    do
    {
        if (kbhit())
        {
            key=getch();
            if (key==0)
            {
                key=getch();
                switch (key)
                {
                    case 59: /* F1-Read/Write */ /* */
                        if (control2==write_mode) /* toggles read/write */ /* */
                            control2=read_mode; /* mode */ /* */
                        else
                            control2=write_mode;
                        display_mode();
                        break;
                    case 60: /* F2-Seed */ /* */
                        data_input(); /* changes seed */ /* */
                        display_data_write(8, 20);
                        checkbit_input();
                        display_cb(8, 65, cb);
                        clear_fault();
                        signature();
                        display_signature(11, 20);
                        clr_lines(18, 23, blue);
                        break;
                    case 62: /* F4-Stuck_at */ /* */
                        mode=stuck_at;
                        start_program();
                        break;
                    case 63: /* F5-Stuck_open */ /* */
                        mode=stuck_open;
                        start_program();
                        break;
                    case 65: /* F7-Clear */ /* */

```

```

clear_fault();
pass=0;
fprintf(stderr, "\n Signatures:\n\n");
for (i=0; i<=500; i++)
{
    cycle[0]=1;
    signature();
    display_signature(11,20);
    fprintf(stderr, "\n cycles: %4d  %02x %02x %02x %02x %02x %
02x %03d %03d %03d %03d %03d %03d", i, sign[5], sign[4], sign[3], sign[2], sign[1], s
ign[0], sign[5], sign[4], sign[3], sign[2], sign[1], sign[0]);
}
fprintf(stderr, "\n\n");
break;
case 66: /* F8-manual */
endloop=false;
byte[0]=byte[1]=byte[2]=byte[3]=0xFF;
cb=0xaa;
manual_screen();
clear_fault();
signature();
display_signature(11,20);
display_mode();
display_fdo();
display_data_write(8,20);
display_cb(8,65,cb);
do
{
    if (kbhit())
    {
        key=getch();
        if (key==0)
        {
            key=getch();
            if (key==59)
            {
                if (control2==write_mode) /* toggles read/wri
te */
                    control2=read_mode; /* mode
*/
                else
                    control2=write_mode;
                display_mode();
            }
            if (key==60)
            {
                data_input(); /* changes seed
*/
                display_data_write(8,20);
                checkbit_input();
                display_cb(8,65,cb);
                /*clear_fault();*/
                signature();
                display_signature(11,20);
                clr_lines(18,23,blue);
            }
            if (key==61)
            {
                clear_fault();
                display_fdo();
            }
        }
    }
}

```



```

        get_cycle(20,5);
        signature();
        display_signature(11,20);
        break;
    case 68: /* F10-Exit */
        finish=true;
        break;
    }
}
}
}
while (!finish);
_setbkcolor(black);
_settextcolor(white);
_clearscreen(_GCLEARSCREEN);
}

start_program(void)
{
    int i,j,k;
    pass=0;
    control2=read_mode;
    clr_lines(17,23,blue);
    for (i=1;i<=1464;i++)
    {
        fault[i]=0;
    }
    get_cycle(20,5); /* get number of cycles */
    display_status(15);
    while (pass<pass_num)
    {
        valid=true;
        not_detected[pass]=0;
        display_pass(3,70);
        if (pass==pass_num-1)
        {
            control2&=0xdf;
            corr=false;
            get_cycles();
            clr_lines(22,24,blue);
        }
        clear_fault(); /* clear fault simulator */
        signature(); /* generate good machine */
        display_signature(11,20); /* signature */
        for (i=0; i<=5; i++)
        {
            good[i]=sign[i];
        }
        outp(772,control1 | 0xc0); /* inject first fault */
        outp(772,control1);
        if (pass==0)
        /*
        first pass of fault simulation, uses all faults,
        injects faults, generates signature, updates status
        display and checks fdo-pin (shift out of fault
        simulator)
        */
        {
            fault_write=fopen("FAULT.LO","w");
            fault_sim();
        }
    }
}

```

```

display_status(15);
for (fault_num=min_fault+1; fault_num<=max_fault; fault_num++)
{
    outp(772,control1 : 0x80); /* shift fault one */
    outp(772,control1); /* position */
    fault_sim();
    display_status(15);
}
read_fdo();
if (fdo!=1)
{
    fdo_error=true;
}
fclose(fault_write);
}
else
/*
File mode, reads the faults from a fault list,
injects a fault, generates the signature and
updates display
*/
{
    fault_mode='all';
    display_pass(3,70);
    itoa(pass-1,ext,10);
    sprintf(filename+7,"%s",ext);
    fault_read=fopen(filename, "r");
    itoa(pass,ext,10);
    sprintf(filename+7,"%s",ext);
    fault_write=fopen(filename, "w");
    fault_alt=min_fault;
    do
    {
        result=fscanf(fault_read,"%d",&fault_num);
        if (result!=EOF)
        {
            shift_fault(fault_num-fault_alt);
            fault_alt=fault_num;
            fault_sim();
            display_status(15);
        }
    }
    while (result!=EOF);
    shift_fault(max_fault-fault_alt);
    read_fdo();
    if (fdo!=1) /* checks fdo pin */
    {
        fdo_error=true;
    }
    fclose(fault_write);
    fclose(fault_read);
}
check_unused(); /* checks for unused faults */
if (valid)
{
    detectable=max_fault-min_fault-not_used+1;
    detected=(detectable-not_detected[pass]);
    coverage[pass]=(float) detected/(float) detectable*100;
    display_result();
}
}

```

```

else
{
    _settextcolor(white);
    _setbkcolor(blue);
    _settextposition(16,5);
    sprintf(buffer,"Result not valid! Error found!");
    _outtext(buffer);
}
pass+=1;
if (fdo_error)
{
    _settextcolor(white);
    _setbkcolor(blue);
    _settextposition(18,5);
    sprintf(buffer,"Error in Faultsimulator!");
    _outtext(buffer);
}
control2;=0x20;
corr=true;
}
}

data_input(void)
/* input databytes byte3 - byte0 in hex-format */
{
    int i,x,y;
    _setbkcolor(cyan);
    _settextcolor(blue);
    for (i=18; i<=23; i++)
    {
        _settextposition(i,5);
        _outtext(" ");
    }
    _settextposition(19,6);
    sprintf (buffer,"Byte 0: "); _outtext(buffer); scanf ("%x",&byte[0]);
    _settextposition(20,6);
    sprintf (buffer,"Byte 1: "); _outtext(buffer); scanf ("%x",&byte[1]);
    _settextposition(21,6);
    sprintf (buffer,"Byte 2: "); _outtext(buffer); scanf ("%x",&byte[2]);
    _settextposition(22,6);
    sprintf (buffer,"Byte 3: "); _outtext(buffer); scanf ("%x",&byte[3]);
}

checkbit_input(void)
/* input checkbits in hex format */
{
    int i;
    _setbkcolor(cyan);
    _settextcolor(blue);
    for (i=20; i<=22; i++)
    {
        _settextposition(i,40);
        _outtext(" ");
    }
    _settextposition(21,40);
    sprintf (buffer," Checkbits: "); _outtext(buffer); scanf ("%x",&cb);
    check=cb;
}

get_fault_num(void)

```

```

/* input fault_num */
{
int i;
_setbkcolor(cyan);
_settextcolor(blue);
for (i=21; i<=23; i++)
{
_settextposition(i,0);
_outtext(" ");
}
_settextposition(22,0);
sprintf (buffer," Fault Nr.: "); _outtext(buffer); scanf ("%d",&fault_num);
}

get_shift_num(void)
/* input shift_num */
{
int i;
_setbkcolor(cyan);
_settextcolor(blue);
for (i=21; i<=23; i++)
{
_settextposition(i,0);
_outtext(" ");
}
_settextposition(22,0);
sprintf (buffer," Shift-Nr.: "); _outtext(buffer); scanf ("%d",&shift_num);
}

get_cycle_num(void)
/* input cycle_num */
{
int i;
pass=0;
_setbkcolor(cyan);
_settextcolor(blue);
for (i=21; i<=23; i++)
{
_settextposition(i,0);
_outtext(" ");
}
_settextposition(22,0);
sprintf (buffer," Cycle Nr.: "); _outtext(buffer); scanf ("%d",&cycle[pass]);
}

get_cycles(void)
/* input cycle_num */
{
int i;
_setbkcolor(cyan);
_settextcolor(blue);
for (i=22; i<=24; i++)
{
_settextposition(i,0);
_outtext(" ");
}
_settextposition(23,0);
sprintf (buffer," Cycle Nr.: "); _outtext(buffer); scanf ("%d",&cycle[pass]);
}

```

```

get_cycle(int x, int y)
/* input number of cycles, terminate with input 0 */
{
    int i;
    _settextcolor(black);
    _setbkcolor(white);
    _settextwindow(x,y,x+3,y+40);
    _clearscreen(_GWINDOW);
    i=0;
    do
    {
        sprintf (buffer, "\n Number of cycles(%d): ", i); _outtext(buffer);
        scanf ("%D",&cycle[i]);
        i+=1;
    }
    while (cycle[i-1]!=0);
    pass_num=i;
    _setbkcolor(blue);
    _clearscreen(_GWINDOW);
    _settextwindow(1,1,25,80);
}

display_byte(int x, int y, int dbyte)
/* displays one byte in hex and bin format */
{
    int i,j;
    _setbkcolor(blue);
    _settextcolor(yellow);
    _settextposition(x-1,y+3);
    sprintf (buffer, "%02x", dbyte);
    _outtext(buffer);
    _setbkcolor(white);
    _settextcolor(black);
    conv_byte_to_bin(dbyte);
    j=0;
    for (i=7; i>=0; i--)
    {
        j+=sprintf(buffer+j, "%d", binbyte[i]);
    }
    _settextposition(x,y);
    _outtext(buffer);
}

display_cb(int x,int y,int dbyte)
/* displays checkbits in hex and bin format */
{
    int i,j;
    _setbkcolor(blue);
    _settextcolor(yellow);
    _settextposition(x-1,y+3);
    sprintf (buffer, "%02x", dbyte);
    _outtext(buffer);
    _setbkcolor(white);
    _settextcolor(black);
    conv_byte_to_bin(dbyte);
    j=0;
    for (i=6; i>=0; i--)
    {

```

```

        j+=sprintf(buffer+j,"%d",binbyte[i]);
    }
    _settextposition(x,y);
    _outtext(buffer);
}

display_data_write(int x,int y)
/* displays databytes byte3 - byte0 */
{
    int j,ypos;
    for (j=0; j<=3; j++)
    {
        ypos=y+j*9;
        display_byte(x,ypos,byte[3-j]);
    }
}

display_signature(int x,int y)
/* displays sigature in hex and bin format */
{
    int i,ypos;
    for (i=0; i<=5; i++)
    {
        ypos=y+i*9;
        display_byte(x,ypos,sign[5-i]);
    }
}

display_mode(void)
/* display "read" or "write" mode */
{
    char *modus;
    if (control2==write_mode)
        modus=" WRITE ";
    else
        modus=" READ ";
    _setbkcolor(white);
    _settextcolor(red);
    _settextposition(3,5);
    _outtext(modus);
}

display_pass(int x,int y)
/* display "all" or "File" mode */
{
    char *modus,*cmode;
    if (fault_mode==all) modus=" all "; else modus=" File ";
    if (corr==true) cmode=" CORRECT "; else cmode=" DETECT ";
    _setbkcolor(white);
    _settextcolor(red);
    _settextposition(3,70);
    _outtext(modus);
    _settextposition(3,35);
    _outtext(cmode);
}

display_fdo(void)
/* display FDO-status*/
{
    read_fdo();
}

```

```

        _setbkcolor(white);
        _settextcolor(red);
        _settextposition(3,70);
        sprintf(buffer," %d ",fdo);
        _outtext(buffer);
    }

display_status(int x)
/* display status of fault simulation */
{
    _setbkcolor(white);
    _settextcolor(black);
    _settextposition(x,20);
    sprintf(buffer,"%4d",fault_num);_outtext(buffer);
    _settextposition(x,40);
    sprintf(buffer,"%4d",not_detected[pass]);_outtext(buffer);
    _settextposition(x,60);
    sprintf(buffer,"%7ld",cycle[pass]);_outtext(buffer);
}

display_result(void)
/* display result of fault simulation */
{
    char *perc="%";
    result_screen();
    _settextcolor(black);
    _setbkcolor(white);
    _settextposition(17,30+pass*8);
    sprintf(buffer,"%7ld",cycle[pass]); _outtext(buffer);
    _settextposition(18,33+pass*8);
    sprintf(buffer,"%4d",not_detected[pass]); _outtext(buffer);
    _settextposition(20,30+pass*8);
    sprintf(buffer,"%5.2f %s",coverage[pass],perc); _outtext(buffer);
}

initialize(void)
/* initializes the EDAC */
{
    int i;
    outp(772,control1);          /* set port 772 to 0      */
    outp(773,control2);          /* set port 773 to E7hex */
    clear_fault();               /* clear Fault Simulator */
    control2=read_mode;          /* set WRITE mode        */
    outp(773,control2);
}

clear_fault(void)
/* clear the fault simulator */
{
    int i;
    outp(774,stuck_open);        /* mode 3                 */
    for (i=0; i<5000; i++)
    {
        outp(772,control1 | 0x80); /* shift in 0s           */
        outp(772,control1);
    }
    outp(774,mode);              /* restore mode           */
}

```



```

shift_fault(int pos)
/* shift fault 'pos' positions */
{
  int i;
  for (i=0; i<pos; i++)
    {
      outp(772,control1 | 0x80);      /* shift in 0s          */
      outp(772,control1);
    }
}

inject(int pos)
/*inject fault*/
{
  outp(772,control1 | 0xC0);
  outp(772,control1);
  if (pos>0) shift_fault(pos);
}

write_seed(void)
/* writes data and checkbits to EDAC */
{
  outp(773,control2 & 0x7E);          /* enable output to EDAC */
  outp(788,byte[0]);                  /* write databyte 0      */
  outp(789,byte[1]);                  /*           1            */
  outp(790,byte[2]);                  /*           2            */
  outp(791,byte[3]);                  /*           3            */
  outp(779,cb);                       /* write checkbits       */
  outp(772,control1 | 0x03);          /* latch in data, R/W    */
  outp(772,control1);
  outp(773,control2);                  /* disable output to EDAC */
  outp(772,control1 | 0x3C);
  outp(772,control1);
}

read_signature(void)
/* reads data, checkbits, and error signals from EDAC */
{
  int control_in;
  outp(773,control2 & 0xF9);          /* output enable          */
  outp(787,0);                         /* latch data into interface */
  outp(770,0);                         /* latch CB into interface */
  outp(768,0);                         /* latch error into interface */
  data[0]=inp(794);                    /* read data and checkbits */
  data[1]=inp(795);
  data[2]=inp(796);
  data[3]=inp(797);
  check=inp(785) & 0x7f;
  control_in=inp(769);                  /* read error signals     */
  me_bit=(control_in & 1);              /* MULT ERROR             */
  error_bit=(control_in & 2)/2;        /* ERROR                  */
  fdo=(control_in & 4)/4;              /* shift output, fault simulator */
  outp(773,control2);                  /* disable output         */
}

read_fdo(void)
/* reads fdo signal from EDAC */
{

```

```

    outp(768,0);
    fdo=(inp(769) & 1)/4;
}

signature(void)
/* generate signature in format: byte3 byte2 byte1 byte0 check error */
{
    int j;
    long i;
    write_seed();
    for (i=1; i<=cycle[pass]; i++)
    {
        for (j=1; j<=41; j++)
        {
            outp(773,control2 ; CLK_B);
            outp(773,control2);
            outp(773,control2 ; CLK_A);
            outp(773,control2);
        }
        outp(772,control1 ; 0x3C);
        outp(772,control1);
    }
    read_signature();
    if (fdo!=0 && fault_num!=max_fault)
    {
        fdo_error=true;
    }
    sign[0]=check;
    sign[1]=me_bit+2*error_bit;
    sign[2]=data[0];
    sign[3]=data[1];
    sign[4]=data[2];
    sign[5]=data[3];
}

fault_sim(void)
/* generates and compares signature */
{
    int i;
    signature();
    i=0;
    while ((sign[i]==good[i]) && (i<=5)) i++;
    if (i<=5)
        fault[fault_num]=cycle[pass];
    else
    {
        not_detected[pass]++;
        fprintf(fault_write,"%4d ",fault_num);
    }
}

check_unused(void)
/* check for unused faults */
{
    int i;
    for (i=0; i<not_used;i++)
    {
        if (fault[unused[i]]!=0)
        {
            valid=false;
        }
    }
}

```

```

        break;
    }
    else
        not_detected[pass!-:
    }
}

conv_byte_to_bin(int byte)
/* convert 1 byte to 8 bits */
{
    int i;
    for (i=0; i<8; i++)
    {
        binbyte[i]=(byte >> i) & 1;
    }
}

edac_screen()
/* initialize screen */
{
    int i,j;
    char *titel="                               EDAC - Fault Simulation";
    char *menu=" F1-Mode F2-Seed F4-Stuck_at F5-Stuck_open F7-PrtSign F8-Ma
nual F10-Exit ";
    _wrapon(_GWRAPOFF);
    _settextcolor(black);
    _setbkcolor(black);
    _clearscreen(_GCLEARSCREEN);
    _setbkcolor(green);
    _settextposition(1,1);
    _outtext(titel);
    _settextposition(25,1);
    _outtext(menu);
    _setbkcolor(blue);
    _settextcolor(yellow);
    for (i=2; i<=24; i++)
    {
        _settextposition(i,1);
        _outtext(line);
    }
    _settextposition(3,13);
    _outtext("- Mode");
    _settextposition(3,60);
    _outtext("faults:");
    _settextposition(5,20);
    _outtext(" Byte 3   Byte 2   Byte 1   Byte 0           Checkbits");
    _settextposition(8,5);
    _outtext("seed:");
    _settextposition(10,5);
    _outtext("good machine");
    _settextposition(11,5);
    _outtext("signature:");
    _settextposition(15,10);
    _outtext("fault #:");
    _settextposition(15,25);
    _outtext("not detected:");
    _settextposition(15,50);
    _outtext("cycles:");
}

```

```

manual_screen()
/* initialize screen */
{
    int i,j;
    char *titel="";
    char *menu= "F1-Mode F2-Seed F3-Clear F4-Inject F5-Shift F6-Sign F7-S_A F8-
S_O F10-Exit ";
    EDAC - Fault Simulation
    _wrapon(_GWRAPOFF);
    _settextcolor(black);
    _setbkcolor(black);
    _clearscreen(_GCLEARSCREEN);
    _setbkcolor(green);
    _settextposition(1,1);
    _outtext(titel);
    _settextposition(25,1);
    _outtext(menu);
    _setbkcolor(blue);
    _settextcolor(yellow);
    for (i=2; i<=24; i++)
    {
        _settextposition(i,1);
        _outtext(line);
    }
    _settextposition(3,13);
    _outtext("- Mode");
    _settextposition(3,60);
    _outtext("FDO:");
    _settextposition(5,20);
    _outtext("Byte 3 Byte 2 Byte 1 Byte 0 Checkbits");
    _settextposition(8,5);
    _outtext("seed:");
    _settextposition(10,5);
    _outtext("good machine");
    _settextposition(11,5);
    _outtext("signature:");
    _settextposition(13,5);
    _outtext("faulty");
    _settextposition(14,5);
    _outtext("signature:");
}

result_screen(void)
/* initialize result display */
{
    _settextcolor(yellow);
    _setbkcolor(blue);
    _settextposition(17,5);
    _outtext("cycles completed:");
    _settextposition(18,5);
    _outtext("# faults not detected:");
    _settextposition(20,5);
    _outtext("fault coverage:");
}

clr_lines(int start, int stop, int color)
/* clear lines from start to stop and set to color */
{

```

```
int i;  
_setbkcolor(color);  
for (i=start; i<=stop; i++)  
{  
    _settextposition(i,1);  
    _outtext(line);  
}
```

```

/*
    File: FAULT1.C
    Date: April 12, 1989

    This program performs stuck-at fault simulation for the 2nd part of
    the breadboard (i.e., latches). It generates the good machine
    signature, then a fault is injected and the generated signature is
    compared with the good machine signature.
*/

#include <stdio.h>
#include <graph.h>
#include <conio.h>

const long black=0;
const long blue=1;
const long green=2;
const long cyan=3;
const long red=4;
const long magenta=5;
const long white=7;
const long yellow=14;
const long bright=15;

const false=0;
const true=1;
const no_fault=0;
const stuck_at=2;
const stuck_open=3;
const write_mode=0xA7;
const read_mode=0xE7;
const CLK_A=0x08;
const CLK_B=0x10;
const max_fault=972;
const not_used=3;
const all=1;

char *line="
";
char binbyte[8];
char buffer[255];
char ext[2];
char filename[12]="FAULT.L";
FILE *fault_write,*fault_read,*stream;

int control1=0;
int control2=0xE7;
int mode=0;
int cnt=0;
int fault_mode=1;
int fault_num,pass2;
int fdo_error=0;
int pass=0;
int finish=0;
int cb=0xff;
int endloop;shift_num;
int check,error,multerror,error_bit,me_bit,cbit,fdo,error_flag,valid;
int not_detected[10],detectable,detected,pass_num,fault_alt,result;
int data[4],sign[6],good[6];
int byte[4] = {0xFF,0xFF,0xFF,0xFF};

```

```

unused[4]={379,381,383};
int fault[1000];
long cycle[10];
float coverage[10];

main()
{
  int i,j,key;
  int detected;
  fault_num=1;
  edac_screen(); /* initialize display */
  initialize(); /* initialize EDAC */
  signature(); /* initial signature */
  display_signature(11,20);
  display_mode();
  display_pass(3,70);
  display_data_write(8,20);
  display_cb(8,65,cb);
  display_status(15);
  do
  {
    if (kbhit())
    {
      key=getch();
      if (key==0)
      {
        key=getch();
        switch (key)
        {
          case 59: /* F1-Read/Write */
            if (control2==write_mode) /* toggles read/write */
              control2=read_mode; /* mode */
            else
              control2=write_mode;
            display_mode();
            break;
          case 60: /* F2-Seed */
            data_input(); /* changes seed */
            display_data_write(8,20);
            checkbit_input();
            display_cb(8,65,cb);
            clear_fault();
            signature();
            display_signature(11,20);
            clr_lines(18,23,blue);
            break;
          case 62: /* F4-Stuck_at */
            pass=0;
            clr_lines(17,23,blue);
            for (i=1;i<=972;i++)
            {
              fault[i]=0;
            }
            get_cycle(20,5); /* get number of cycles */
            display_status(15);
            mode=0; /* set stuck-at mode */
            while (pass<pass_num)
            {
              valid=true;
            }
          }
        }
      }
    }
  }
}

```

```

not_detected[pass]=0;
display_pass(3,70);
clear_fault(); /* clear fault simulator */
if (pass==0)
/* first pass of fault simulation, uses all faults,
  injects faults, generates signature, updates status
  display and checks fdo-pin (shift out of fault
  simulator)
*/
{
simulation(0,0,0,0);
not_detected[pass]=0;
simulation(0xff,0xff,0,0);
not_detected[pass]=0;
simulation(0xff,0xff,0xff,0xff);
not_detected[pass]=0;
control2=write_mode;
simulation(0x00,0x00,0xff,0xff);
control2=read_mode;
not_detected[pass]-=6;
single_sim();
fault_write=fopen("FAULT.LO","w");
for (i=1;i<=972;i++)
{
if (fault[i]==0)
{
fprintf(fault_write,"%4d ",i);
}
}
fclose(fault_write);
}
else
/* File mode, reads the faults from a fault list,
  injects a fault, generates the signature and
  updates display
*/
{
signature(); /* generate good machine */
display_signature(11,20); /* signature */
for (i=0; i<=5; i++)
{
good[i]=sign[i];
}
outp(772,control1 | 0xC0); /* inject first fault */
outp(772,control1);
fault_mode=!all;
display_pass(3,70);
itoa(pass-1,ext,10);
sprintf(filename+7,"%s",ext);
fault_read=fopen(filename, "r");
itoa(pass,ext,10);
sprintf(filename+7,"%s",ext);
fault_write=fopen(filename, "w");
fault_alt=1;
do
{
result=fscanf(fault_read,"%4d",&fault_num);
if (result!=EOF)
{
shift_fault(fault_num-fault_alt);
}
}
}

```



```

        fault_alt=fault_num;
        fault_sim();
        display_status(15);
    }
}
while (result!=EOF);
shift_fault(max_fault-fault_alt);
read_fdo();
if (fdo!=1) /* checks fdo pin */
{
    fdo_error=true;
}
fclose(fault_write);
fclose(fault_read);
}
check_unused(); /* checks for unused faults */
if (valid)
{
    detectable=max_fault-not_used;
    detected=(detectable-not_detected[pass]);
    coverage[pass]=(float) detected/(float) detectable*100;
    display_result();
}
else
{
    _settextcolor(white);
    _setbkcolor(blue);
    _settextposition(16,5);
    sprintf(buffer,"Result not valid! Error found!");
    _outtext(buffer);
}
pass+=1;
if (fdo_error)
{
    _settextcolor(white);
    _setbkcolor(blue);
    _settextposition(18,5);
    sprintf(buffer,"Error in Faultsimulator!");
    _outtext(buffer);
}
}
break;
case 65:
clear_fault();
pass=0;
fprintf(stdprn, "\n Signatures:\n\n");
for (i=0;i<=100;i++)
{
    cycle[0]=i;
    signature();
    display_signature(11,20);
    fprintf(stdprn, "\n cycles: %4d %02x %02x %02x %02x %02x %02x %02x %03d %03d %03d %03d %03d %03d", i, sign[5], sign[4], sign[3], sign[2], sign[1], sign[0], sign[0], sign[5], sign[4], sign[3], sign[2], sign[1], sign[0]);
}
fprintf(stdprn, "\n\n");
break;
case 66: /* F8-manual */
endloop=false;
manual_screen();

```

```

clear_fault();
signature();
display_signature(11,20);
display_mode();
display_fdo();
display_data_write(8,20);
display_cb(8,65,cb);
do
{
  if (kbhit())
  {
    key=getch();
    if (key==0)
    {
      key=getch();
      if (key==59) /* F1-Read/Write
*/
      {
        if (control2==write_mode) /* toggles read/write
*/
          control2=read_mode; /* mode
*/
      }
      else
      {
        control2=write_mode;
        display_mode();
      }
    }
    if (key==60) /* F2-Seed
*/
    {
      data_input(); /* changes seed
*/
      display_data_write(8,20);
      checkbit_input();
      display_cb(8,65,cb);
      clear_fault();
      signature();
      display_signature(11,20);
      clr_lines(18,23,blue);
    }
    if (key==61) /* F3-Clear
*/
    {
      clear_fault();
      display_fdo();
    }
    if (key==62) /* F4-Inject Fault
*/
    {
      get_fault_num();
      inject(fault_num);
      display_fdo();
    }
    if (key==63) /* F5-Shift Fault
*/
    {
      get_shift_num();
      shift_fault(shift_num);
      display_fdo();
    }
    if (key==64) /* F6-Signature

```

```

        */
        {
            get_cycle_num();
            signature();
            display_signature(11,20);
            display_fdo();
        }
        if (key==65) /* F7-Fault Simulatio
n */
        {
            mode=stuck_at; /* set stuck-at mode
*/
            outp(774,mode);
            signature();
            outp(774,0);
            display_signature(14,20);
        }
        if (key==68)
        {
            endloop=true;
            byte[0]=byte[1]=byte[2]=byte[3]=0xFF;
            ;
            fault_num=1;
            edac_screen(); /* initialize display
*/
            clear_fault();
            signature(); /* initial signature
*/
            display_signature(11,20);
            display_mode();
            display_pass(3,70);
            display_data_write(8,20);
            display_cb(8,65,cb);
            display_status(15);
        }
    }
}
while (!endloop);
break;
case 68: /* F10-Exit */
    finish=true;
    break;
}
}
}
while (!finish);
    _setbkcolor(black);
    _settextcolor(white);
    _clearscreen(_GCLEARSCREEN);
}

data_input(void)
/* input databytes byte3 - byte0 in hex-format */
{
    int i,x,y;
    _setbkcolor(cyan);
    _settextcolor(blue);
    for (i=18; i<=23; i++)

```

```

        {
            _settextposition(i,5);
            _outtext(" ");
        }
        _settextposition(19,6);
        sprintf (buffer,"Byte 0: "); _outtext(buffer); scanf ("%x",&byte[0]);
        _settextposition(20,6);
        sprintf (buffer,"Byte 1: "); _outtext(buffer); scanf ("%x",&byte[1]);
        _settextposition(21,6);
        sprintf (buffer,"Byte 2: "); _outtext(buffer); scanf ("%x",&byte[2]);
        _settextposition(22,6);
        sprintf (buffer,"Byte 3: "); _outtext(buffer); scanf ("%x",&byte[3]);
    }

checkbit_input(void)
/* input checkbits in hex format */
{
    int i;
    _setbkcolor(cyan);
    _settextcolor(blue);
    for (i=20; i<=22; i++)
    {
        _settextposition(i,40);
        _outtext(" ");
    }
    _settextposition(21,40);
    sprintf (buffer," Checkbits: "); _outtext(buffer); scanf ("%x",&cb);
    check=cb;
}

get_fault_num(void)
/* input fault_num */
{
    int i;
    _setbkcolor(cyan);
    _settextcolor(blue);
    for (i=21; i<=23; i++)
    {
        _settextposition(i,0);
        _outtext(" ");
    }
    _settextposition(22,0);
    sprintf (buffer," Fault Nr.: "); _outtext(buffer); scanf ("%d",&fault_num);
}

get_shift_num(void)
/* input shift_num */
{
    int i;
    _setbkcolor(cyan);
    _settextcolor(blue);
    for (i=21; i<=23; i++)
    {
        _settextposition(i,0);
        _outtext(" ");
    }
    _settextposition(22,0);
    sprintf (buffer," Shift-Nr.: "); _outtext(buffer); scanf ("%d",&shift_num);
}

```

```

get_cycle_num(void)
/* input cycle_num */
{
    int i;
    pass=0;
    _setbkcolor(cyan);
    _settextcolor(blue);
    for (i=21; i<=23; i++)
        {
            _settextposition(i,0);
            _outtext(" ");
        }
    _settextposition(22,0);
    sprintf (buffer," Cycle Nr.: "); _outtext(buffer); scanf ("%d",&cycle[pass]);
}

get_cycle(int x, int y)
/* input number of cycles, terminate with input 0 */
{
    int i;
    _settextcolor(black);
    _setbkcolor(white);
    _settextwindow(x,y,x+3,y+40);
    _clearscreen(_GWINDOW);
    cycle[0]=0;
    i=1;
    do
        {
            sprintf (buffer,"\n Number of cycles(%d): ",i); _outtext(buffer);
            scanf ("%D",&cycle[i]);
            i+=1;
        }
    while (cycle[i-1]!=0);
    pass_num=i-1;
    _setbkcolor(blue);
    _clearscreen(_GWINDOW);
    _settextwindow(1,1,25,80);
}

display_byte(int x, int y, int dbyte)
/* displays one byte in hex and bin format */
{
    int i,j;
    _setbkcolor(blue);
    _settextcolor(yellow);
    _settextposition(x-1,y+3);
    sprintf (buffer,"%02x",dbyte);
    _outtext(buffer);
    _setbkcolor(white);
    _settextcolor(black);
    conv_byte_to_bin(dbyte);
    j=0;
    for (i=7; i>=0; i--)
        {
            j+=sprintf(buffer+j,"%d",binbyte[i]);
        }
    _settextposition(x,y);
    _outtext(buffer);
}

```

```

display_cb(int x,int y,int dbyte)
/* displays checkbits in hex and bin format */
{
    int i,j;
    _setbkcolor(blue);
    _settextcolor(yellow);
    _settextposition(x-1,y+3);
    sprintf (buffer,"%02x",dbyte);
    _outtext(buffer);
    _setbkcolor(white);
    _settextcolor(black);
    conv_byte_to_bin(dbyte);
    j=0;
    for (i=6; i>=0; i--)
        {
            j+=sprintf(buffer+j,"%d",binbyte[i]);
        }
    _settextposition(x,y);
    _outtext(buffer);
}

display_data_write(int x,int y)
/* displays databytes byte3 - byte0 */
{
    int j,ypos;
    for (j=0; j<=3; j++)
        {
            ypos=y+j*9;
            display_byte(x,ypos,byte[3-j]);
        }
}

display_signature(int x,int y)
/* displays sigature in hex and bin format */
{
    int i,ypos;
    for (i=0; i<=5; i++)
        {
            ypos=y+i*9;
            display_byte(x,ypos,sign[5-i]);
        }
}

display_mode(void)
/* display "read" or "write" mode */
{
    char *modus;
    if (control2==write_mode)
        modus=" WRITE ";
    else
        modus=" READ ";
    _setbkcolor(white);
    _settextcolor(red);
    _settextposition(3,5);
    _outtext(modus);
}

display_pass(int x,int y)
/* display "all" or "File" mode */

```

```

    {
    char *modus;
    if (fault_mode==all)
        modus=" all ";
    else
        modus=" File ";
    _setbkcolor(white);
    _settextcolor(red);
    _settextposition(3,70);
    _outtext(modus);
    }

display_fdo(void)
/* display FDO-status*/
{
    read_fdo();
    _setbkcolor(white);
    _settextcolor(red);
    _settextposition(3,70);
    sprintf(buffer," %d ",fdo);
    _outtext(buffer);
}

display_status(int x)
/* display status of fault simulation */
{
    _setbkcolor(white);
    _settextcolor(black);
    _settextposition(x,20);
    sprintf(buffer,"%4d",fault_num);_outtext(buffer);
    _settextposition(x,40);
    sprintf(buffer,"%4d",not_detected[pass]);_outtext(buffer);
    _settextposition(x,60);
    sprintf(buffer,"%7ld",cycle[pass]);_outtext(buffer);
}

display_result(void)
/* display result of fault simulation */
{
    char *perc="%";
    result_screen();
    _settextcolor(black);
    _setbkcolor(white);
    _settextposition(17,30+pass*8);
    sprintf(buffer,"%7ld",cycle[pass]); _outtext(buffer);
    _settextposition(18,33+pass*8);
    sprintf(buffer,"%4d",not_detected[pass]); _outtext(buffer);
    _settextposition(20,30+pass*8);
    sprintf(buffer,"%5.2f %s",coverage[pass],perc); _outtext(buffer);
}

initialize(void)
/* initializes the EDAC */
{
    int i;
    outp(774,stuck_open);
    outp(772,control1);
    outp(773,control2);
    clear_fault();
    /* set port 772 to 0 */
    /* set port 773 to E7hex */
    /* clear Fault Simulator */
}

```

```

    outp(774,no_fault);
    control2=read_mode;
    outp(773,control2);          /* set Read mode          */
}

clear_fault(void)
/* clear the fault simulator */
{
    int i;
    for (i=0; i<5000; i++)
    {
        outp(772,control1 | 0x80);    /* shift in 0s          */
        outp(772,control1);
    }
}

shift_fault(int pos)
/* shift fault 'pos' positions */
{
    int i;
    for (i=0; i<pos; i++)
    {
        outp(772,control1 | 0x80);    /* shift in 0s          */
        outp(772,control1);
    }
}

inject(int pos)
/*inject fault*/
{
    outp(772,control1 | 0xC0);        /* shift in a 1          */
    outp(772,control1);
    if (pos>1) shift_fault(pos-1);
}

write_seed(void)
/* writes data and checkbits to EDAC */
{
    control2&=0x7E;
    outp(773,control2);                /* enable output to EDAC */
    outp(788,byte[0]);                 /* write databyte 0      */
    outp(789,byte[1]);                 /*           1            */
    outp(790,byte[2]);                 /*           2            */
    outp(791,byte[3]);                 /*           3            */
    outp(779,cb);                       /* write checkbits       */
    outp(772,control1 | 0x03);         /* latch in data, R/W    */
    outp(772,control1);
    outp(772,control1 | 0x3C);         /* latch data into       */
    outp(772,control1);                 /* output registers      */
}

write_data(int dat,int ckb)
/* writes data and checkbits to EDAC */
{
    control2&=0x7E;
    outp(773,control2);                /* enable output to EDAC */
    outp(788,dat);                     /* write databyte 0      */
    outp(789,dat);                     /*           1            */
    outp(790,dat);                     /*           2            */
}

```



```

    outp(791,dat);
    outp(779,ckb);
    outp(772,control1 | 0x03);
    outp(772,control1);
    outp(772,control1 | 0x3C);
    outp(772,control1);
}

read_signature(void)
/* reads data, checkbits, and error signals from EDAC */
{
    int control_in;
    control2|=0x81;
    outp(773,control2 & 0xF9);
    outp(787,0);
    outp(770,0);
    outp(768,0);
    data[0]=inp(794);
    data[1]=inp(795);
    data[2]=inp(796);
    data[3]=inp(797);
    check=inp(785) & 0x7f;
    control_in=inp(769);
    me_bit=(control_in & 1);
    error_bit=(control_in & 2)/2;
    fdo=(control_in & 4)/4;
    control2&=0x7E;
    outp(773,control2);
    sign[0]=check;
    sign[1]=me_bit+2*error_bit;
    sign[2]=data[0];
    sign[3]=data[1];
    sign[4]=data[2];
    sign[5]=data[3];
}

single_sim(void)
/* sequence necessary to detect faults 967,961, and 919 */
{
    int i;
    clear_fault();
    control2=write_mode;
    write_data(0x00,0xff);
    good_signature();
    write_data(0xff,0xff);
    outp(773,control2 | CLK_B);
    outp(773,control2);
    fault_num=967;
    inject(fault_num);
    outp(774,stuck_at);
    write_data(0x00,0xff);
    compare_signature();
    display_status(15);
    clear_fault();
    write_data(0x00,0xff);
    control2=read_mode;
    outp(773,control2);
    good_signature();
    write_data(0x00,0xff);
    control2=write_mode;
}

```

```

fault_num=961;                               /* fault # 961          */
inject(fault_num);
outp(774,stuck_at);
write_data(0x00,0xff);
control2=read_mode;
outp(773,control2);
compare_signature();
display_status(15);
clear_fault();
write_data(0x00,0x00);
good_signature();
control2=write_mode;
write_data(0x00,0x00);
outp(773,control2 | CLK_B);
outp(773,control2);
control2=read_mode;
fault_num=919;                               /* fault # 919          */
inject(fault_num);
outp(774,stuck_at);
write_data(0x00,0x00);
compare_signature();
display_status(15);
for (i=0;i<=2;i++)
    {
        fault_num=374;
        test_fault();
        fault_num=376;
        test_fault();
        fault_num=380;
        test_fault();
    }
}

test_fault(void)
/* sequence necessary to detect faults 374,376, and 380 */
{
    clear_fault();
    control2|=0x08;
    write_data(0x00,0x00);
    good_signature();
    inject(fault_num);
    outp(774,stuck_at);
    write_data(0x00,0x00);
    compare_signature();
    display_status(15);
    outp(774,0);
    control2&=0xf7;
}

good_signature(void)
/* generates good signature in pass 0 */
{
    int i;
    read_signature();
    display_signature(11,20);
    getch();
    for (i=0;i<=5;i++)
        {
            good[i]=sign[i];
        }
}

```

```

    }

read_fdo(void)
/* reads fdo signal from EDAC */
{
    control2|=0x81;
    outp(768,0); /* latch data into interface */
    fdo=(inp(769) & 4)/4; /* shift output, fault simulator */
}

signature(void)
/* generate signature in format: byte3 byte2 byte1 byte0 check error */
{
    int j;
    long i;
    write_data(0,0);
    outp(773,control2 | CLK_B); /* initializes latches */
    outp(773,control2); /* enable faults */
    outp(774,mode); /* latch data into */
    outp(772,control1 | 0x3C); /* output registers */
    write_seed(); /* repeat number of cycles */
    for (i=1; i<=cycle[pass]; i++) /* shift LFSR 41 times */
    {
        for (j=1; j<=41; j++)
        {
            outp(773,control2 | CLK_B);
            outp(773,control2);
            outp(773,control2 | CLK_A);
            outp(773,control2);
        }
        outp(772,control1 | 0x3C); /* latch data into */
        outp(772,control1); /* output registers */
    }
    read_signature(); /* read signature */
    outp(774,no_fault); /* disable faults */
    if (fdo!=0 && fault_num!=max_fault) /* check fdo output */
    {
        fdo_error=true;
    }
}

compare_signature(void)
/* compares signature */
{
    int i;
    i=0;
    read_signature(); /* read signature */
    outp(774,0);
    if (fdo!=0 && fault_num!=max_fault) /* check fdo output */
    {
        fdo_error=true;
    }
    i=0;
    while ((sign[i]==good[i]) && (i<=5)) i++;
    if (i<=5)
        fault[fault_num]=1;
    else
    {
        fault[fault_num]=0;
    }
}

```

```

        not_detected[pass]++;
    }
}

simulation(int data0,int cb0,int datal,int cb1)
/* performs fault simulation in pass 0 */
{
    int i;
    write_data(datal,cb1);          /* write seed          */
    read_signature();              /* read and display good */
    display_signature(11,20);      /* machine signature    */
    getch();
    for (i=0; i<=5; i++)
    {
        good[i]=sign[i];
    }
    outp(772,control1 ; 0xC0);     /* inject first fault   */
    outp(772,control1);
    if (fault[1]==0)               /* if not detected     */
    {                               /* perform fault simulation */
        write_data(data0,cb0);
        outp(773,control2 ; CLK_B);
        outp(773,control2);
        outp(774,mode);
        write_data(datal,cb1);
        compare_signature();
        display_status(15);
    }
    for (fault_num=2; fault_num<=max_fault; fault_num++)
    {
        outp(772,control1 ; 0x80); /* shift fault one     */
        outp(772,control1);       /* position             */
        if (fault[fault_num]==0)  /* if not detected     */
        {                          /* perform fault simulation */
            write_data(data0,cb0);
            outp(773,control2 ; CLK_B);
            outp(773,control2);
            outp(774,mode);
            write_data(datal,cb1);
            compare_signature();
            display_status(15);
        }
    }
    read_fdo();                    /* check FDO status    */
    if (fdo!=1)
    {
        fdo_error=true;
    }
}

fault_sim(void)
/* generates and compares signature */
{
    int i;
    signature();
    i=0;
    while ((sign[i]==good[i]) && (i<=5)) i++;
    if (i<=5)
        fault[fault_num]=cycle[pass];
}

```

```

else
{
    not_detected[pass]++;
    fprintf(fault_write, "%4d ", fault_num);
}
}

```

```

check_unused(void)
/* check for unused faults */
{
    int i;
    for (i=0; i<not_used; i++)
    {
        if (fault[unused[i]]!=0)
        {
            valid=false;
            break;
        }
        else
            not_detected[pass]--;
    }
}

```

```

conv_byte_to_bin(int byte)
/* convert 1 byte to 8 bits */
{
    int i;
    for (i=0; i<8; i++)
    {
        binbyte[i]=(byte >> i) & 1;
    }
}

```

```

edac_screen()
/* initialize screen */
{
    int i,j;
    char *titel="
";
    char *menu=" F1-Mode F2-Seed F4-Stuck_at F7-PrtSign F8-Manual
F10-Exit ";
    _wraon(_GWRAPOFF);
    _settextcolor(black);
    _setbkcolor(black);
    _clearscreen(_GCLEARSCREEN);
    _setbkcolor(green);
    _settextposition(1,1);
    _outtext(titel);
    _settextposition(25,1);
    _outtext(menu);
    _setbkcolor(blue);
    _settextcolor(yellow);
    for (i=2; i<=24; i++)
    {
        _settextposition(i,1);
        _outtext(line);
    }
    _settextposition(3,13);
    _outtext("- Mode");
    _settextposition(3,60);
}

```

EDAC - Fault Simulation

F4-Stuck_at F7-PrtSign F8-Manual

```

    _outtext("faults:");
    _settextposition(5,20);
    _outtext(" Byte 3   Byte 2   Byte 1   Byte 0           Checkbits");
    _settextposition(8,5);
    _outtext("seed:");
    _settextposition(10,5);
    _outtext("good machine");
    _settextposition(11,5);
    _outtext("signature:");
    _settextposition(15,10);
    _outtext("fault #:");
    _settextposition(15,25);
    _outtext("not detected:");
    _settextposition(15,50);
    _outtext("cycles:");
}

manual_screen()
/* initialize screen */
{
    int i,j;
    char *titel="                               EDAC - Fault Simulation";
    char *menu=" F1-Mode F2-Seed F3-Clear F4-Inject F5-Shift F6-Sign F7-Fa";
ult    F10-Exit ";
    _wrapon(_GWRAPOFF);
    _settextcolor(black);
    _setbkcolor(black);
    _clearscreen(_GCLEARSCREEN);
    _setbkcolor(green);
    _settextposition(1,1);
    _outtext(titel);
    _settextposition(25,1);
    _outtext(menu);
    _setbkcolor(blue);
    _settextcolor(yellow);
    for (i=2; i<=24; i++)
    {
        _settextposition(i,1);
        _outtext(line);
    }
    _settextposition(3,13);
    _outtext("- Mode");
    _settextposition(3,60);
    _outtext("FDO:");
    _settextposition(5,20);
    _outtext(" Byte 3   Byte 2   Byte 1   Byte 0           Checkbits");
    _settextposition(8,5);
    _outtext("seed:");
    _settextposition(10,5);
    _outtext("good machine");
    _settextposition(11,5);
    _outtext("signature:");
    _settextposition(13,5);
    _outtext("faulty");
    _settextposition(14,5);
    _outtext("signature:");
}

```

```

result_screen(void)
/* initialize result display */
{
    _settextcolor(yellow);
    _setbkcolor(blue);
    _settextposition(17,5);
    _outtext("cycles completed:");
    _settextposition(18,5);
    _outtext("# faults not detected:");
    _settextposition(20,5);
    _outtext("fault coverage:");
}

clr_lines(int start, int stop, int color)
/* clear lines from start to stop and set to color */
{
    int i;
    _setbkcolor(color);
    for (i=start; i<=stop; i++)
    {
        _settextposition(i,1);
        _outtext(line);
    }
}

```

APPENDIX G: PC-INTERFACE BOARD

G.1 Logic Diagram of PC-Interface Board

G.1.1 Map of PC-Interface Board.....G-1

G.1.2 Figure 1.....G-2

G.1.3 Figure 2.....G-3

G.1.4 Figure 3.....G-4

G.1.5 Figure 4.....G-5

G.1.6 Figure 5.....G-6

G.1.7 Figure 6.....G-7

G.1.8 Figure 7.....G-8

G.1.9 Figure 8.....G-9

G.1.10 Figure 9.....G-10

G.1.11 Figure 10.....G-11

G.1.12 Figure 11.....G-12

G.1.13 Figure 12.....G-13

G.1.14 Figure 13.....G-14

G.1.15 Figure 14.....G-15

G.1.16 Figure 15.....G-16

G.1.17 Figure 16.....G-17

G.1.18 **Figure** 17.....G-18

G.1.19 **Figure** 18.....G-19

G.1.20 **Figure** 19.....G-20

G.1.21 **Figure** 20.....G-21

Map of PC Interface Board

Figure 1	Figure 2	Figure 3	Figure 4
Figure 5	Figure 6	Figure 7	Figure 8
Figure 9	Figure 10	Figure 11	Figure 12
Figure 13	Figure 14	Figure 15	Figure 16
Figure 17	Figure 18	Figure 19	Figure 20

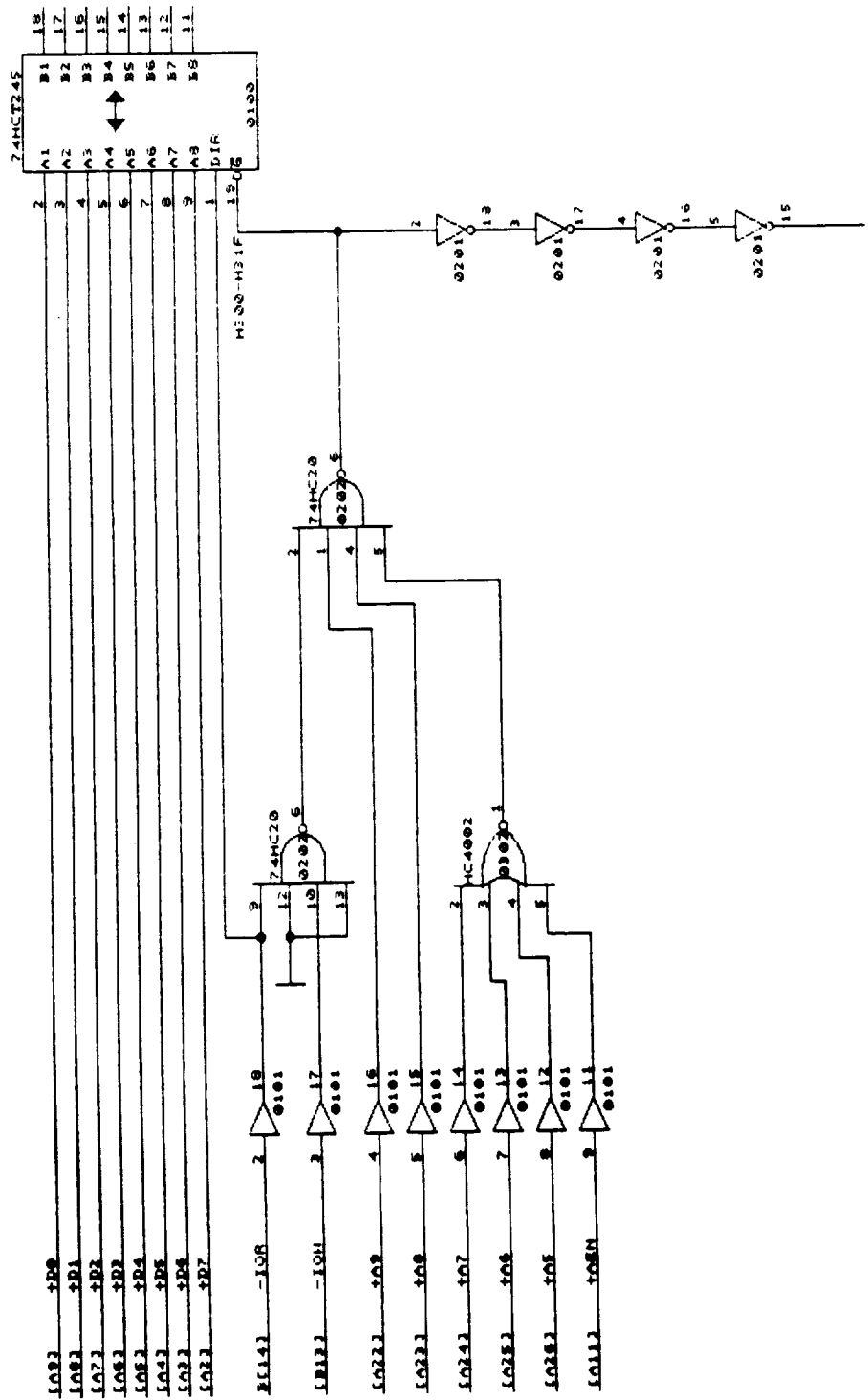


Figure 1

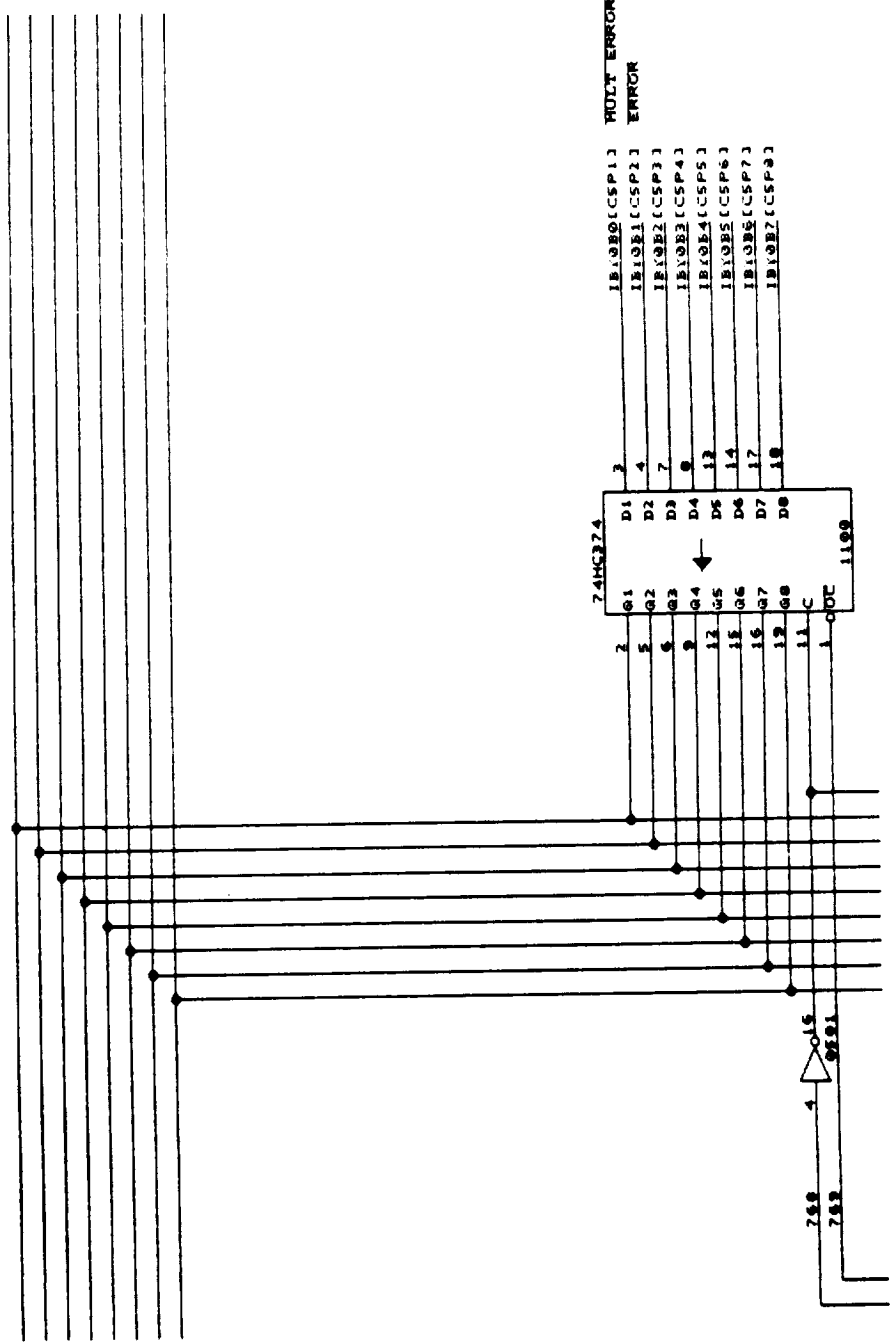


Figure 2

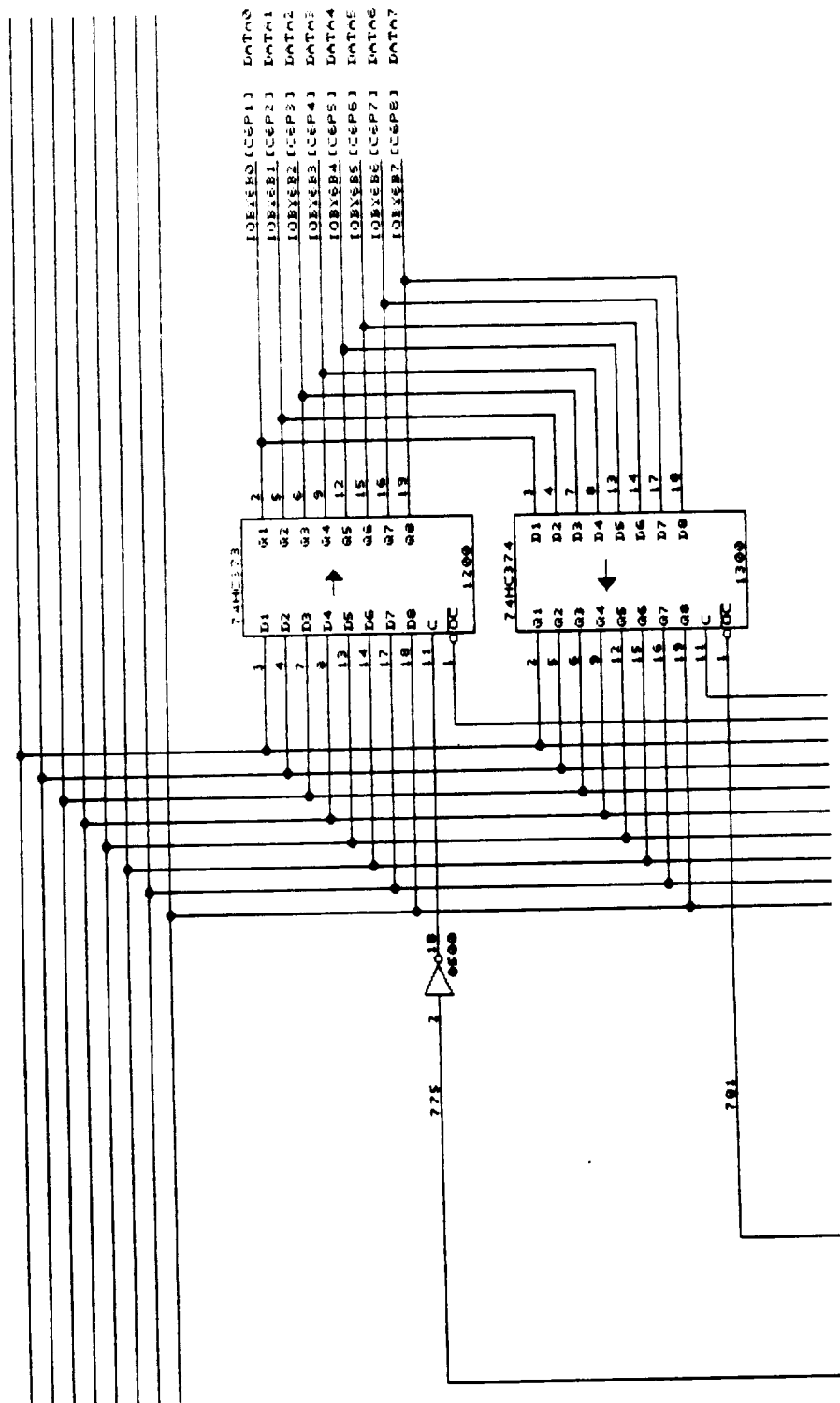


Figure 3

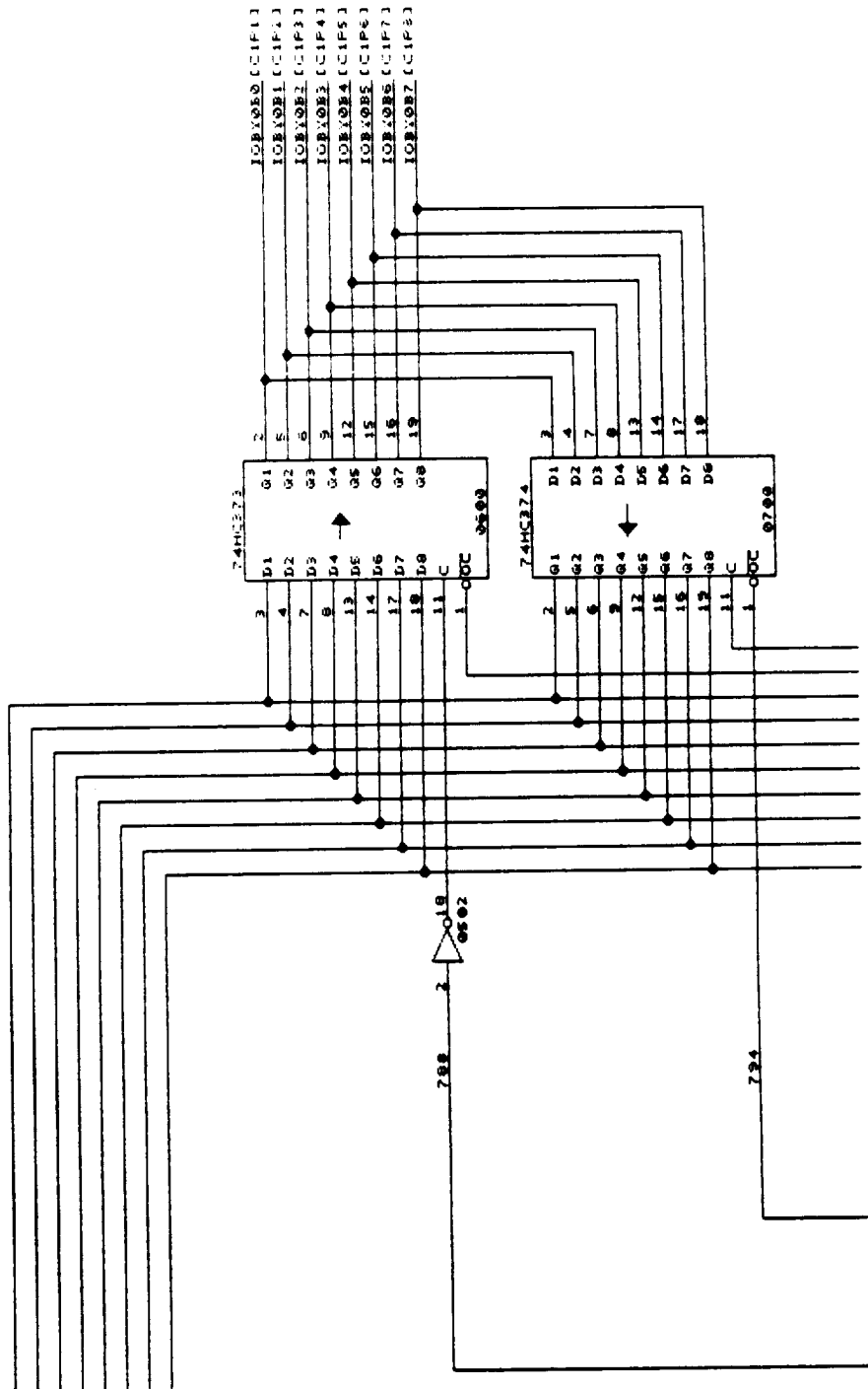


Figure 4

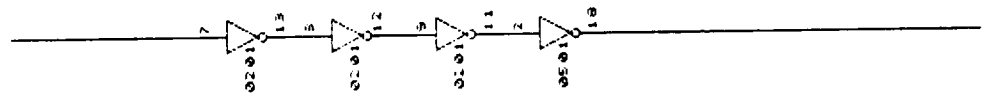


Figure 5

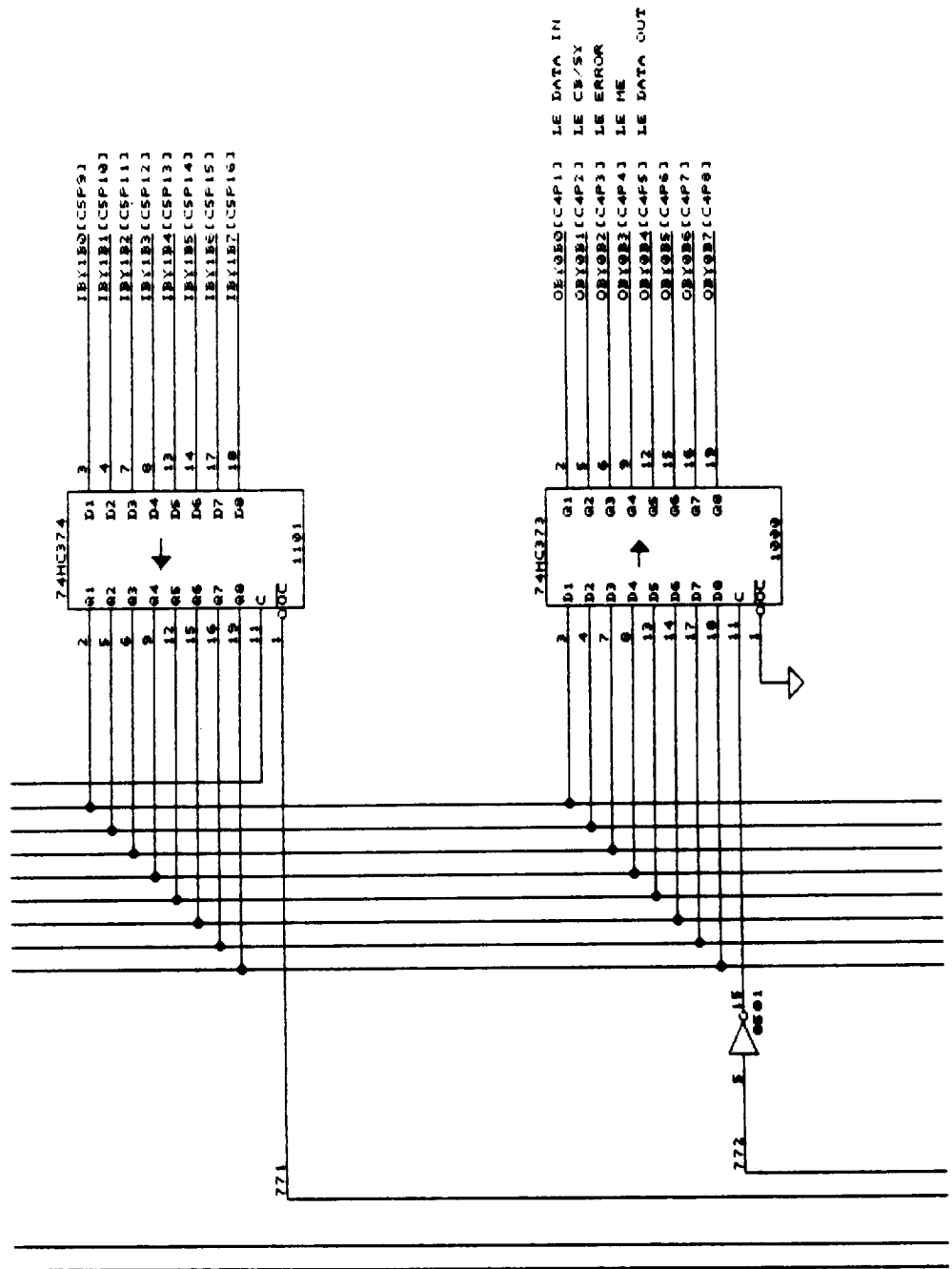


Figure 6

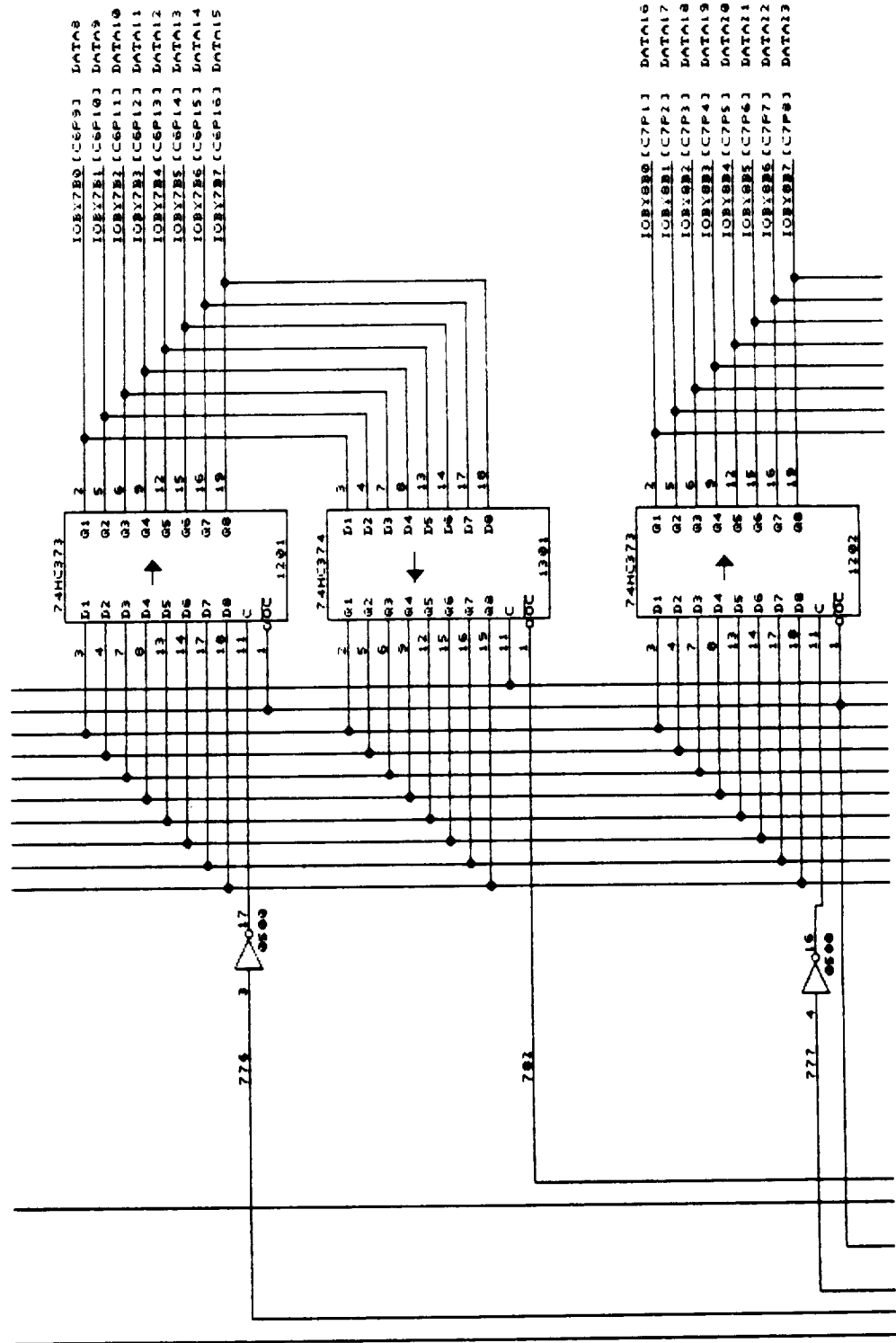


Figure 7

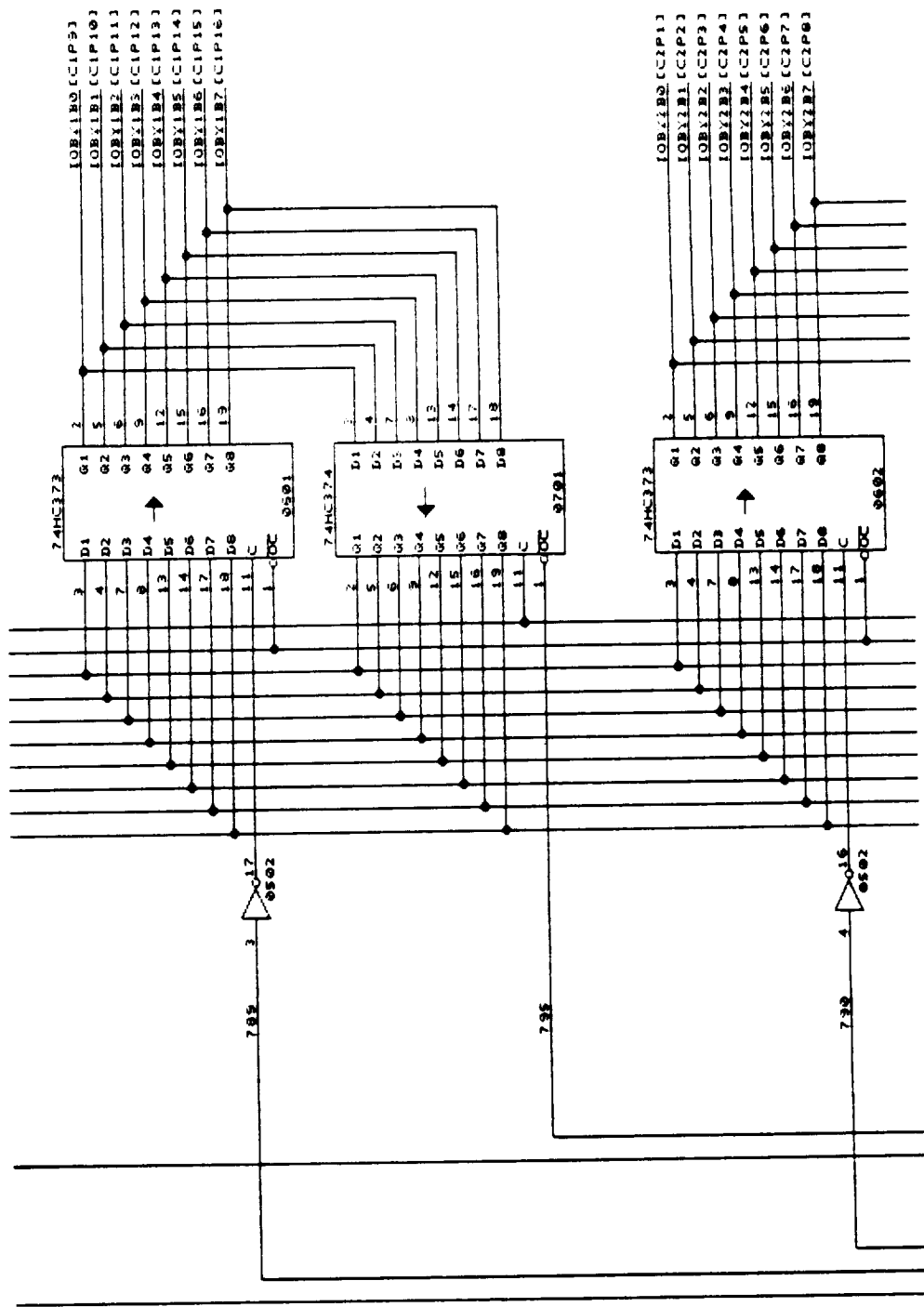


Figure 8

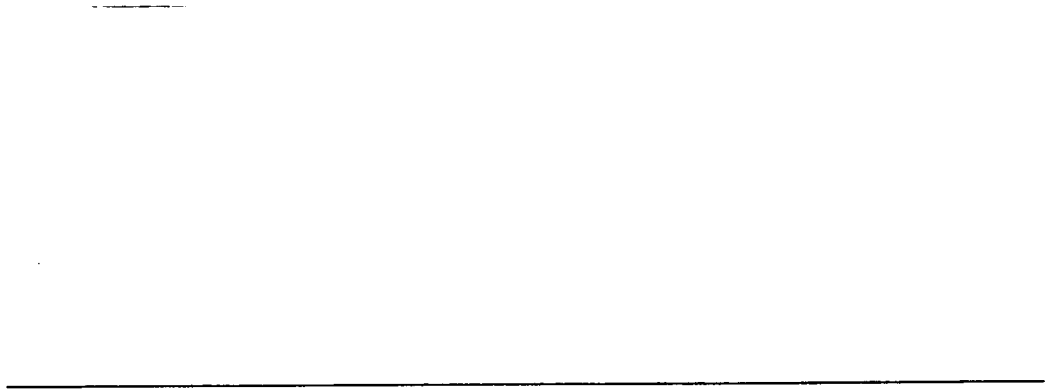


Figure 9

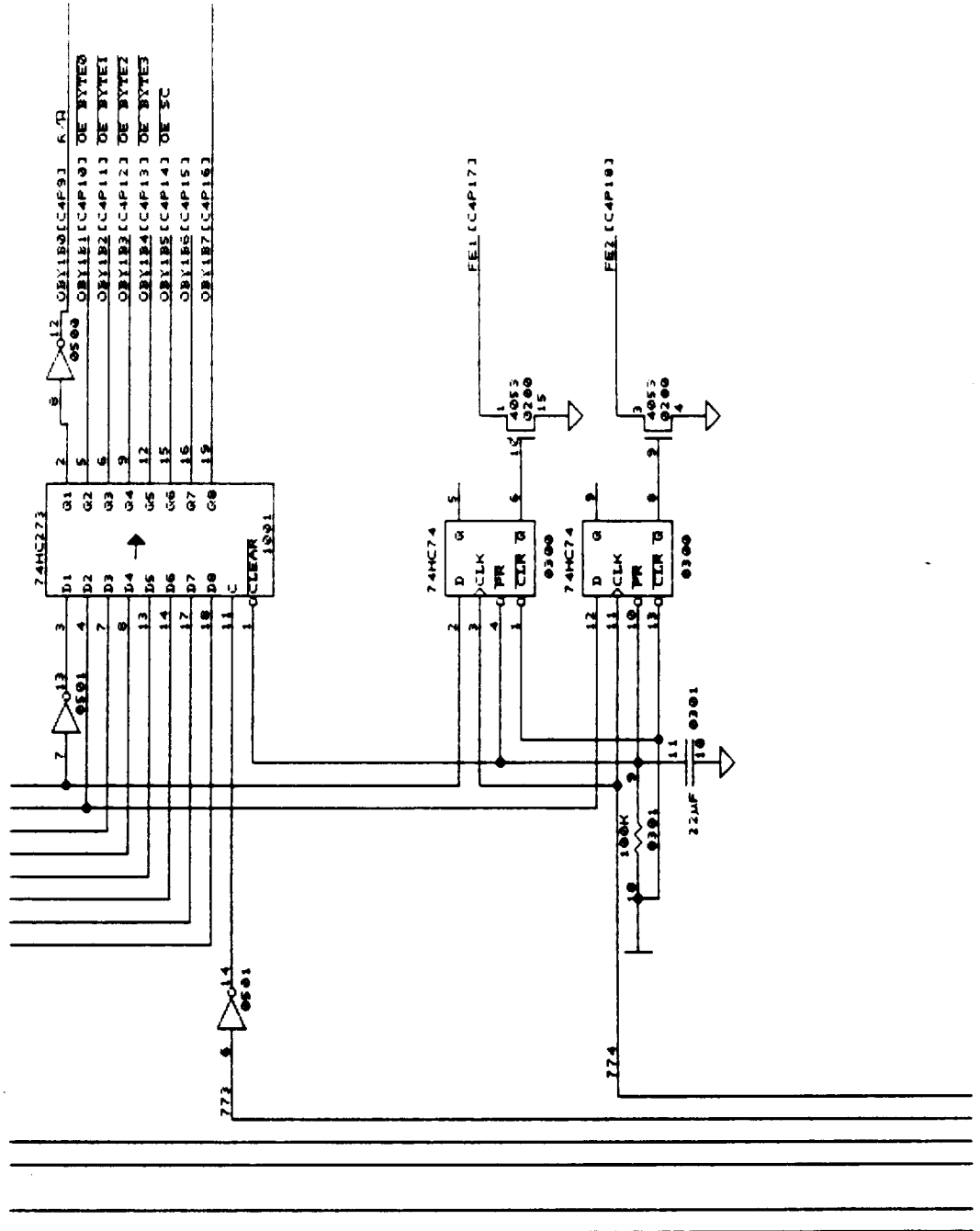


Figure 10

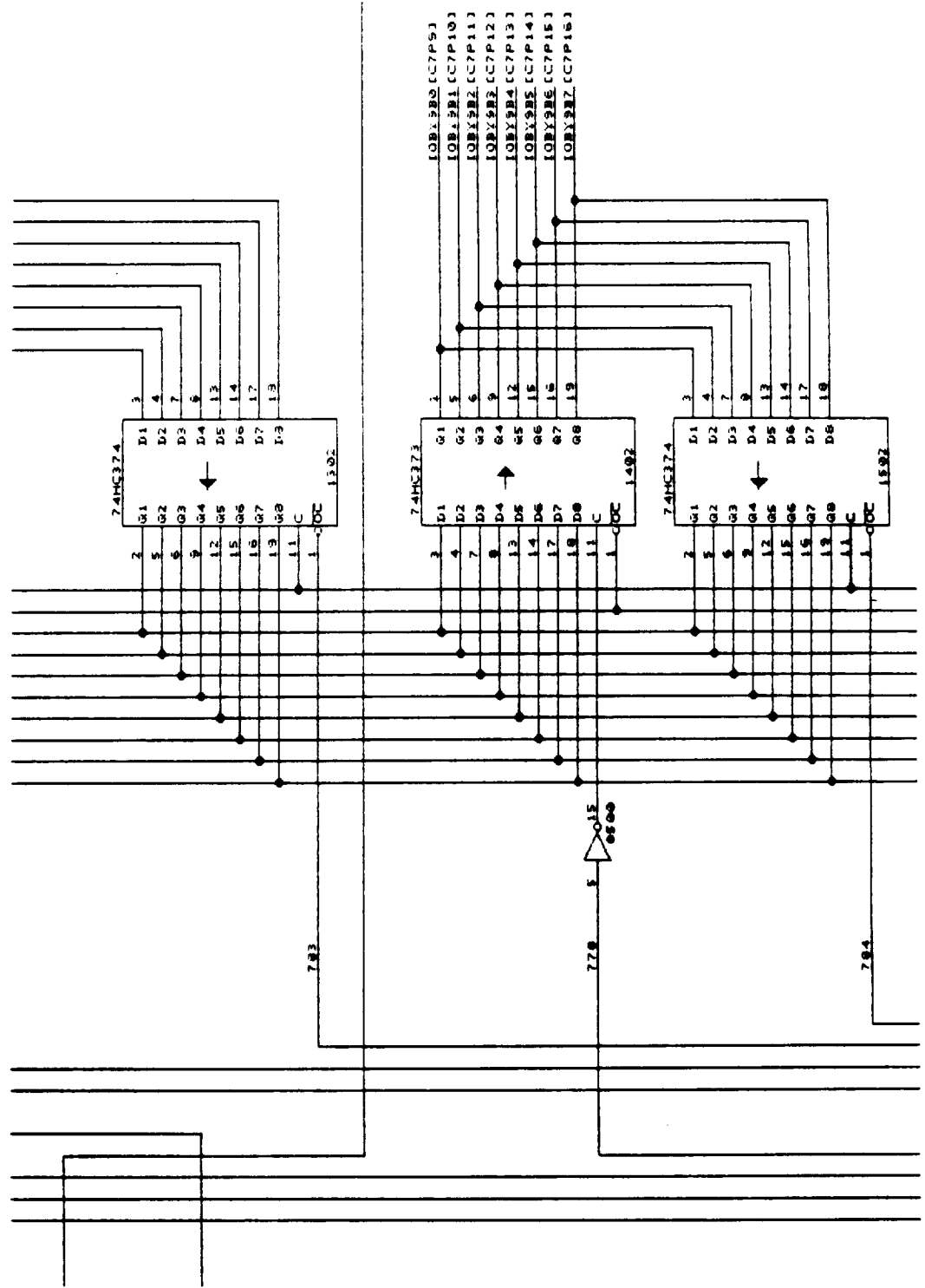


Figure 11

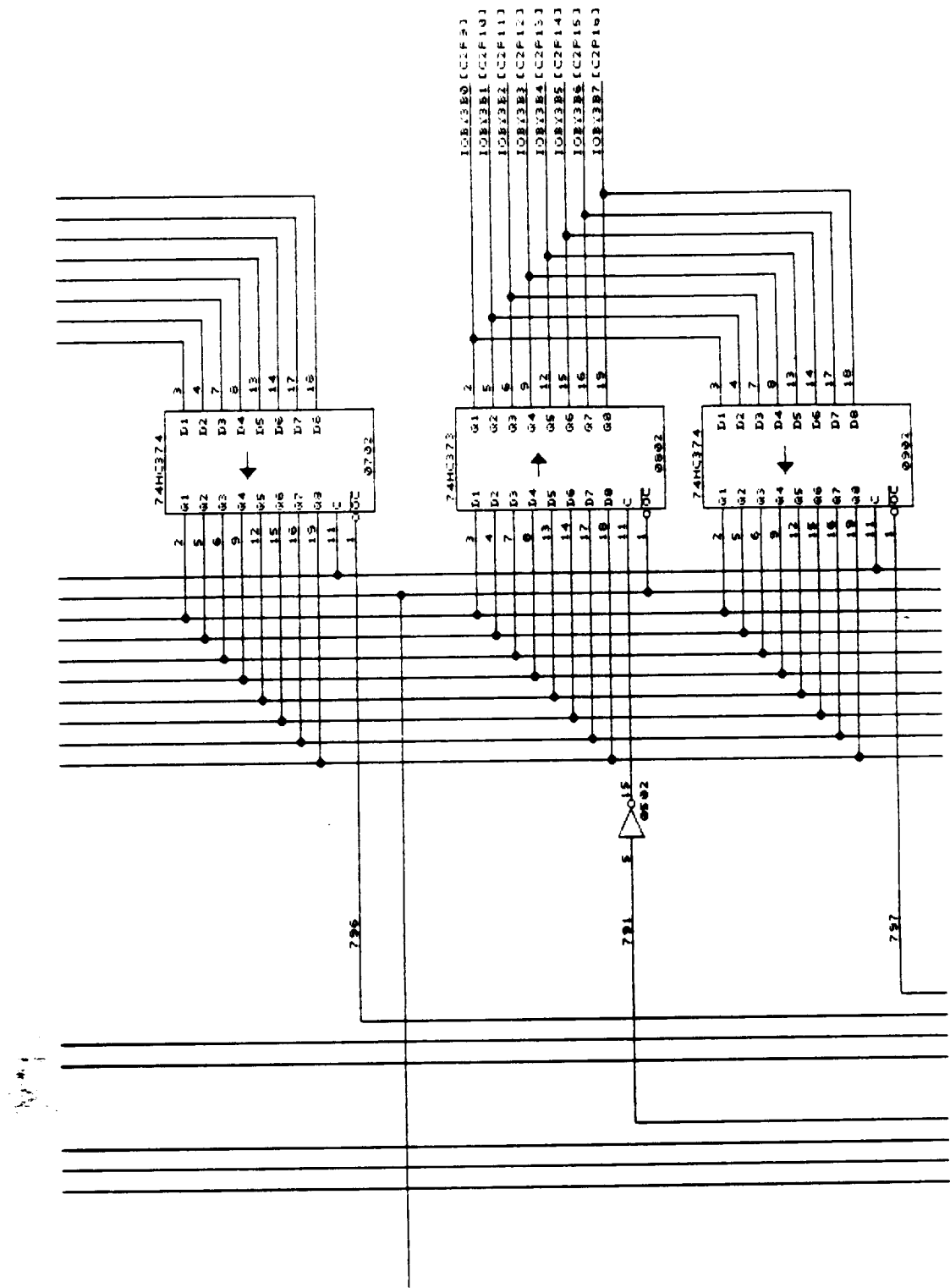


Figure 12

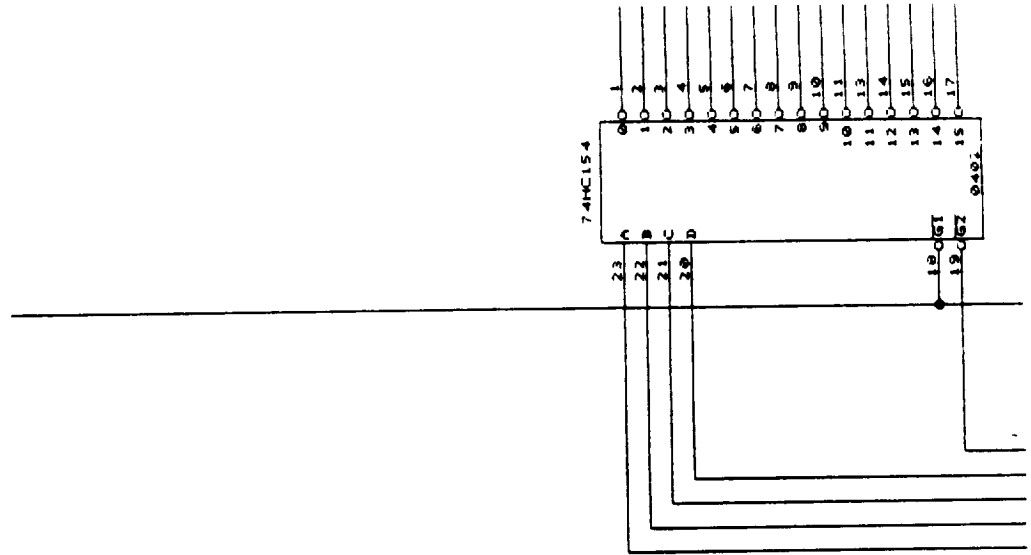


Figure 13

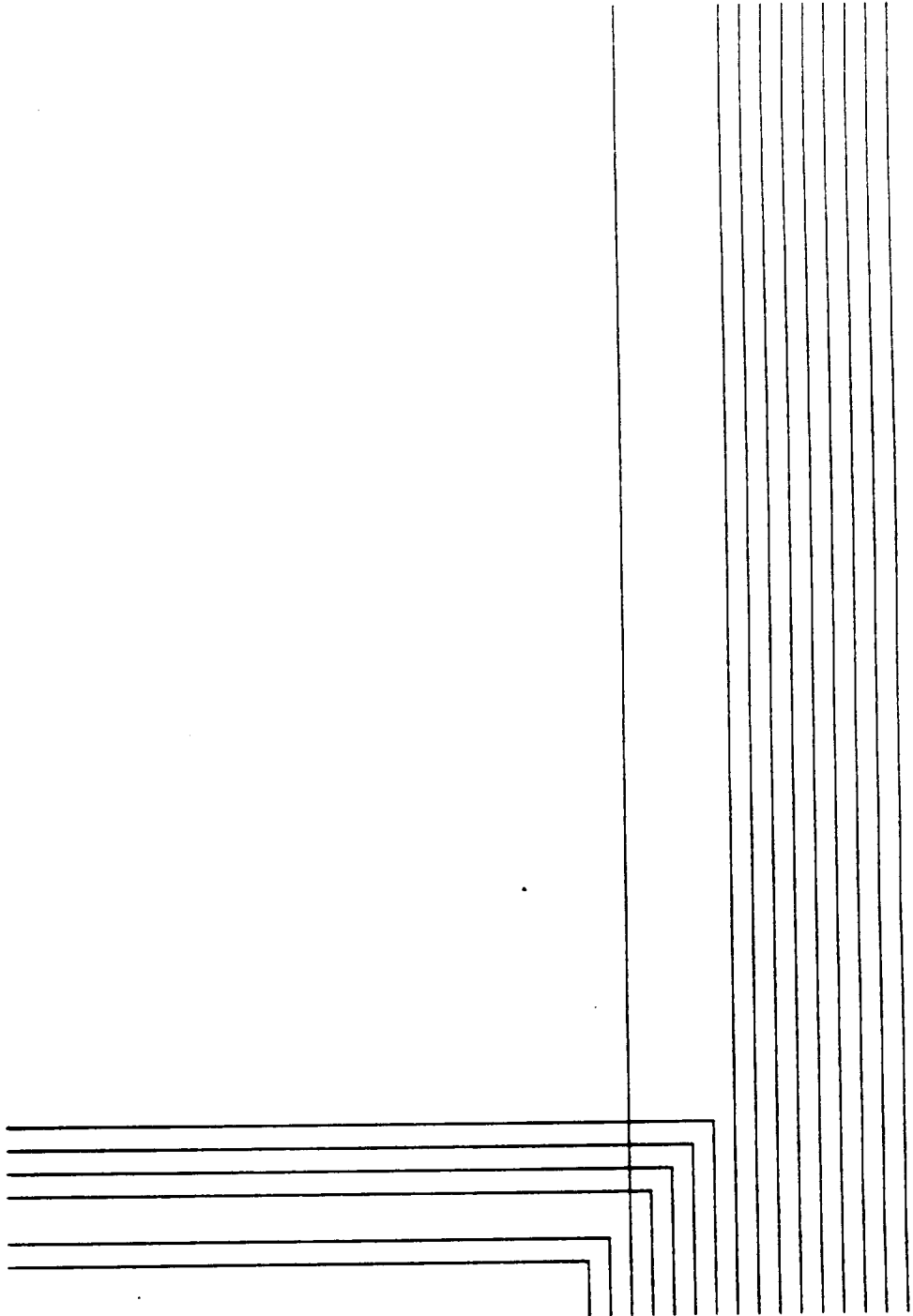


Figure 14

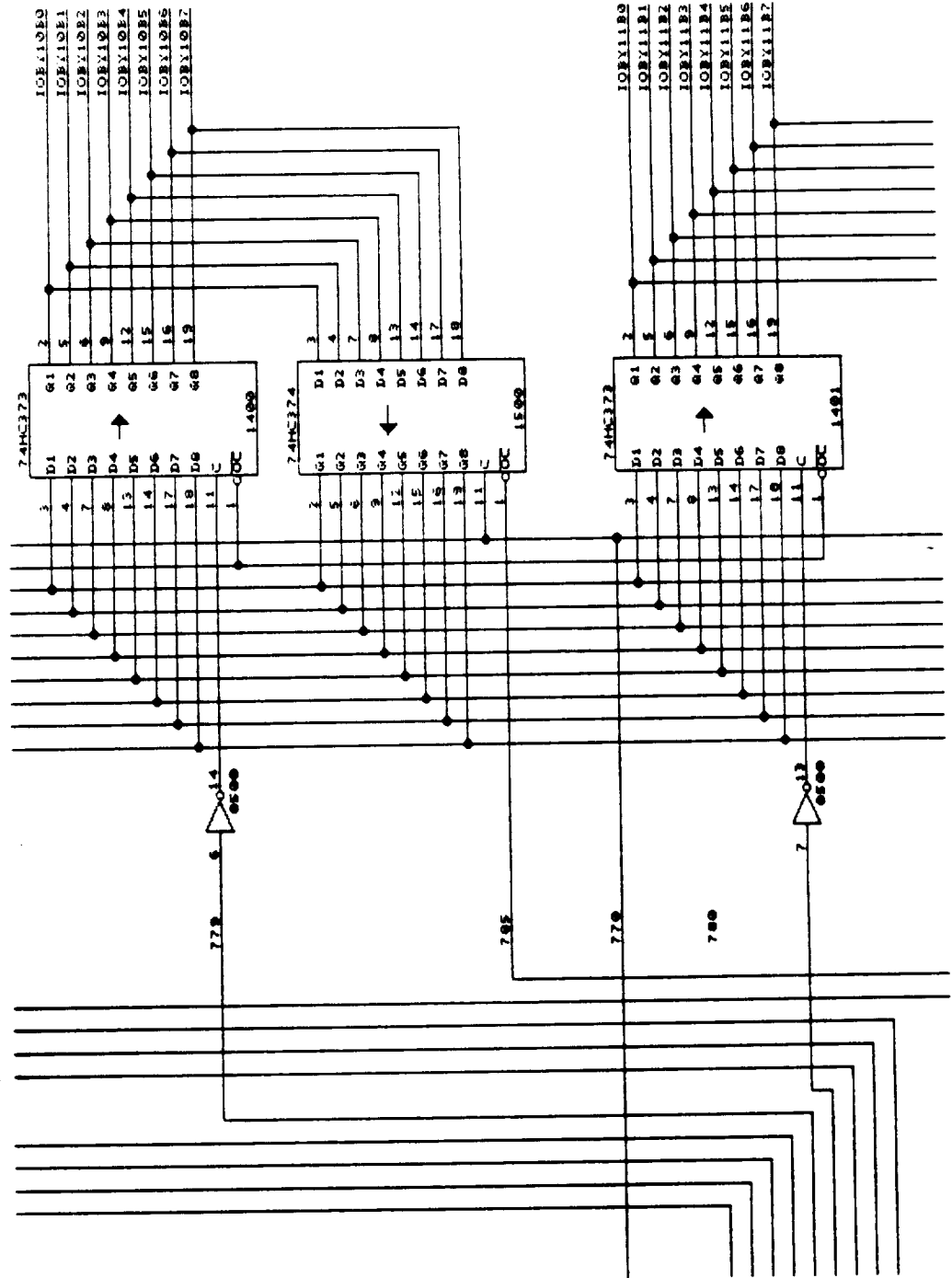


Figure 15

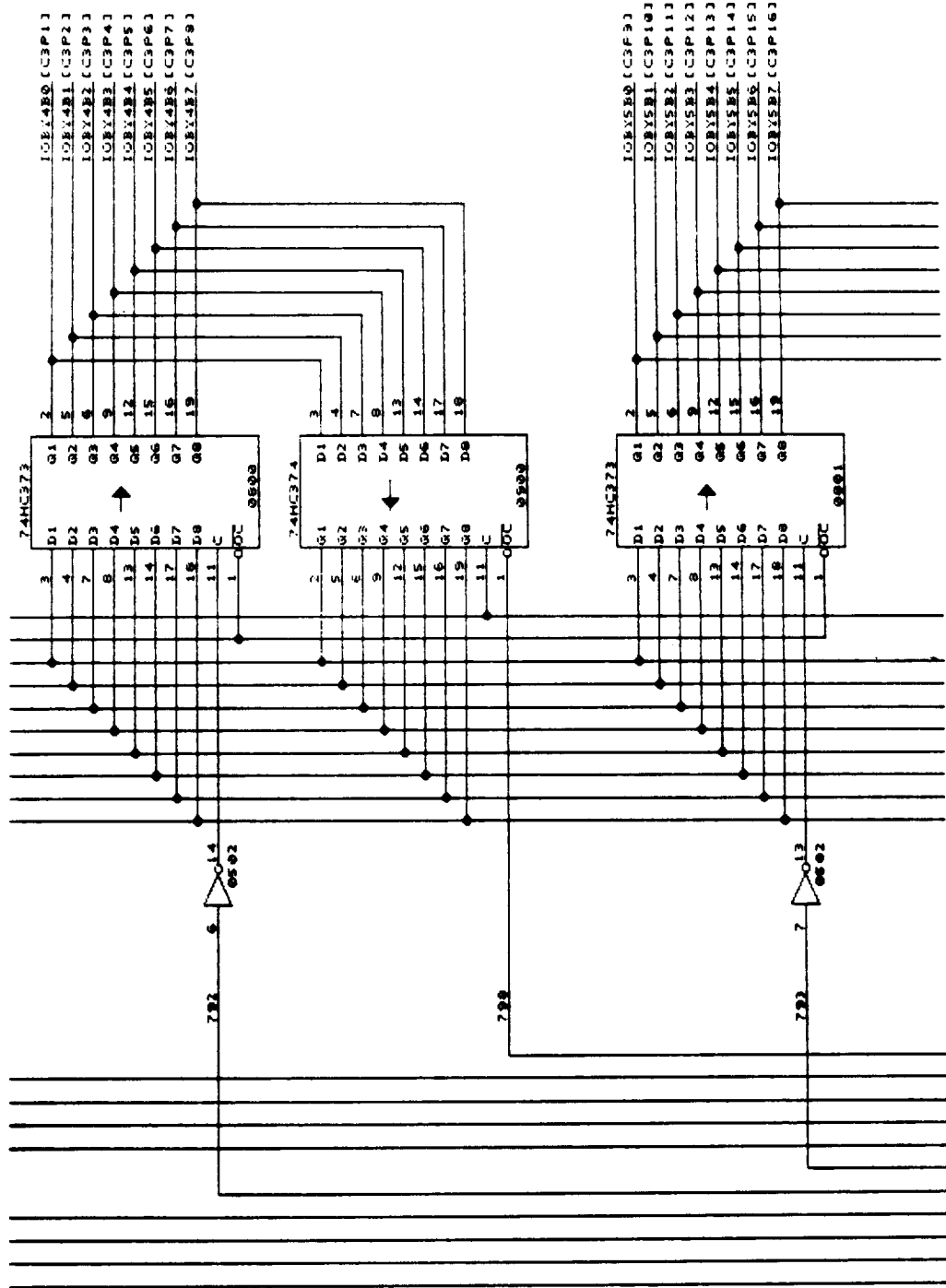


Figure 16

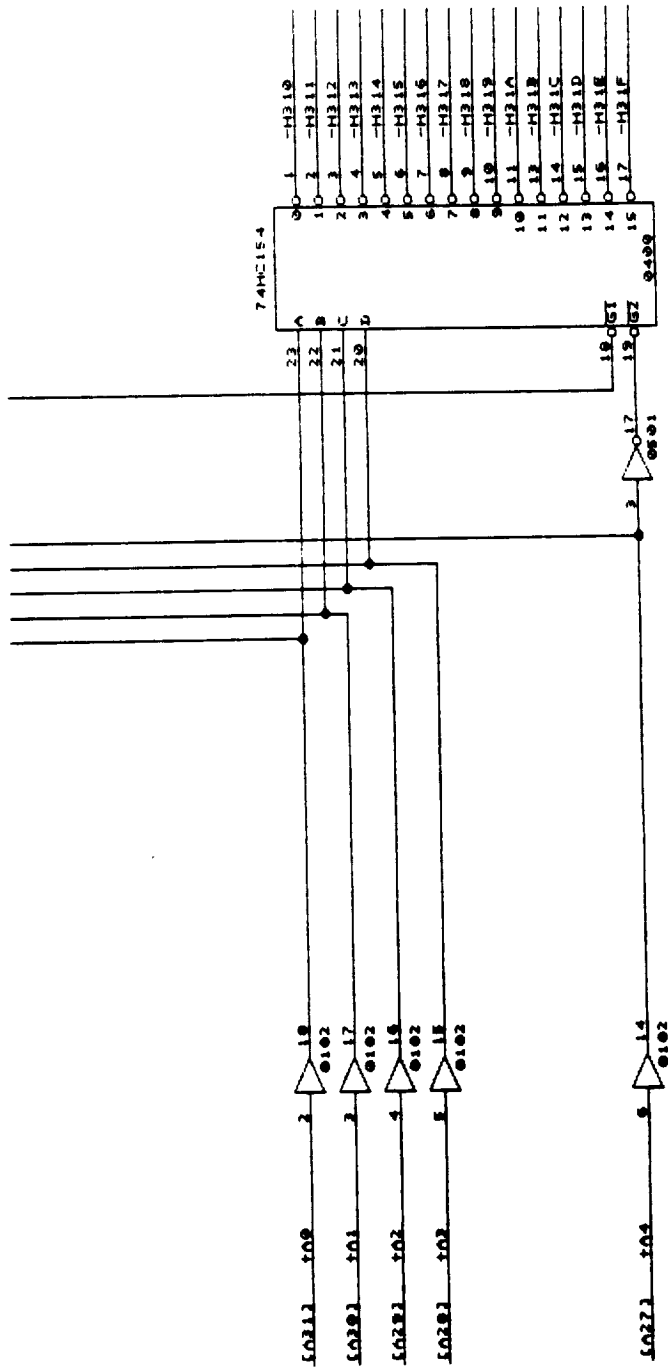


Figure 17

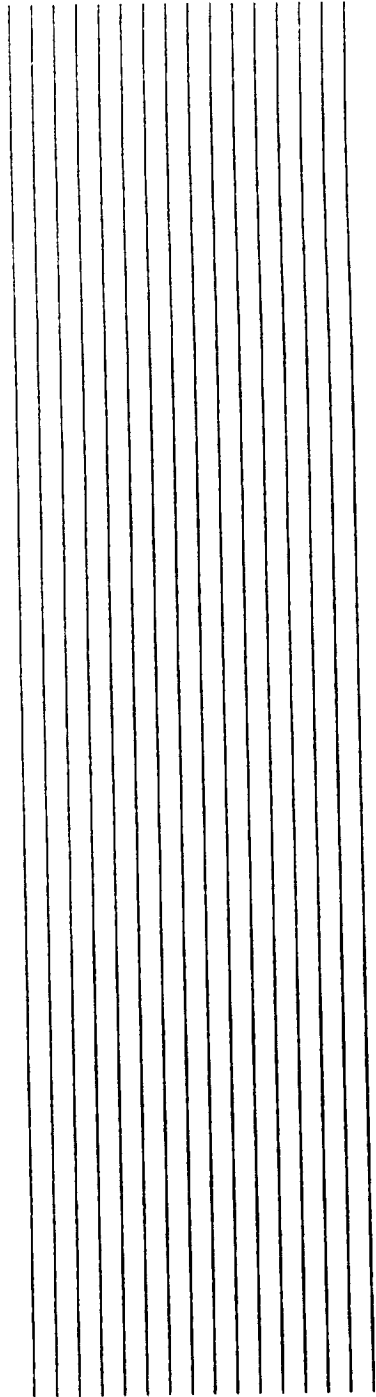


Figure 18

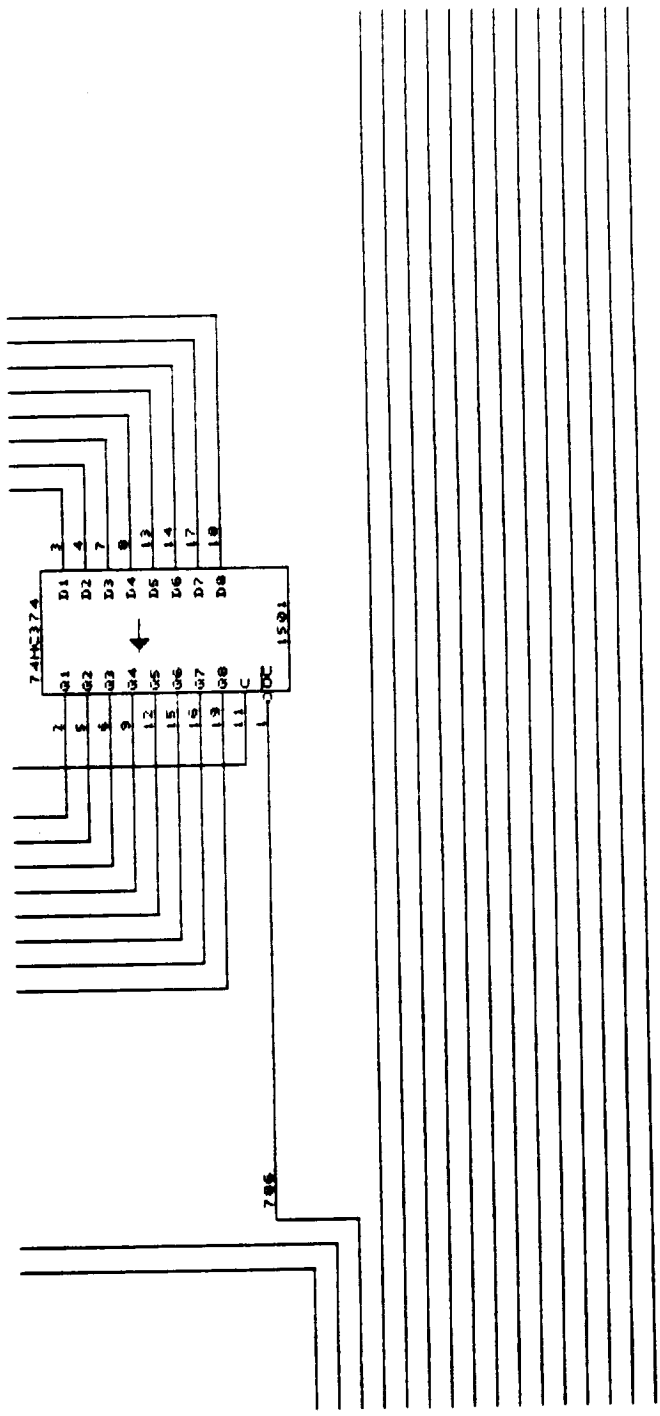


Figure 19

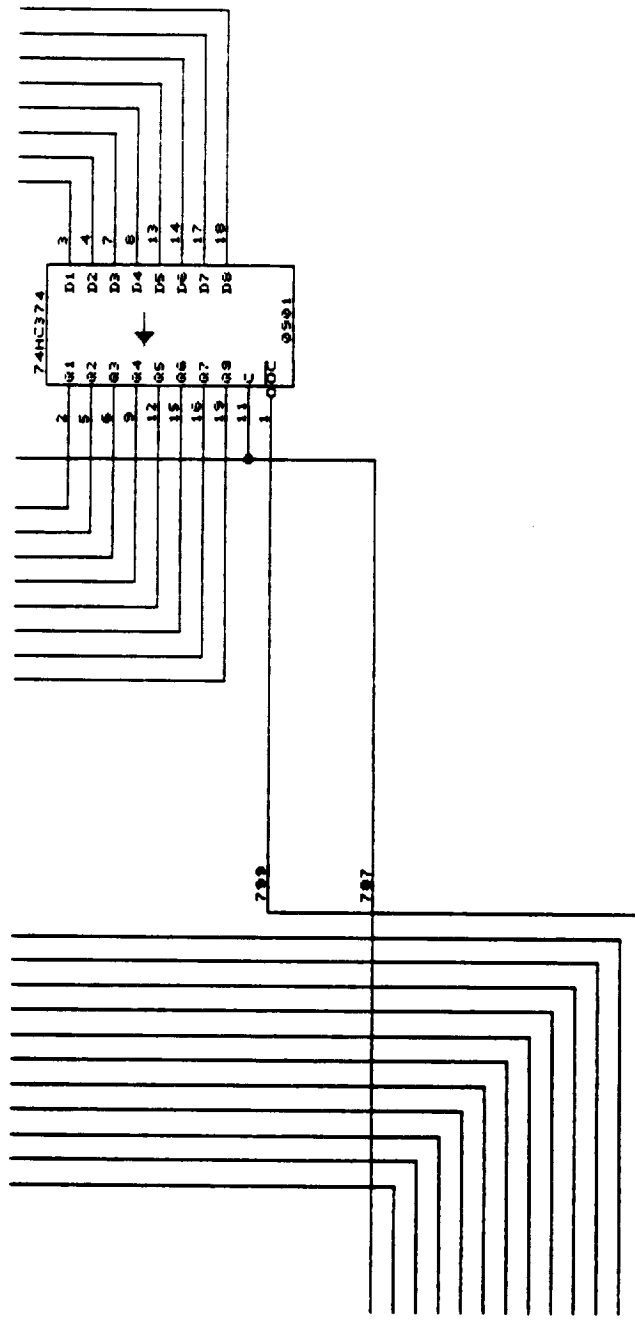


Figure 20