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) [redacted] p
373

N93-17771

Unclas

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 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.

TABLE OF CONTENTS

APPENDIXES

A INTRODUCTION

B ERROR DETECTION AND CORRECTION DESIGN

D.2 Microsimulator

D.3 Fault Simulation

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 CB0 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 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 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 posses, 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

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		X		X			X		X	X			X		X	X			X
CB1			X				X			X	X		X			X			X		X		X		X		X		X	X		X
CB2		X				X			X		X	X			X			X			X	X		X			X	X		X	X	
CB3	X				X				X	X				X	X			X	X	X				X	X	X				X		X
CB4					X	X	X	X						X		X	X							X	X	X	X	X	X		X	
CB5	X	X	X	X										X		X	X	X	X	X	X	X	X								X	
CB6	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]

PRES_15.DHG

SYNDROME DECODE

SYNDROME BITS																	
	S6	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	S5	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	S4	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0
	S3	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
S0 S1 S2																	
0 0 0		*	C5	T	C4	T	18	T	C3	T	31	T	27	T	M	T	C6
0 0 1		C2	T	17	T	8	T	14	T	23	T	M	T	26	T	30	T
0 1 1		T	11	T	5	T	M	T	2	T	M	T	M	T	M	T	21
0 1 0		C1	T	16	T	7	T	13	T	22	T	M	T	25	T	29	T
1 1 0		T	9	T	3	T	M	T	0	T	M	T	M	T	M	T	19
1 1 1		M	T	M	T	M	T	M	T	M	T	M	T	M	T	M	T
1 0 1		T	10	T	4	T	M	T	1	T	M	T	M	T	M	T	20
1 0 0		C0	T	15	T	6	T	12	T	M	T	M	T	24	T	28	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

PRES_16.DWG

THREE OR MORE THAN TWO ERRORS (TOME)

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

$$\begin{aligned} TOME = {} & S0.S3.S6 + S0.S1.S2 + S4.S5.S6 + S0.S3.S4.S5 \\ & + S1.S3.S4.S5 + S1.S3.S6\ (\ S4 + S5\ ) \end{aligned}$$

PRES_17.DWG

LOGIC EQUATIONS

$$\mathrm{ERROR} = S0 + S1 + S2 + S3 + S4 + S5 + S6$$

$$\overline{\mathrm{ERROR}} = \overline{S0}.\overline{S1}.\overline{S2}.\overline{S3}.\overline{S4}.\overline{S5}.\overline{S6}$$

$$\overline{\mathrm{MULT\ ERROR}} = (\ \overline{\overline{S0 \oplus S1 \oplus S2 \oplus S3 \oplus S4 \oplus S5 \oplus S6}\ ).(\mathrm{ERROR}) + \mathrm{TOME}}$$

$$\begin{aligned}\mathrm{TOME} = {} & S0.S3.S6 + S0.S1.S2 + S4.S5.S6 + S0.S3.S4.S5 \\ & + S1.S3.S4.S5 + S1.S3.S6\ (\ S4 + S5\ )\end{aligned}$$

PRES_10.DWG

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= 50pF)	100nsec

PRES_03.DWG

EDAC UNIT DESIGN APPROACH

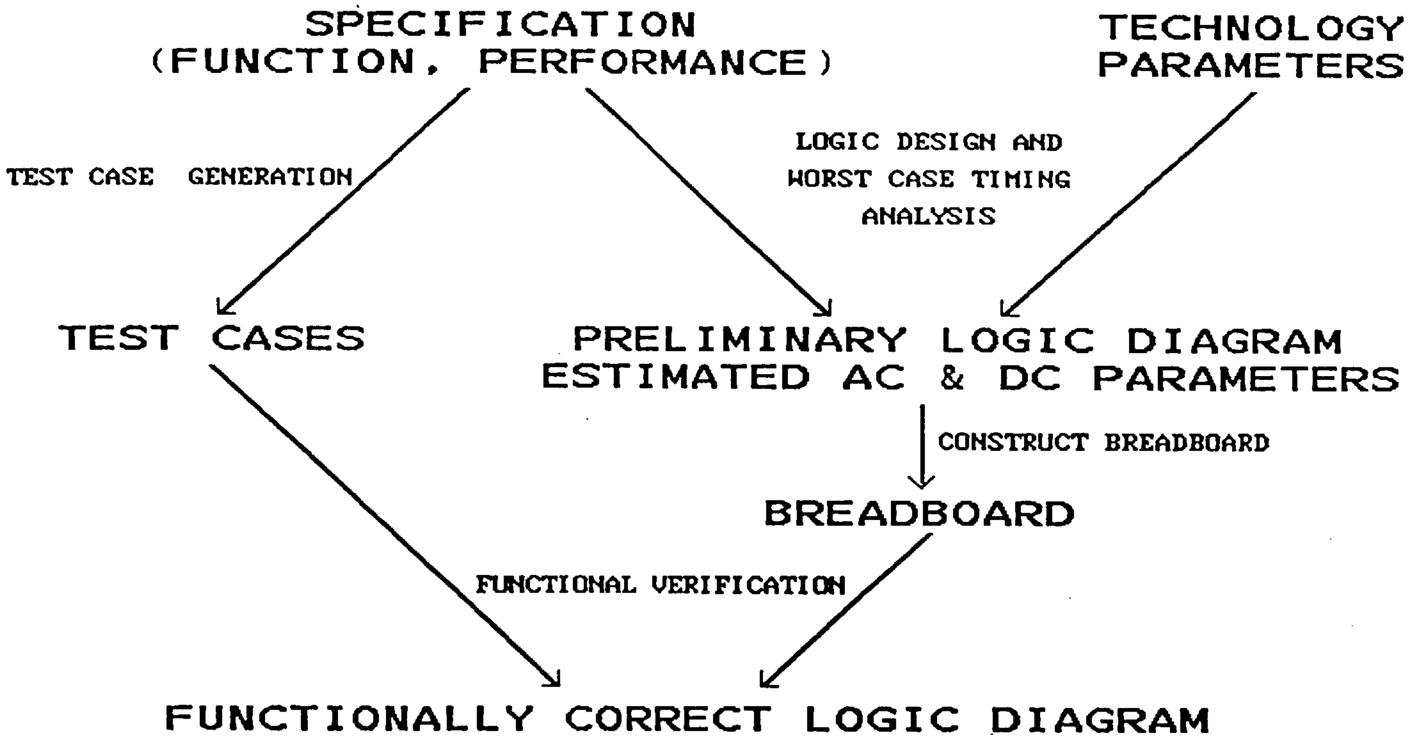


Figure 1

EDAC UNIT DESIGN APPROACH

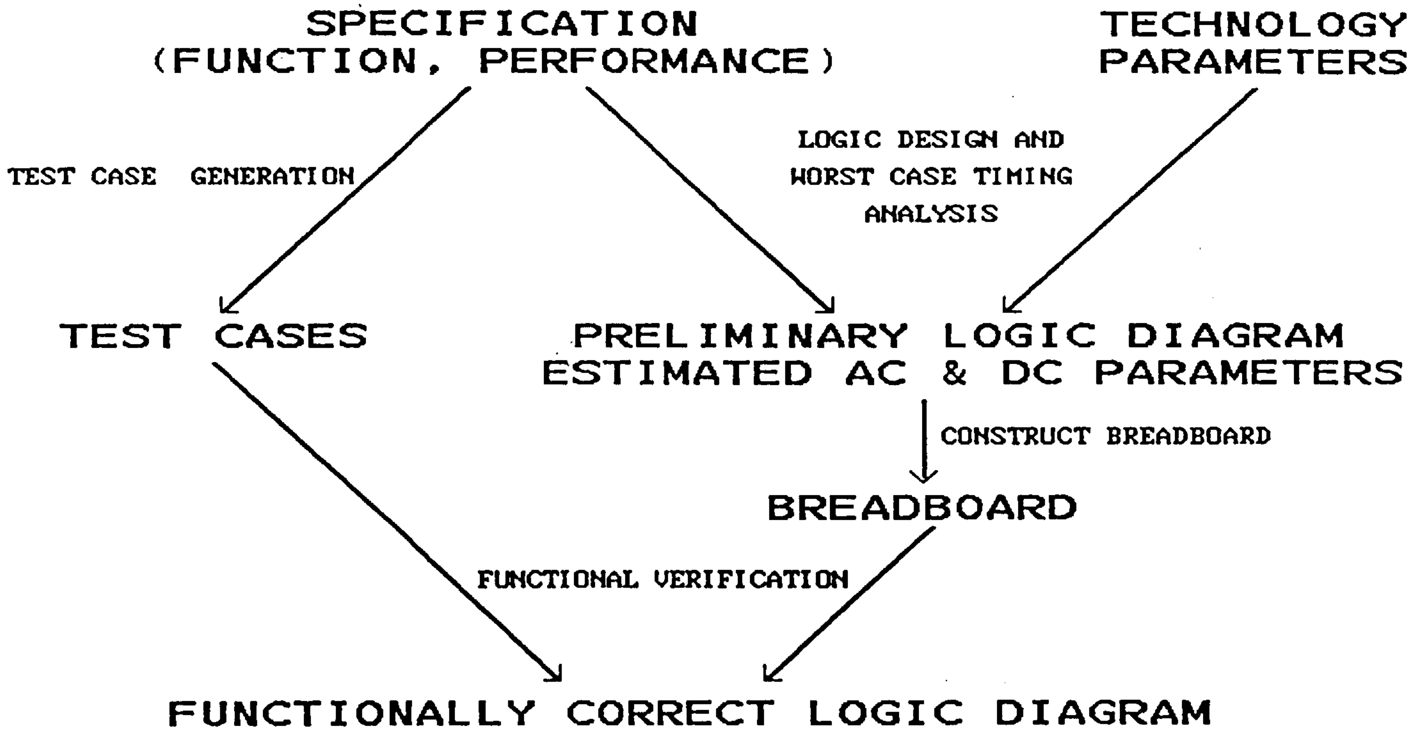
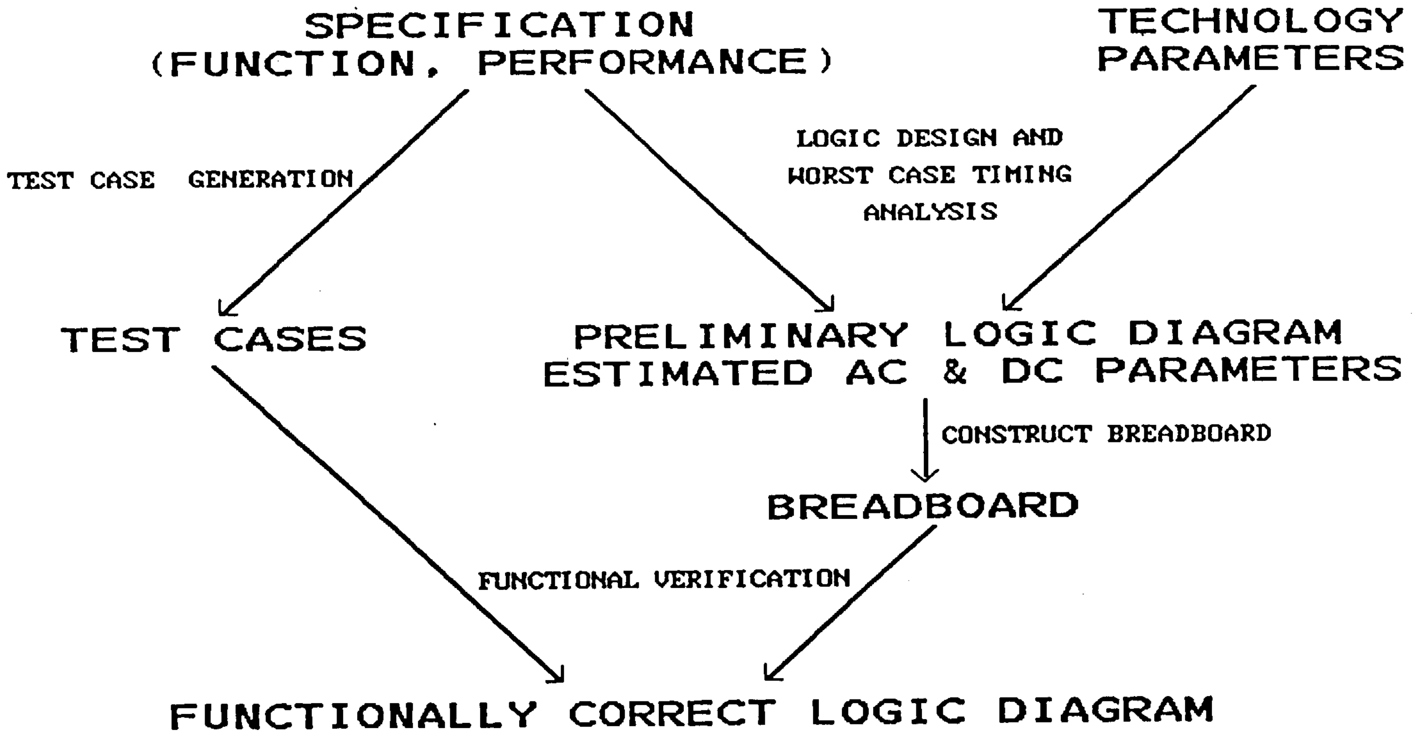

SPECIFICATION (FUNCTION, PERFORMANCE)
TECHNOLOGY PARAMETERS
TEST CASE GENERATION
LOGIC DESIGN AND WORST CASE TIMING ANALYSIS
TEST CASES
PRELIMINARY LOGIC DIAGRAM ESTIMATED AC & DC PARAMETERS
CONSTRUCT BREADBOARD
BREADBOARD
FUNCTIONAL VERIFICATION
FUNCTIONALLY CORRECT LOGIC DIAGRAM

Figure 2

EDAC UNIT DESIGN APPROACH



SPECIFICATION (FUNCTION, PERFORMANCE)

TECHNOLOGY PARAMETERS

TEST CASE GENERATION

LOGIC DESIGN AND WORST CASE TIMING ANALYSIS

TEST CASES

PRELIMINARY LOGIC DIAGRAM ESTIMATED AC & DC PARAMETERS

CONSTRUCT BREADBOARD

BREADBOARD

FUNCTIONAL VERIFICATION

FUNCTIONALLY CORRECT LOGIC DIAGRAM

Figure 3

EDAC UNIT DESIGN APPROACH

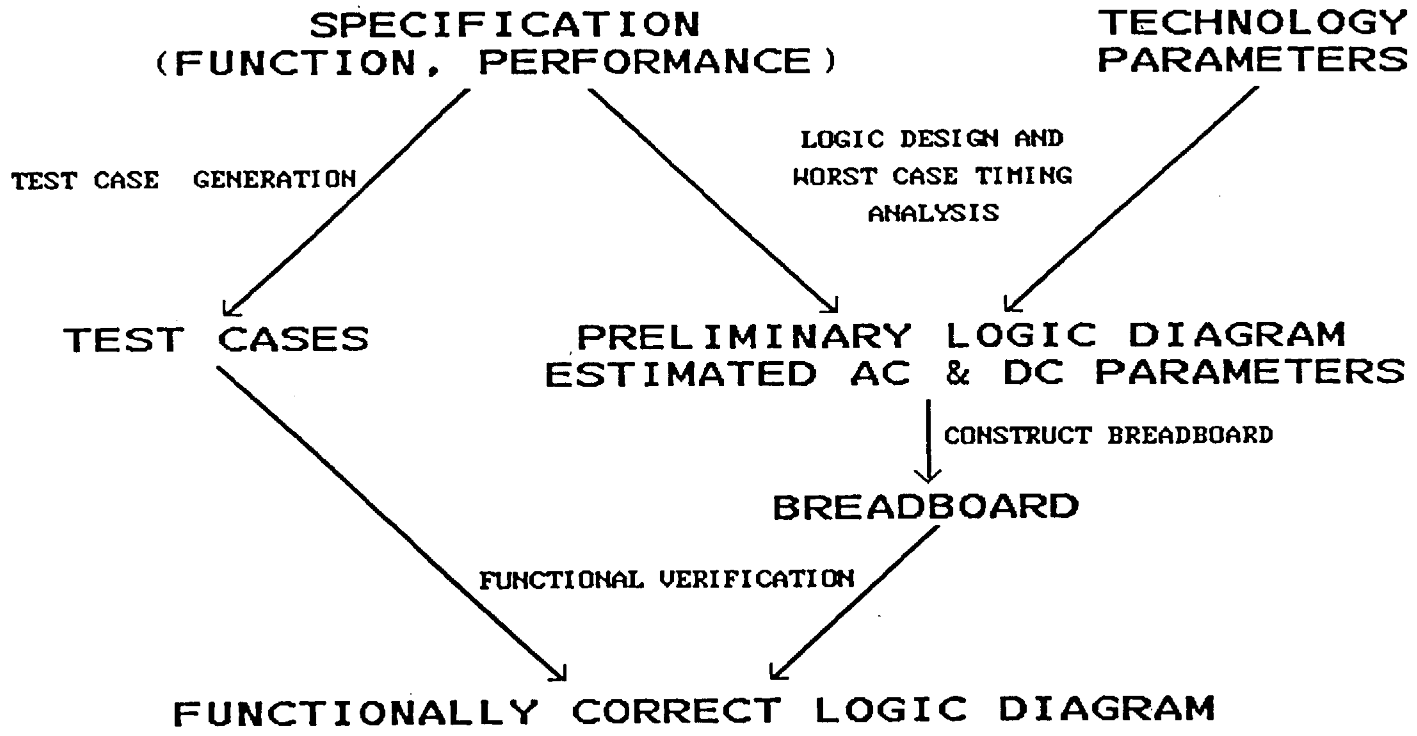


Figure 4

EDAC UNIT DESIGN APPROACH

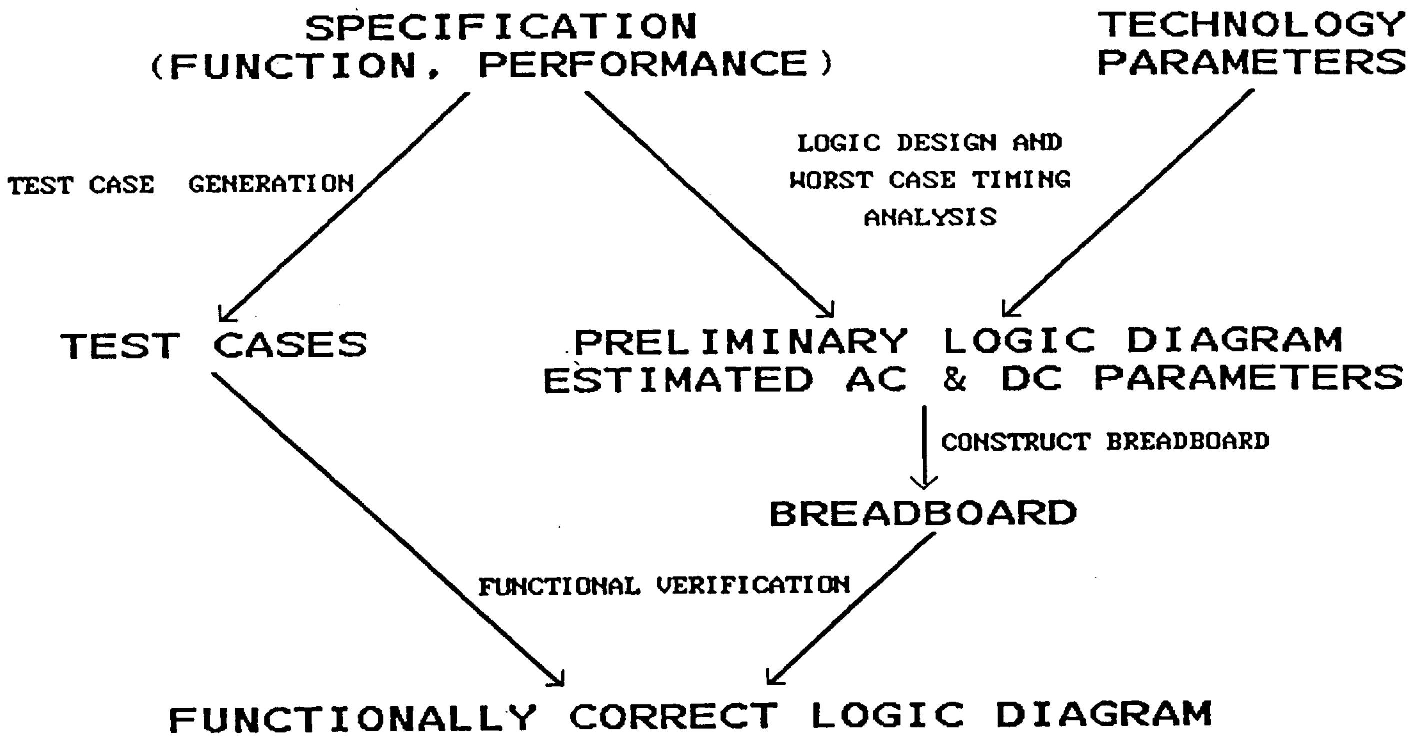


Figure 5

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

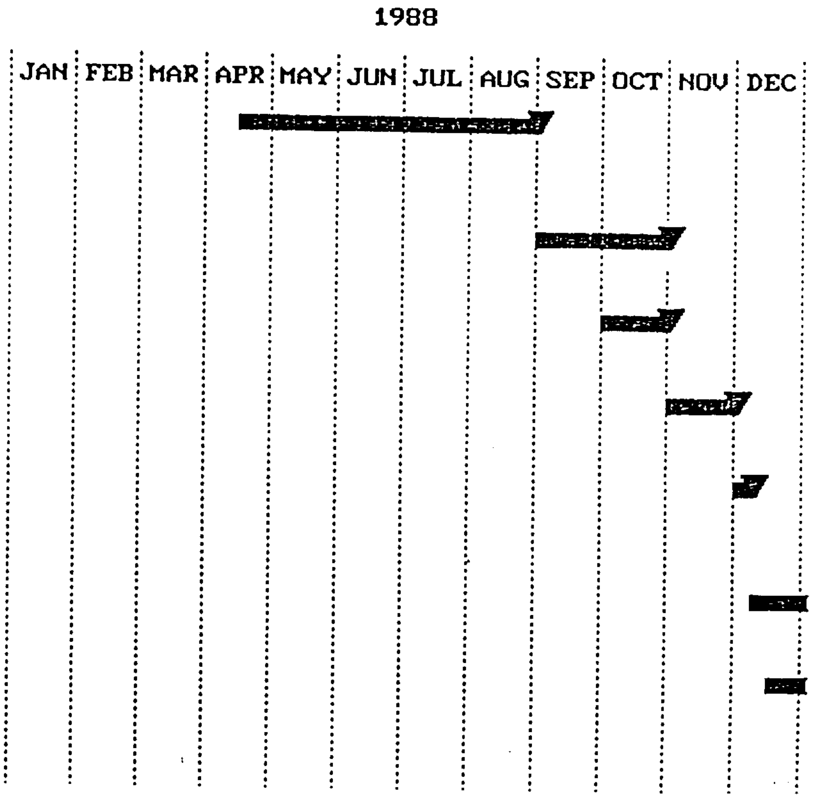
Status as of: 28 November 1989
Creation Date: 18 April 1988

Achievement Responsibility: Constantin Timoc

SBIR SCHEDULE

TASK

1. Logic design and worst case timing analysis
2. Construct breadboard
3. Test case generation
4. Functional verification
5. Replace standard chips with microsimulator chips
6. Test generation, fault simulation
7. Modify built-in self-test circuit in the functionally correct logic diagram to accomplish >95% fault coverage

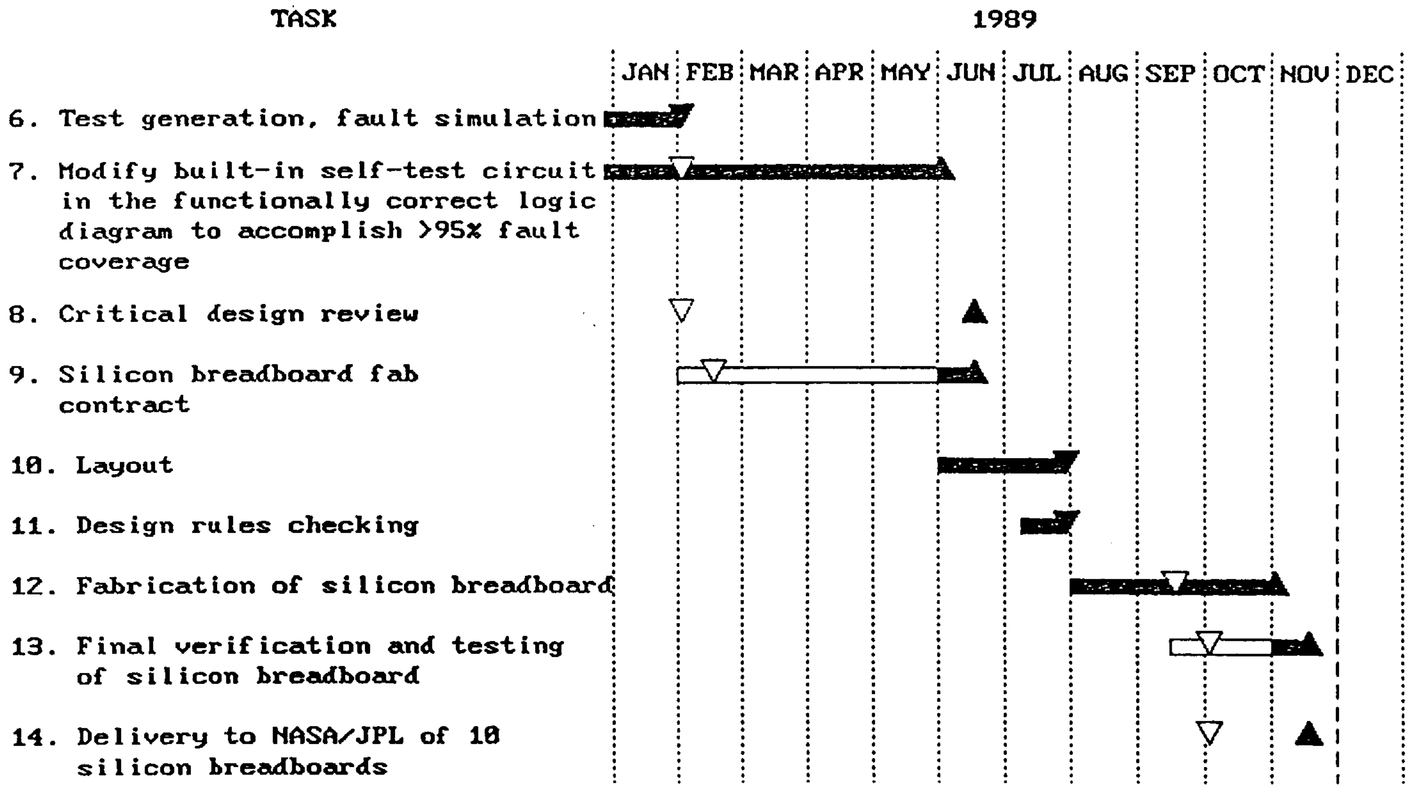


EDCSCH1.DWG

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

Map of EDAC Block Diagram

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

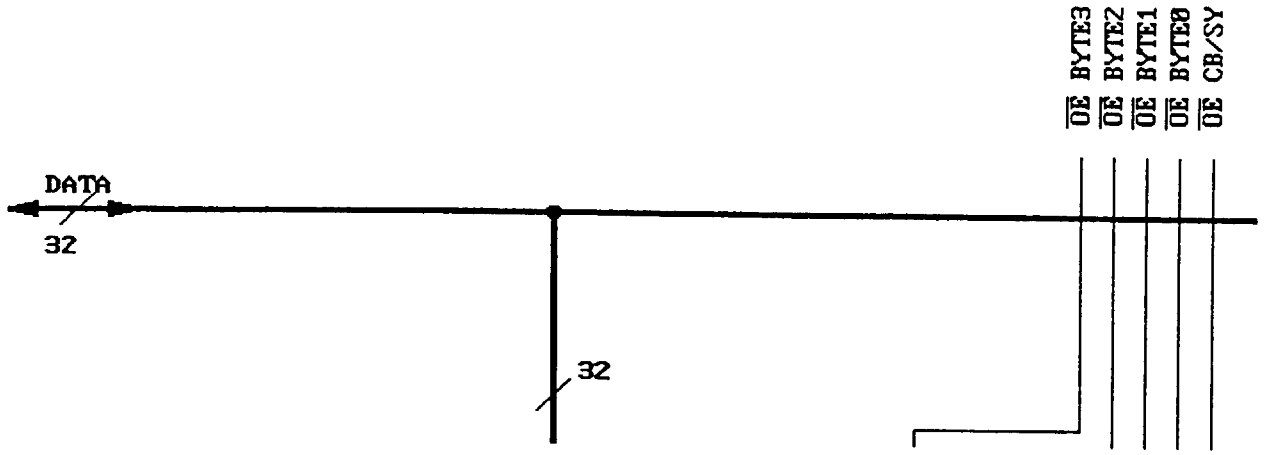


Figure 1

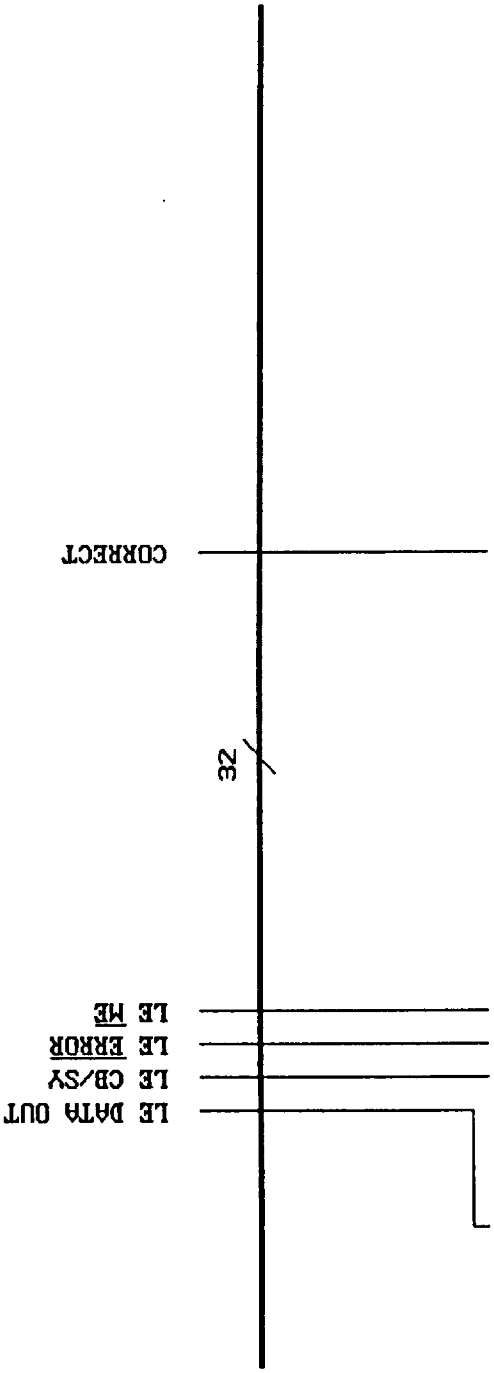


Figure 2

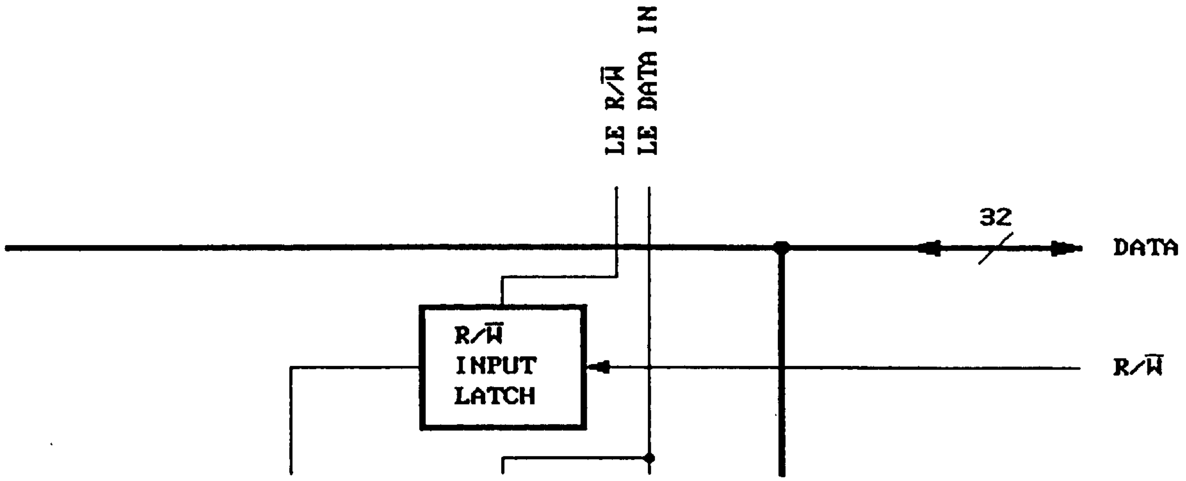

LE R/$\overline{W}$
LE DATA IN
32
DATA
R/$\overline{W}$
R/$\overline{W}$ INPUT LATCH

Figure 3

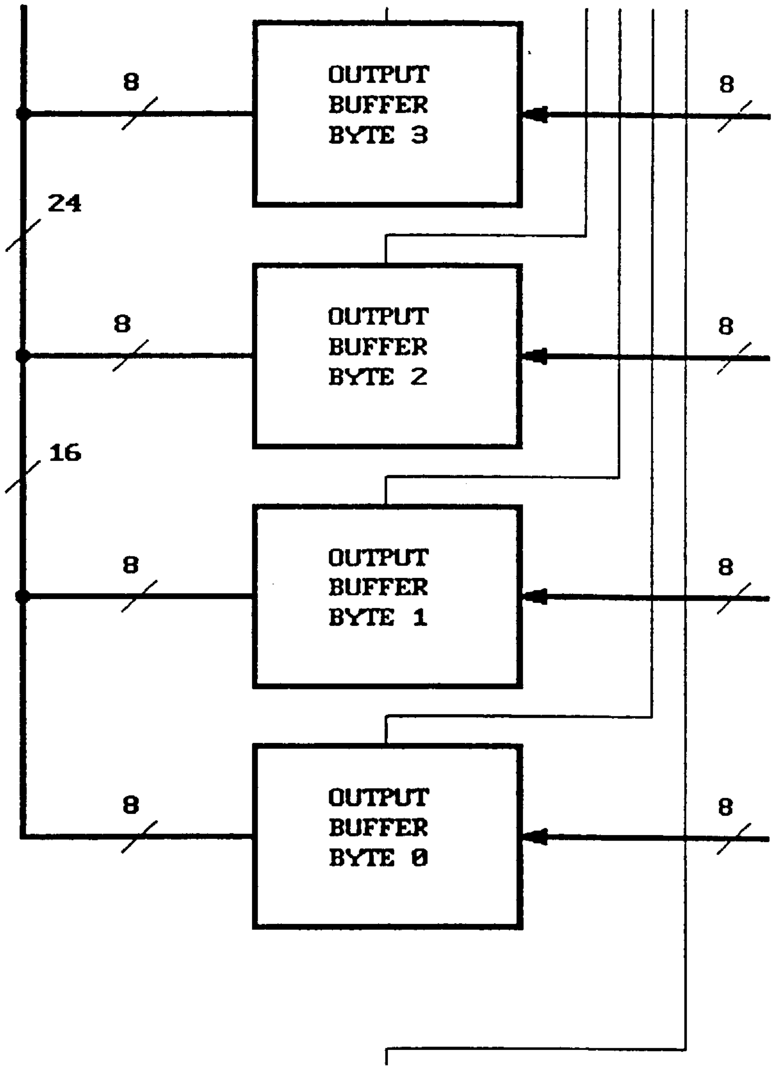

8
8
8
8
OUTPUT
BUFFER
BYTE 3
OUTPUT
BUFFER
BYTE 2
OUTPUT
BUFFER
BYTE 1
OUTPUT
BUFFER
BYTE 0
8
8
8
8
24
16

Figure 4

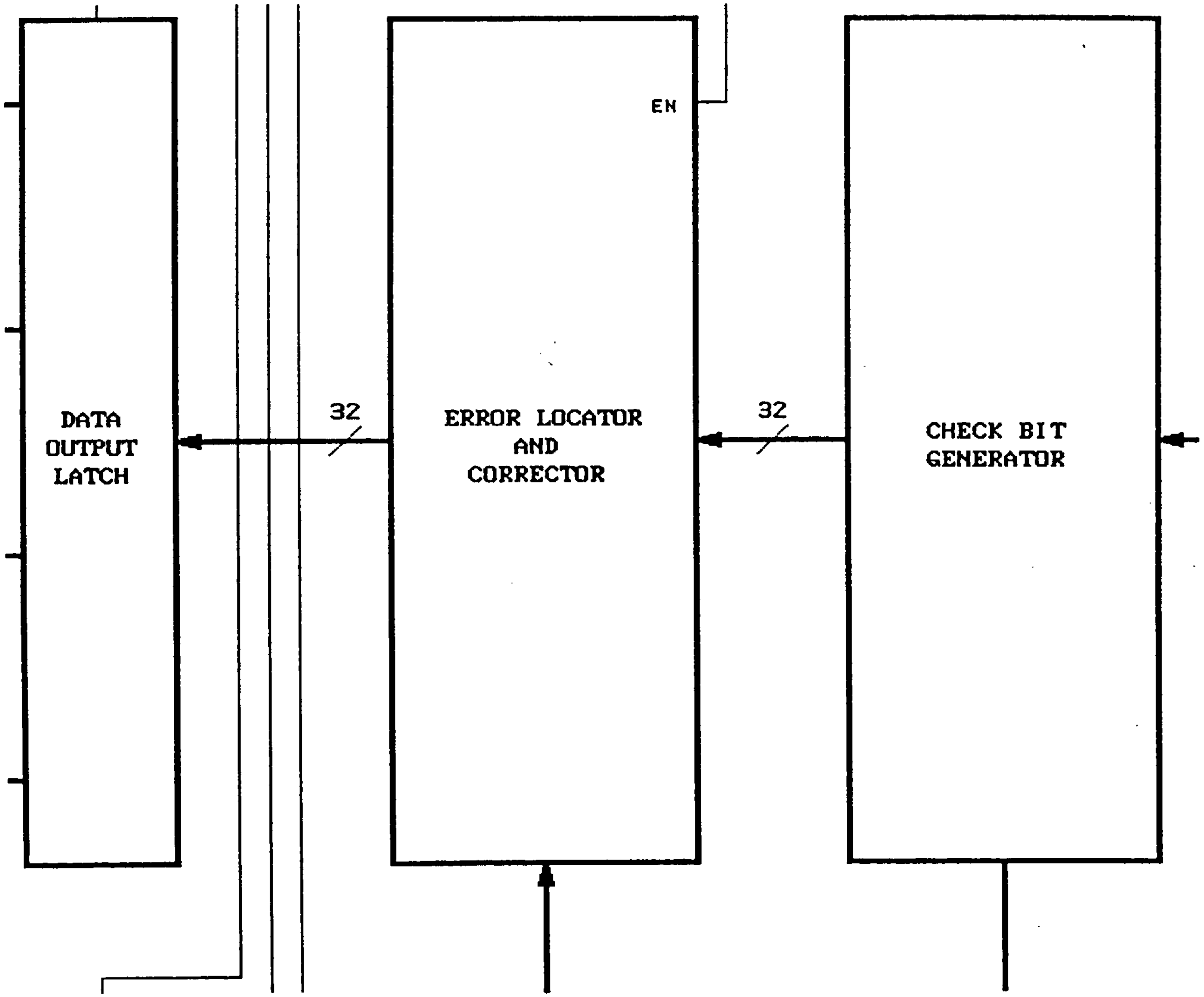

DATA
OUTPUT
LATCH
32
ERROR LOCATOR
AND
CORRECTOR
EN
32
CHECK BIT
GENERATOR

Figure 5

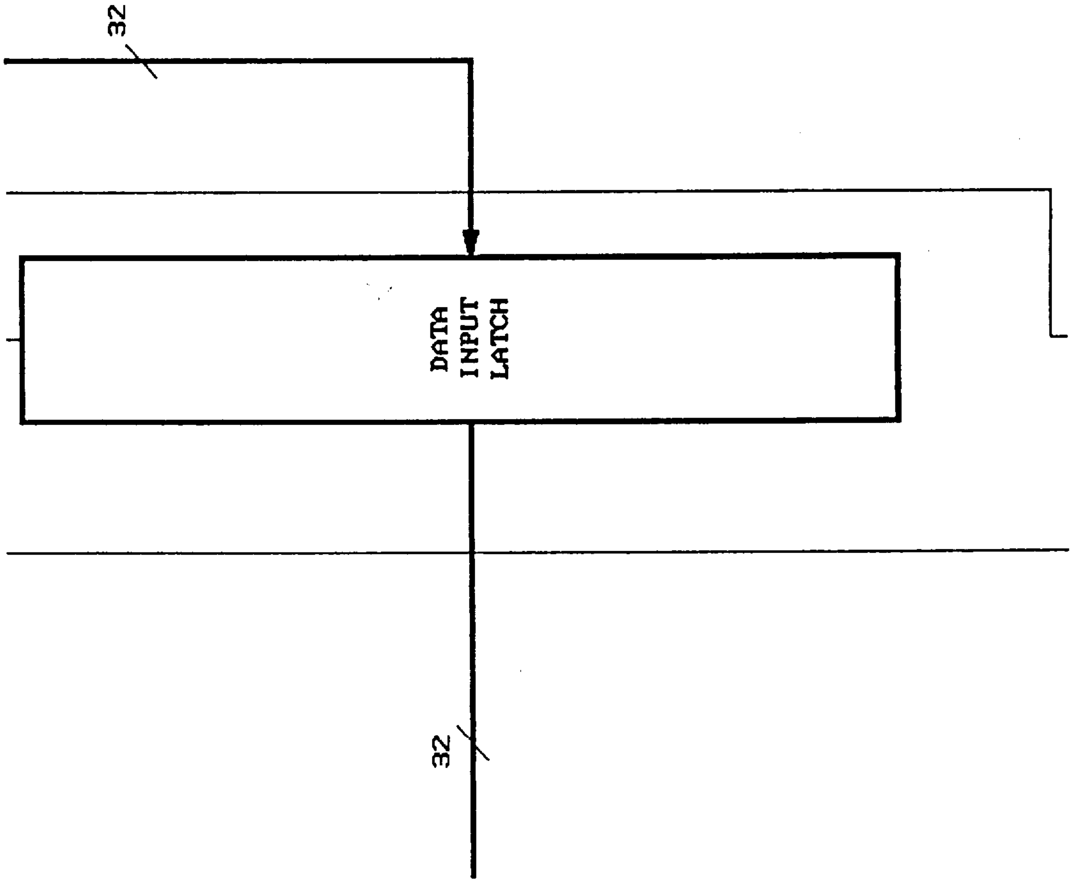


Figure 6

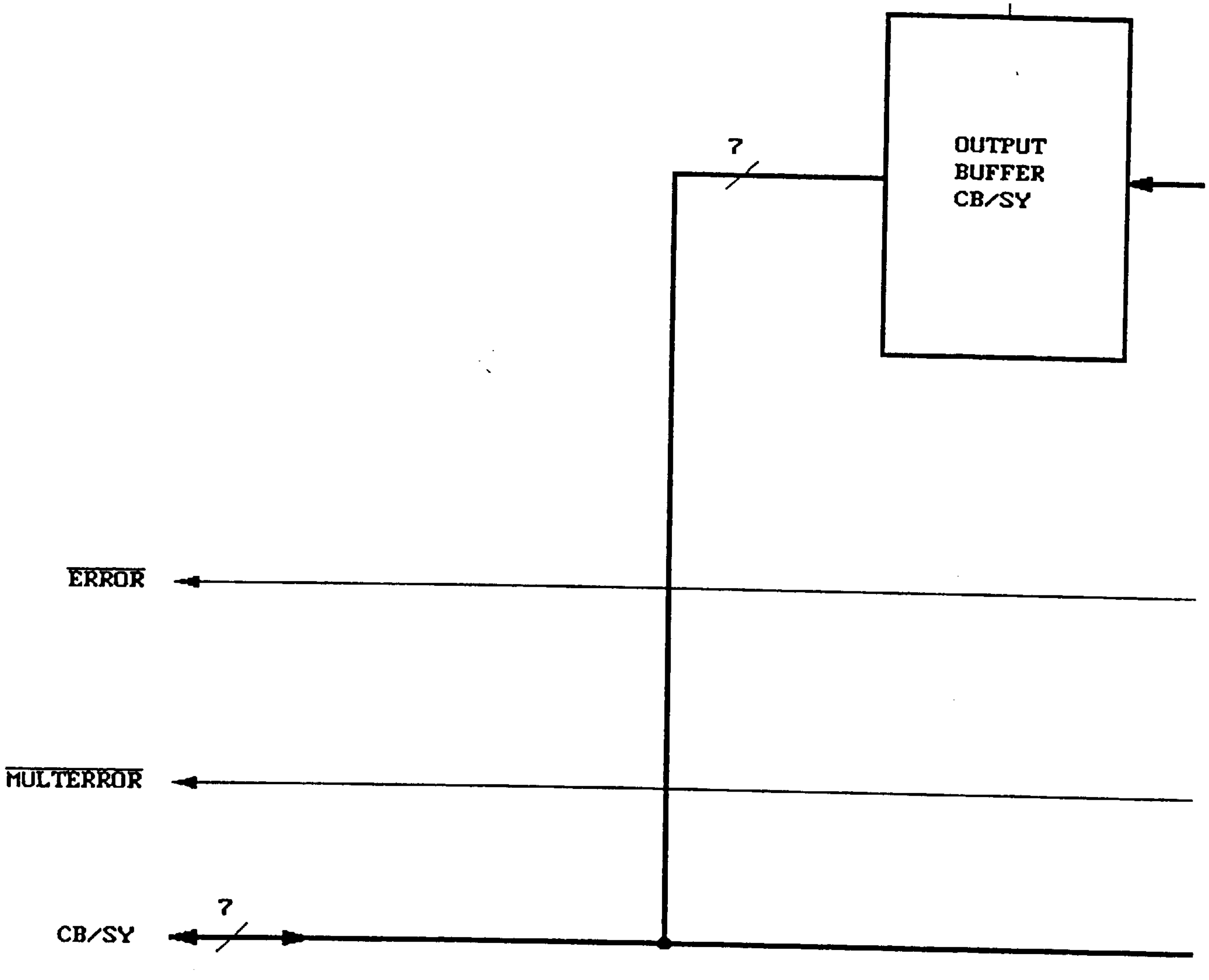

OUTPUT
BUFFER
CB/SY
7
$\overline{\text{ERROR}}$
$\overline{\text{MULTERROR}}$
CB/SY
7

Figure 7

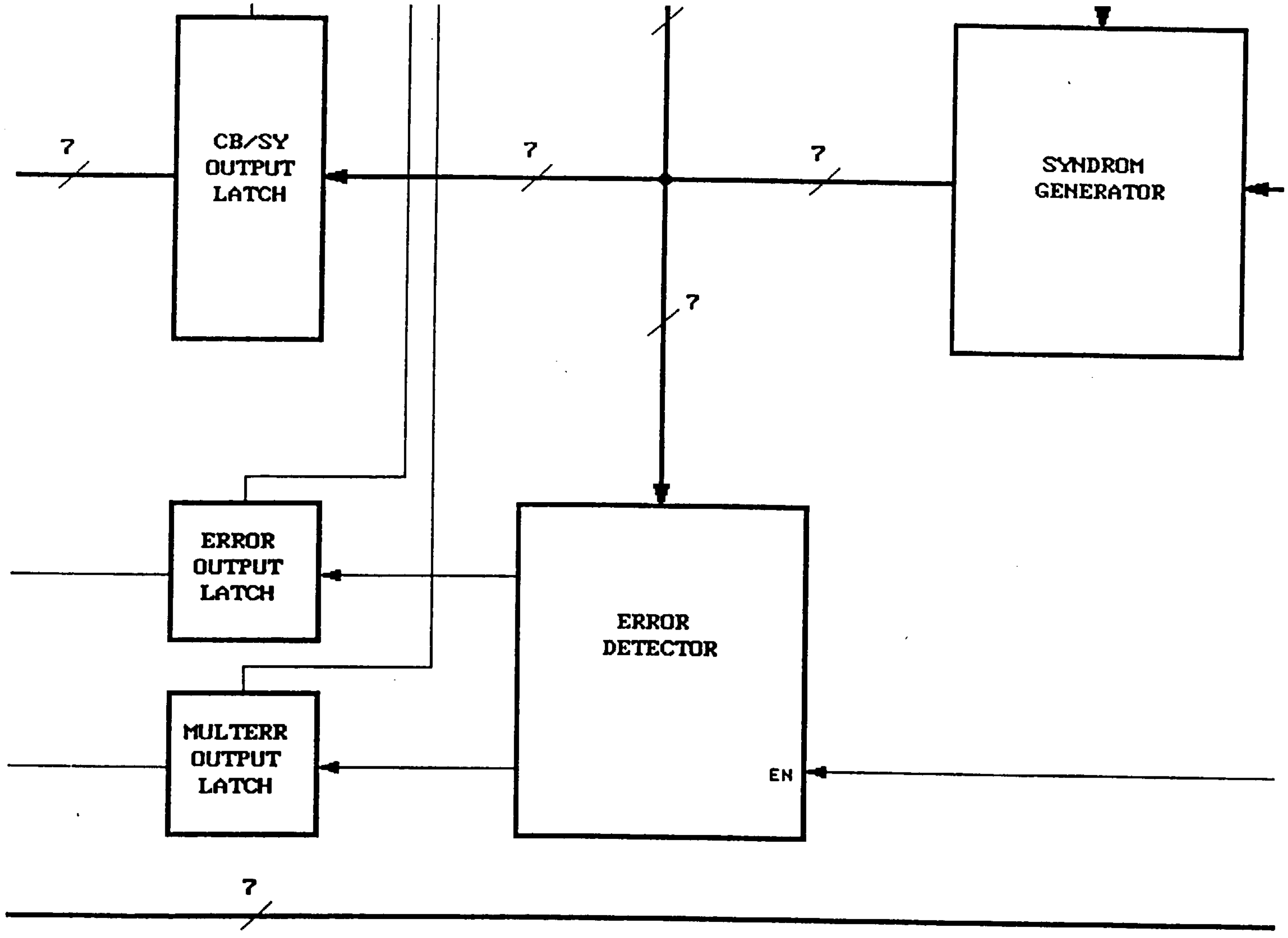

CB/SY
OUTPUT
LATCH
SYNDROM
GENERATOR
ERROR
OUTPUT
LATCH
ERROR
DETECTOR
EN
MULTERR
OUTPUT
LATCH
7
7
7
7
7

Figure 8

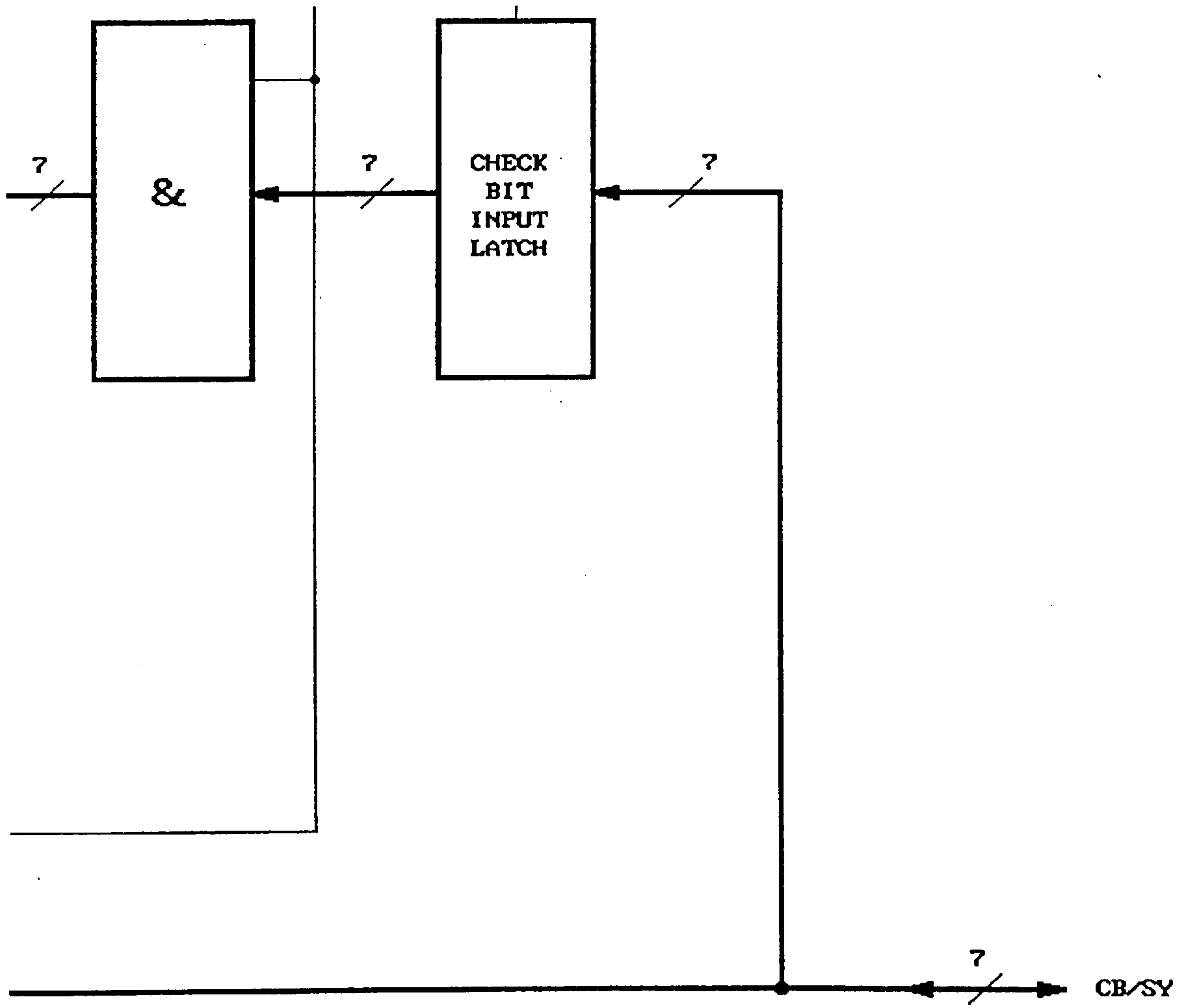
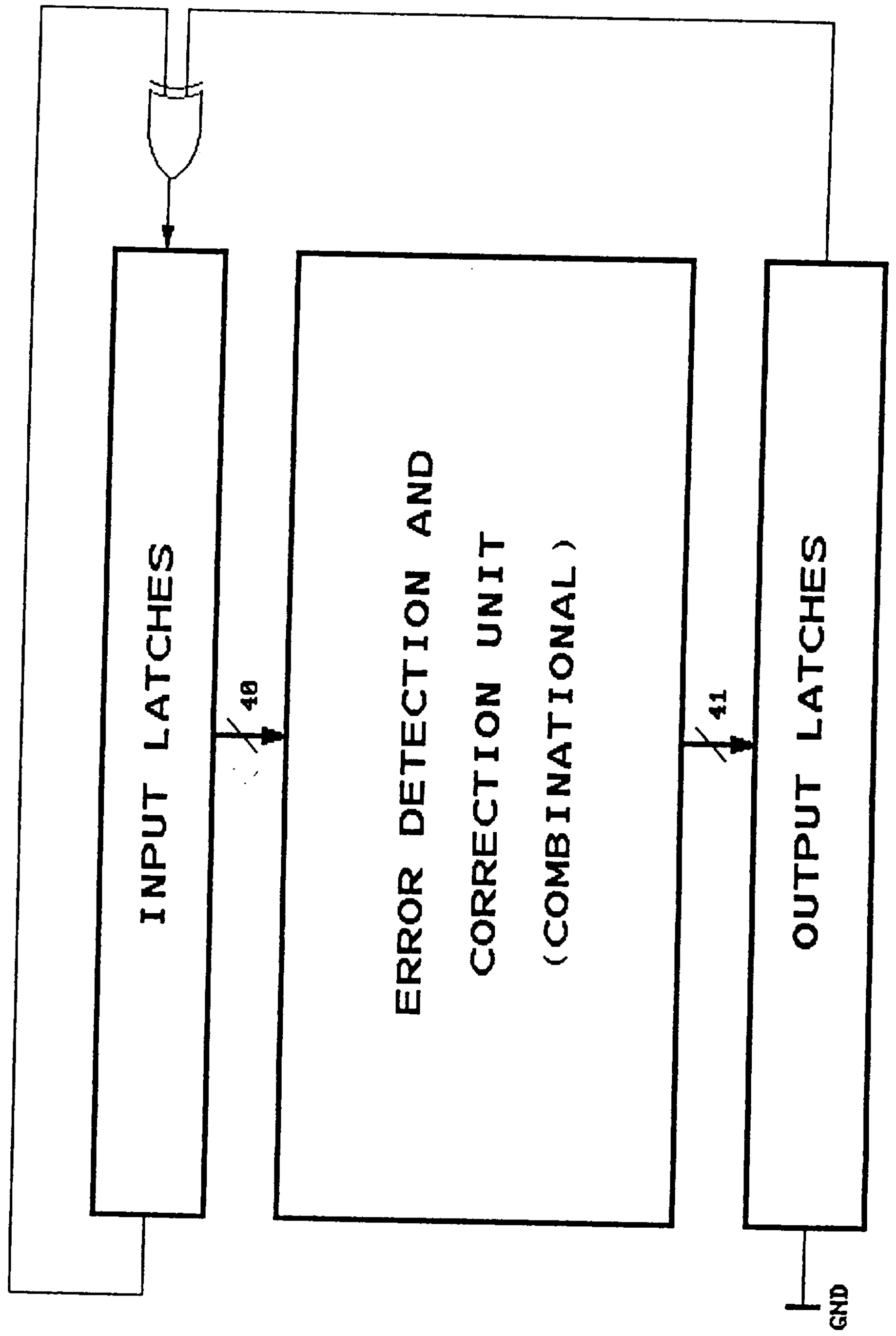


Figure 9

BUILT-IN SELF-TEST



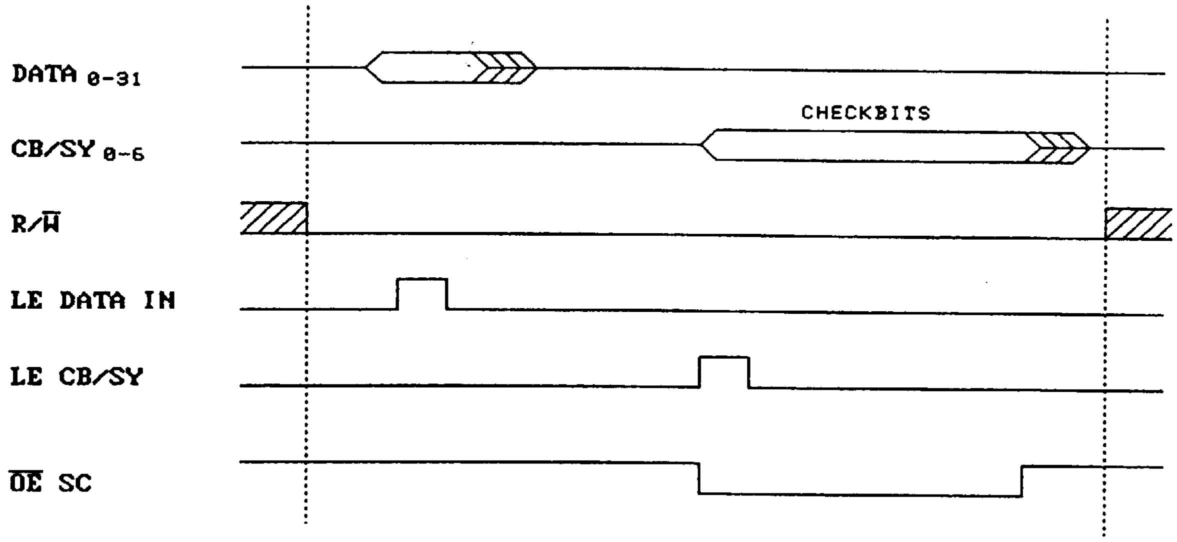
EDAC OPERATING MODES

MODE	R/$\overline{W}$	CORRECT	DATA OUT	CB/SY OUT	$\overline{\text{ERROR}}$ $\overline{\text{MULT\_ERROR}}$
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 **	-

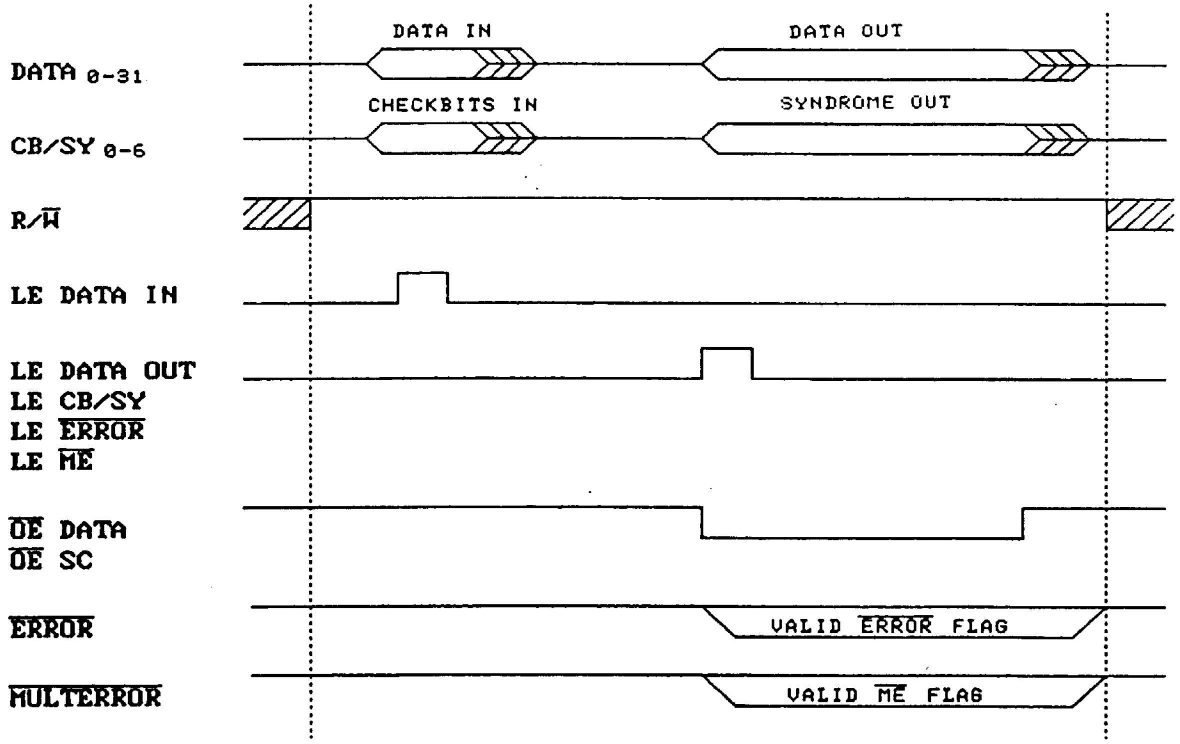
* changing during Self-Test

** depending on R/$\overline{W}$ signal

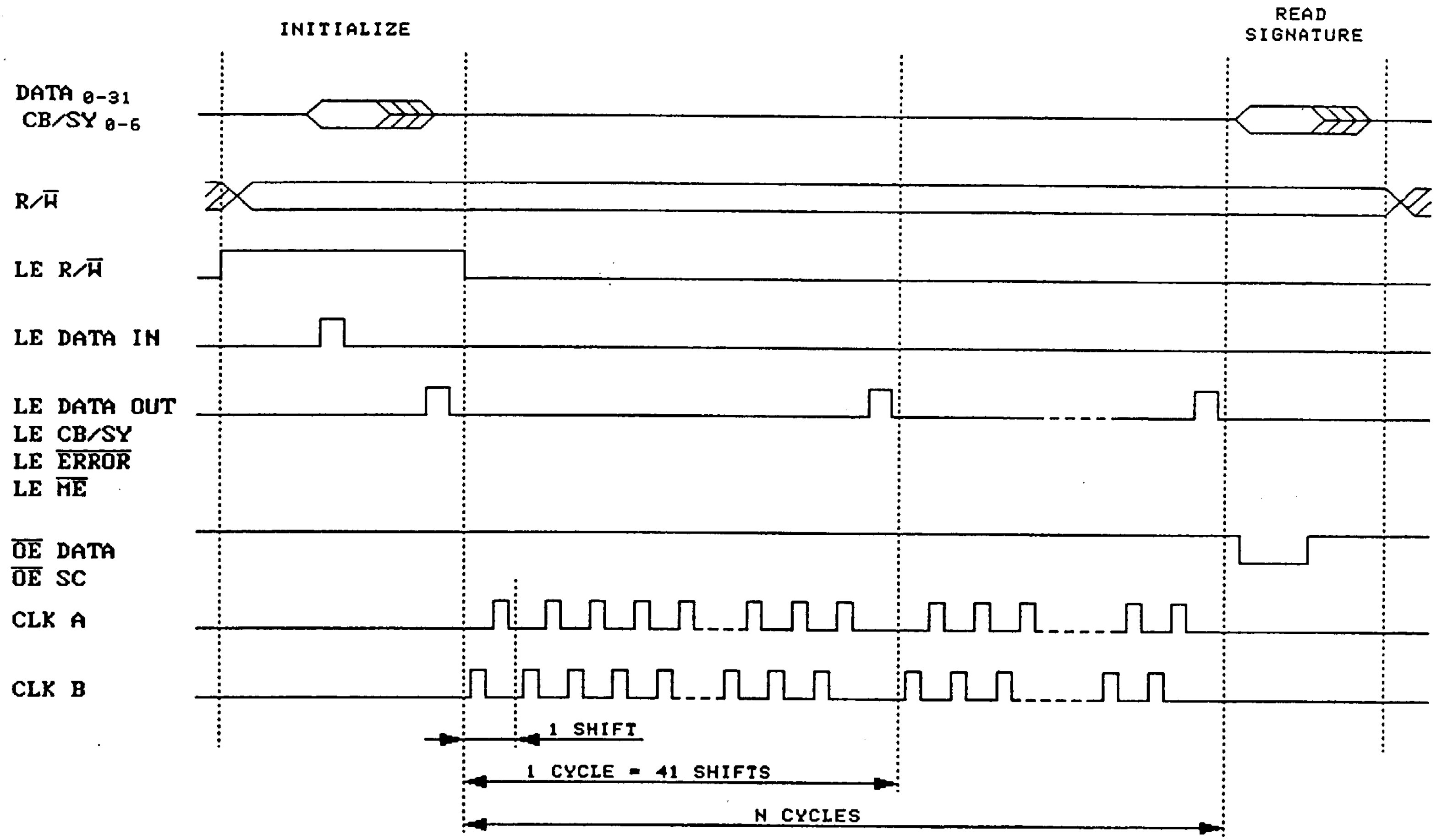
WRITE MODE SWITCHING WAVEFORMS



READ MODE SWITCHING WAVEFORMS



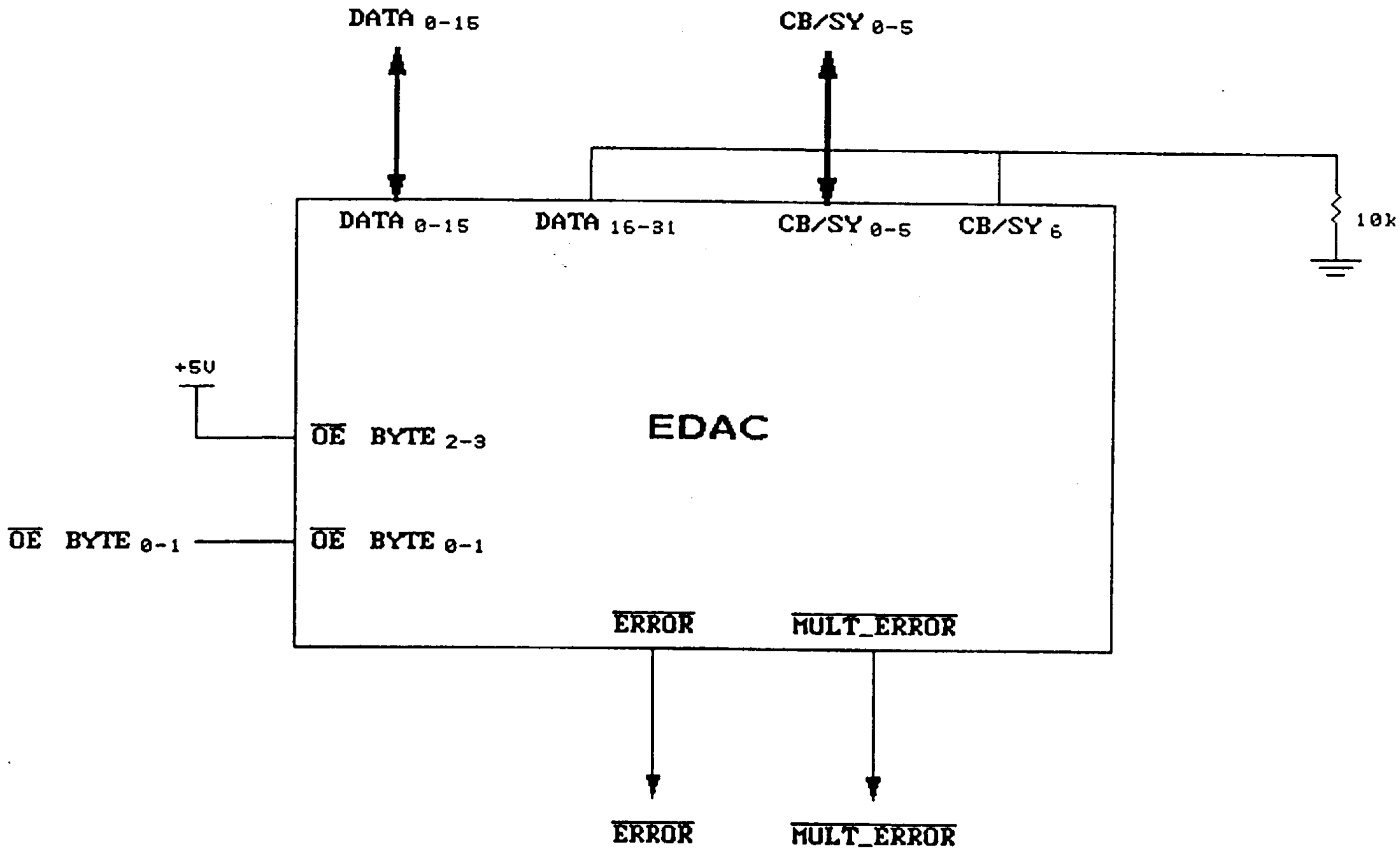
SELF TEST MODE SWITCHING WAVEFORMS



BUILT-IN SELF-TEST SIGNALS AND TEST VECTORS

					SEED					SIGNATURE						
CYCLES	R/$\overline{W}$	CORRECT	CLK_A	CLK_B	BYTE 3	BYTE 2	BYTE 1	BYTE 0	CB/SY	BYTE 3	BYTE 2	BYTE 1	BYTE 0	MULT_ERROR	ERROR	CB/SY
360	1	1	0	0	FF	FF	FF	FF	AA	DB	03	C7	04	1	1	6C
20	1	0	0	0	FF	FF	FF	FF	AA	23	9C	B4	E6	0	1	54
6	1	1	0	0	FF	FF	FF	FF	FF	06	9E	48	52	0	0	07
0 0 0	1 1 1	1 1 1	0 0 0	0 1 0	00 - 00	00 - 00	00 - 00	00 - 00	00 - 00	- - 00	- - 00	- - 00	- - 00	- - 1	- - 1	- - 00
0 0 0	1 1 1	1 1 1	0 0 0	0 1 0	FF - 00	FF - 00	FF - 00	FF - 00	FF - 00	- - 00	- - 00	- - 00	- - 00	- - 1	- - 1	- - 00
0 0 0	1 1 1	1 1 1	0 0 0	0 1 0	FF - FF	FF - FF	FF - FF	FF - FF	FF - FF	- - FF	- - FA	- - 96	- - 59	- - 0	- - 0	- - 3E
0 0 0	0 0 0	1 1 1	0 0 0	0 1 0	00 - FF	00 - FF	00 - FF	00 - FF	00 - FF	- - FF	- - FF	- - FF	- - FF	- - 1	- - 1	- - 41
0 0 0	0 0 0	1 1 1	0 0 0	0 1 0	FF - 00	FF - 00	FF - 00	FF - 00	FF - FF	- - 00	- - 00	- - 00	- - 00	- - 1	- - 1	- - 00
0 0 0	1 0 1	1 1 1	0 0 0	0 0 0	00 00 -	00 00 -	00 00 -	00 00 -	FF FF -	- - 00	- - 00	- - 00	- - 00	- - 1	- - 1	- - 00
0 0 0	0 0 1	1 1 1	0 0 0	0 1 0	00 - 00	00 - 00	00 - 00	00 - 00	00 - 00	- - 00	- - 00	- - 00	- - 00	- - 1	- - 1	- - 00
0	1	1	1	0	00	00	00	00	00	80	00	00	00	1	1	00

16 - BIT CONFIGURATION


DATA 0-15
CB/SY 0-5
DATA 0-15
DATA 16-31
CB/SY 0-5
CB/SY 6
10k
+5V
$\overline{\text{OE}}$ BYTE 2-3
EDAC
$\overline{\text{OE}}$ BYTE 0-1
$\overline{\text{OE}}$ BYTE 0-1
$\overline{\text{ERROR}}$
$\overline{\text{MULT\_ERROR}}$
$\overline{\text{ERROR}}$
$\overline{\text{MULT\_ERROR}}$

APPENDIX C: FAULT SIMULATOR BREADBOARD

C.1 Logic Diagram of Fault Simulator Breadboard

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

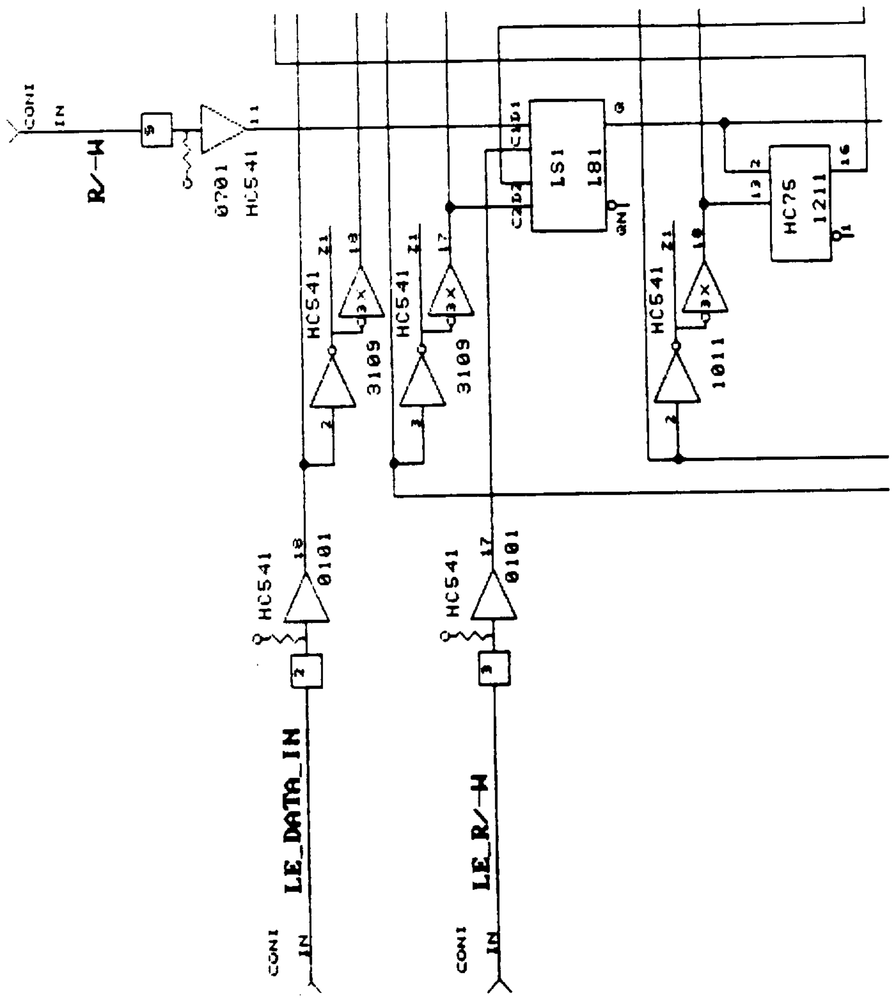

R/-W
CON1
IN
5
0701
HC541
11
LE_DATA_IN
CON1
IN
2
HC541
18
0101
LE_R/-W
CON1
IN
3
HC541
17
0101
HC541
3109
2
21
19
HC541
3109
3
21
17
C1D1
C2D2
LS1
181
G
GN
HC541
1011
2
21
18
HC75
1211
13
2
16
1

Figure 1

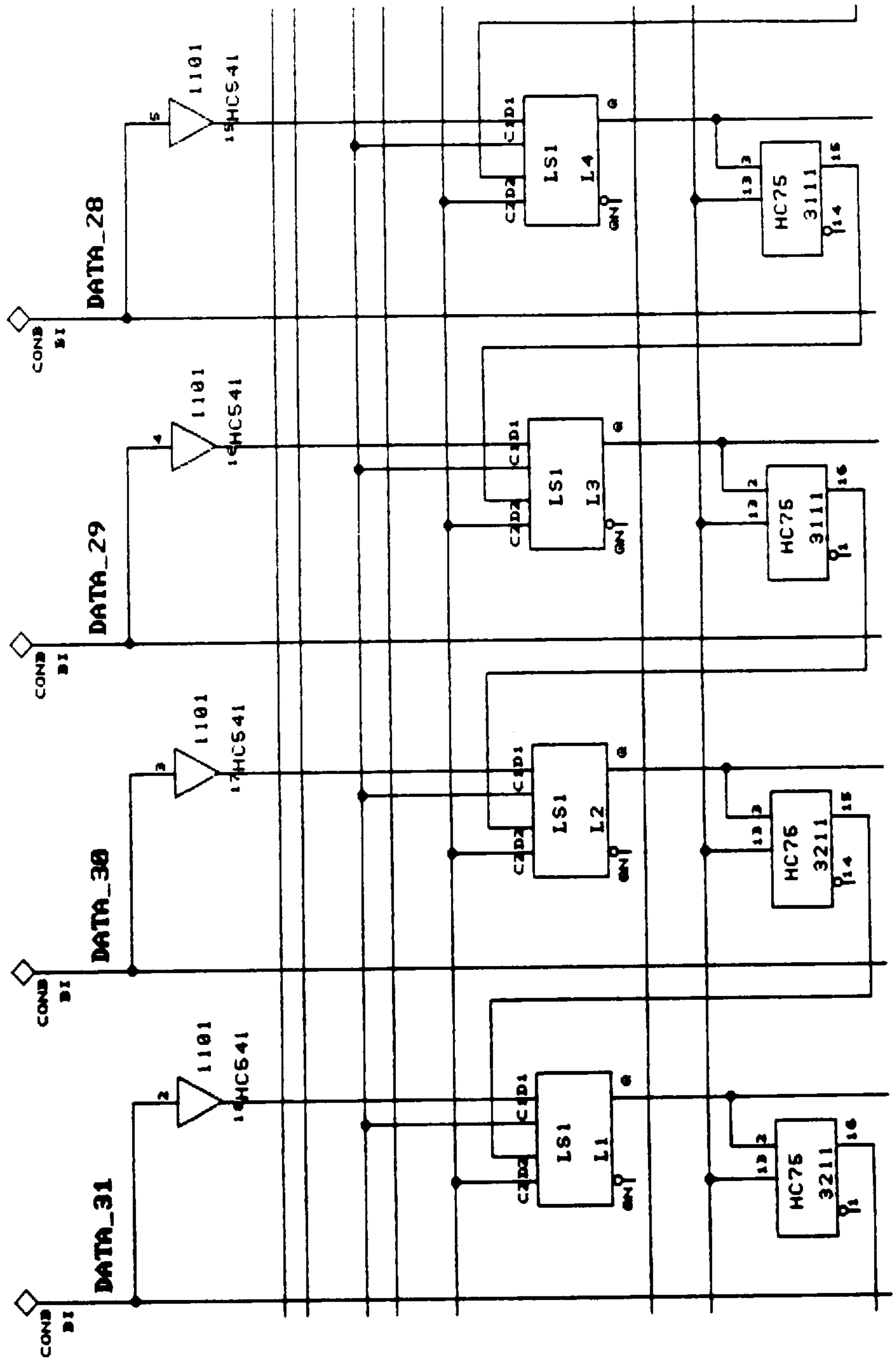

DATA_28
DATA_29
DATA_30
DATA_31
CONB
BI
1101
HC541
LS1
L1
L2
L3
L4
C1D1
C2D2
GN
Q
HC75
3111
3211

Figure 2

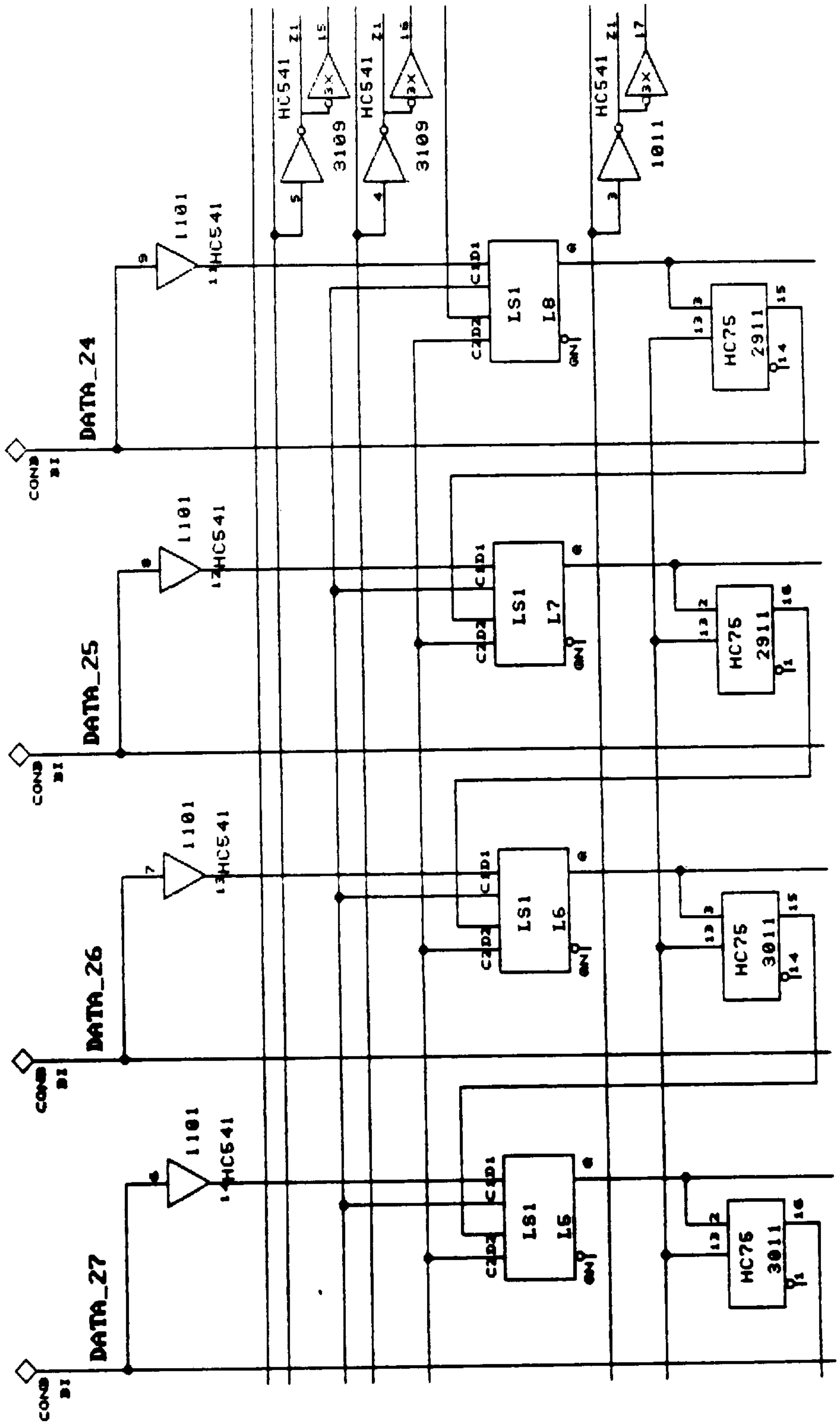

DATA_24
DATA_25
DATA_26
DATA_27
CONB
BI
HC541
1101
HC541
3109
1011
Z1
3X
LS1
L8
L7
L6
L5
C1D1
C2D2
GN
HC75
2911
3011

Figure 3

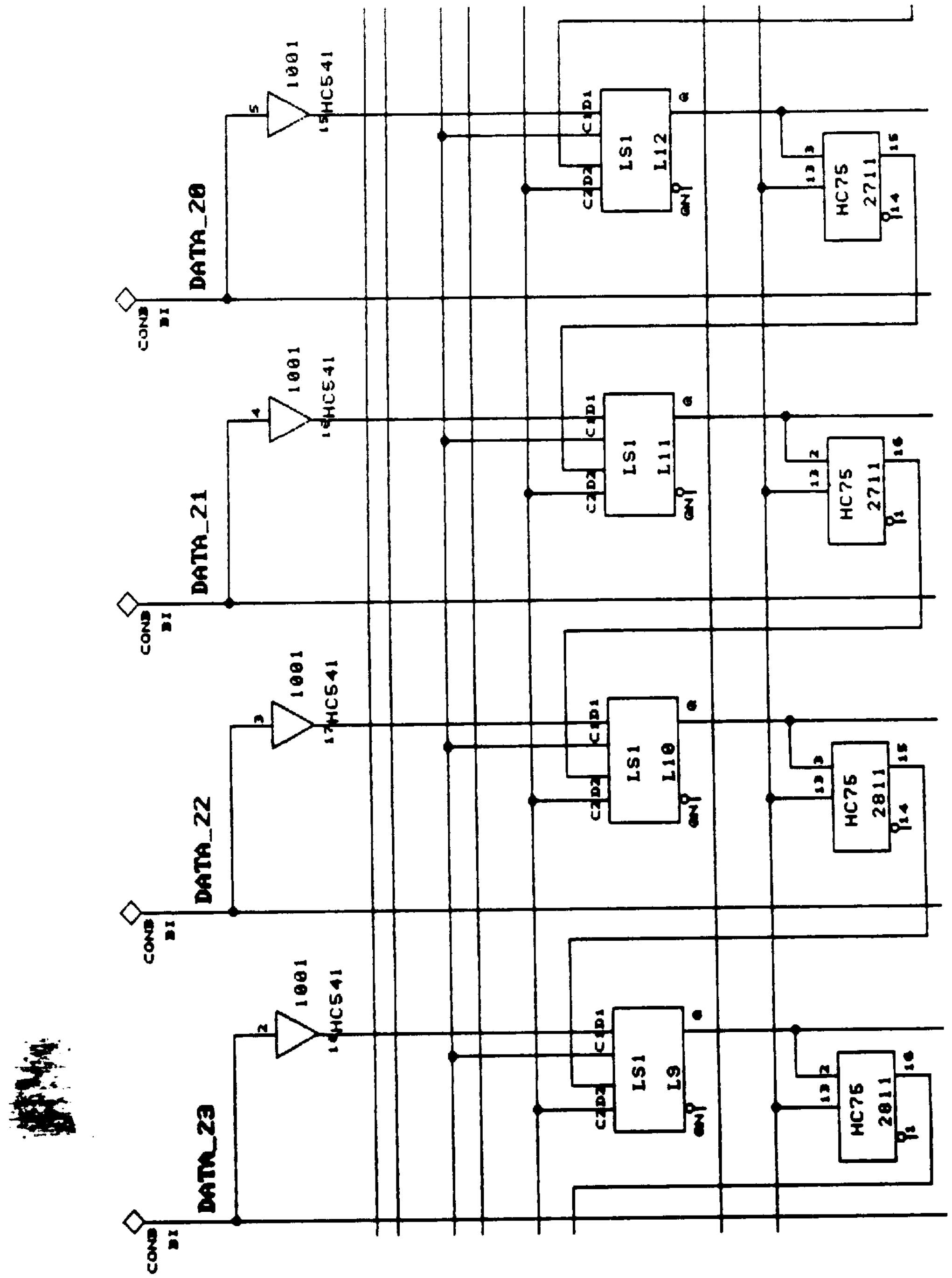

DATA_20
DATA_21
DATA_22
DATA_23
CONB
BI
1001
HC541
LS1
L12
L11
L10
L9
C1D1
C2D2
Q
GN
HC75
2711
2811

Figure 4

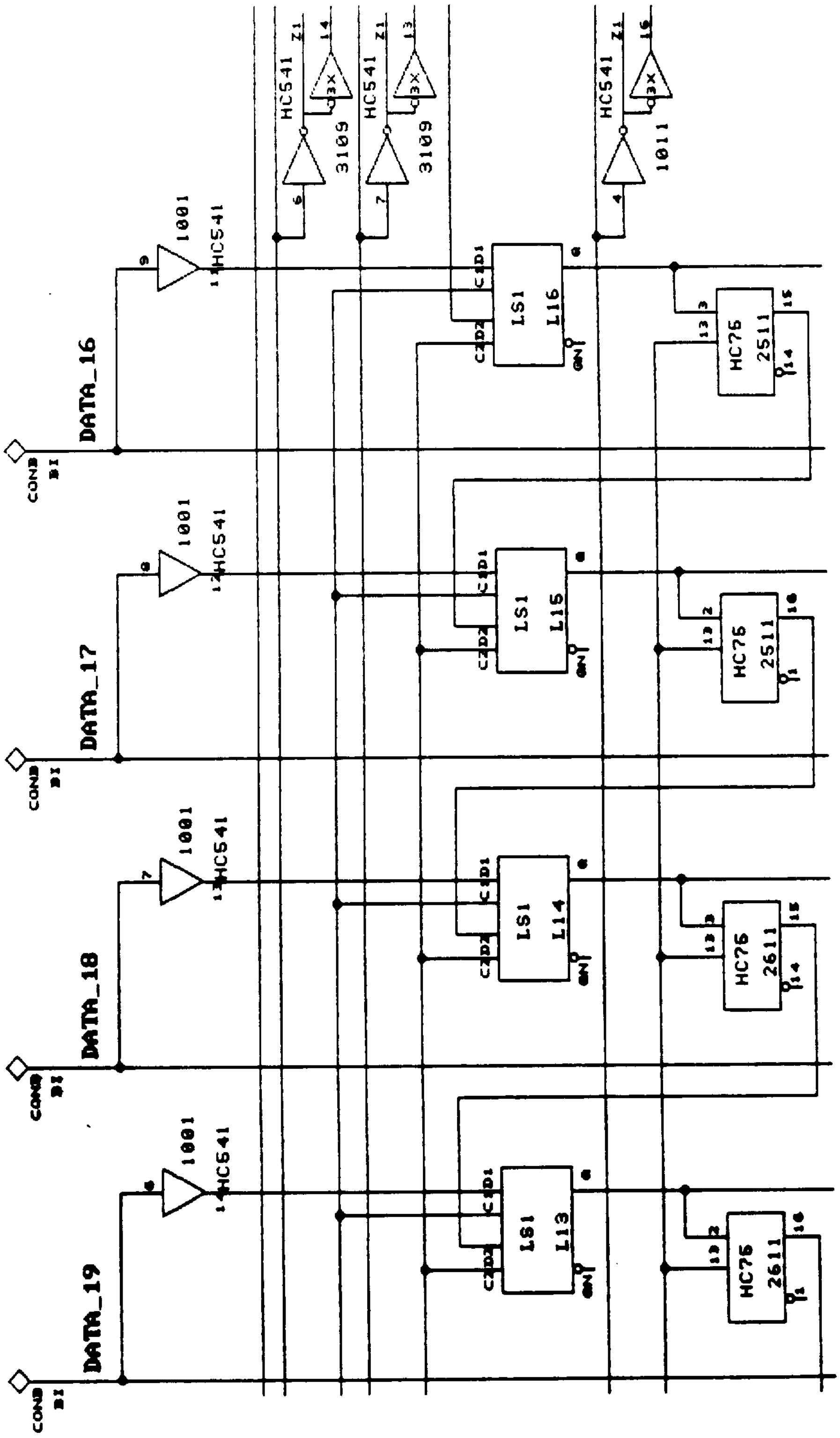

DATA_16
DATA_17
DATA_18
DATA_19
CONB
BI
1001
HC541
HC541
3109
1011
3X
Z1
LS1
L16
L15
L14
L13
C1D1
C2D2
Q
GN
HC75
2511
2611

Figure 5

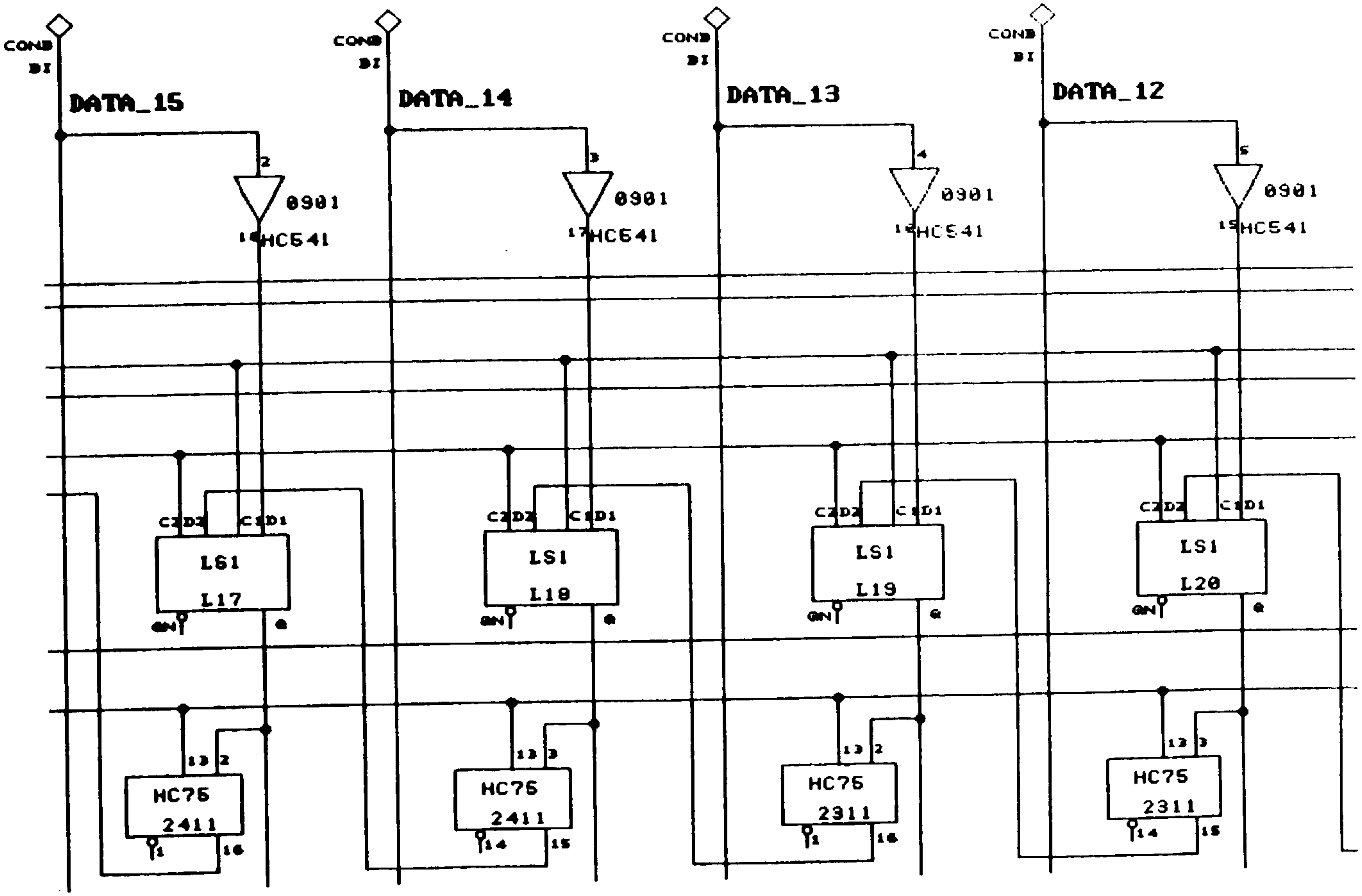

CONB
BI
DATA_15
2
0901
16 HC541
C2D2
C1D1
LS1
L17
GN
Q
13
2
HC75
2411
1
16
CONB
BI
DATA_14
3
0901
17 HC541
C2D2
C1D1
LS1
L18
GN
Q
13
3
HC75
2411
14
15
CONB
BI
DATA_13
4
0901
12 HC541
C2D2
C1D1
LS1
L19
GN
Q
13
2
HC75
2311
1
16
CONB
BI
DATA_12
5
0901
15 HC541
C2D2
C1D1
LS1
L20
GN
Q
13
3
HC75
2311
14
15

Figure 6

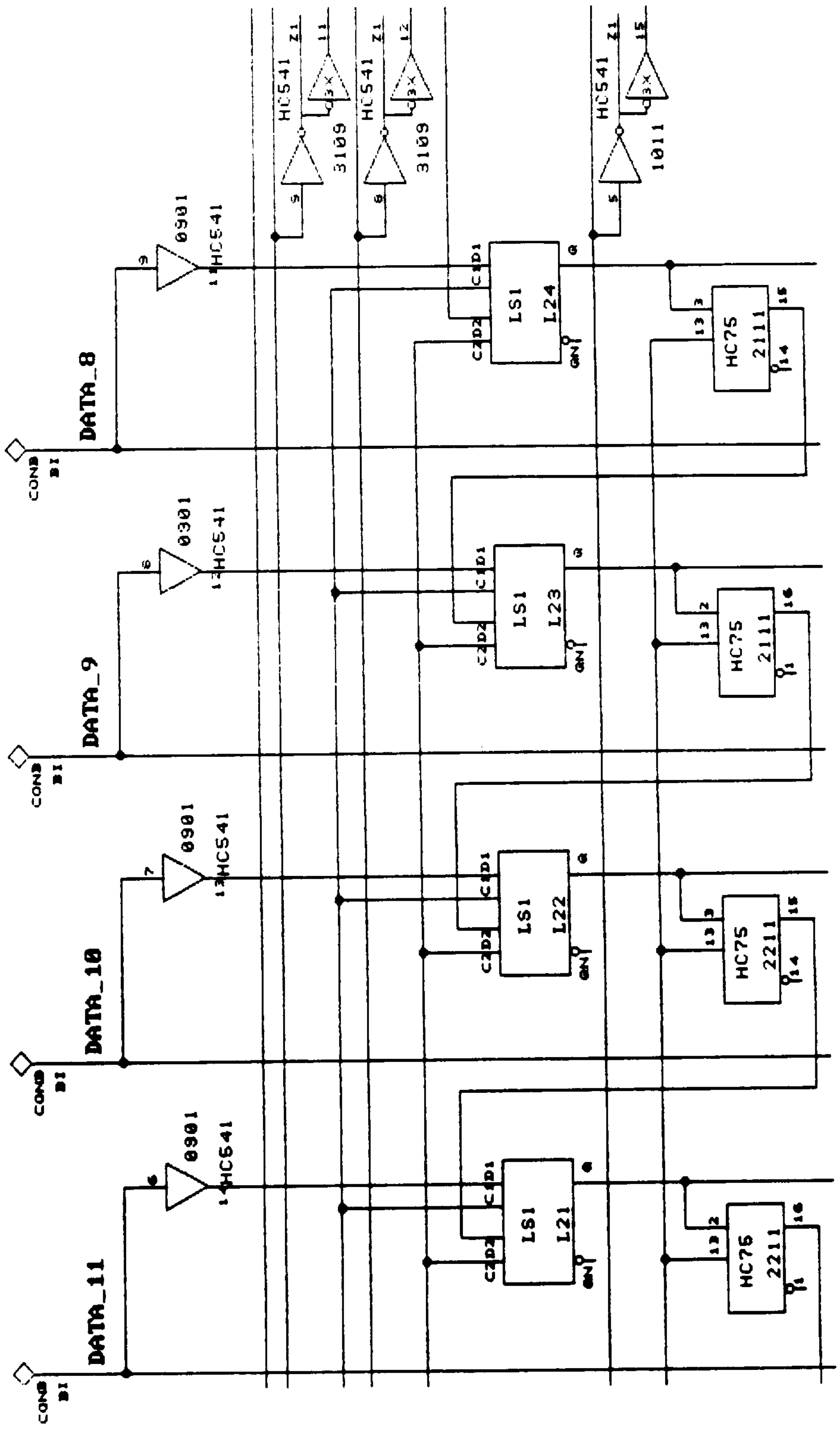

DATA_8
DATA_9
DATA_10
DATA_11
CONB
BI
HC541
0901
0301
LS1
L24
L23
L22
L21
HC75
2111
2211
3109
1011
Z1
11
12
15
3X
C1D1
C2D2
GN
Q

Figure 7

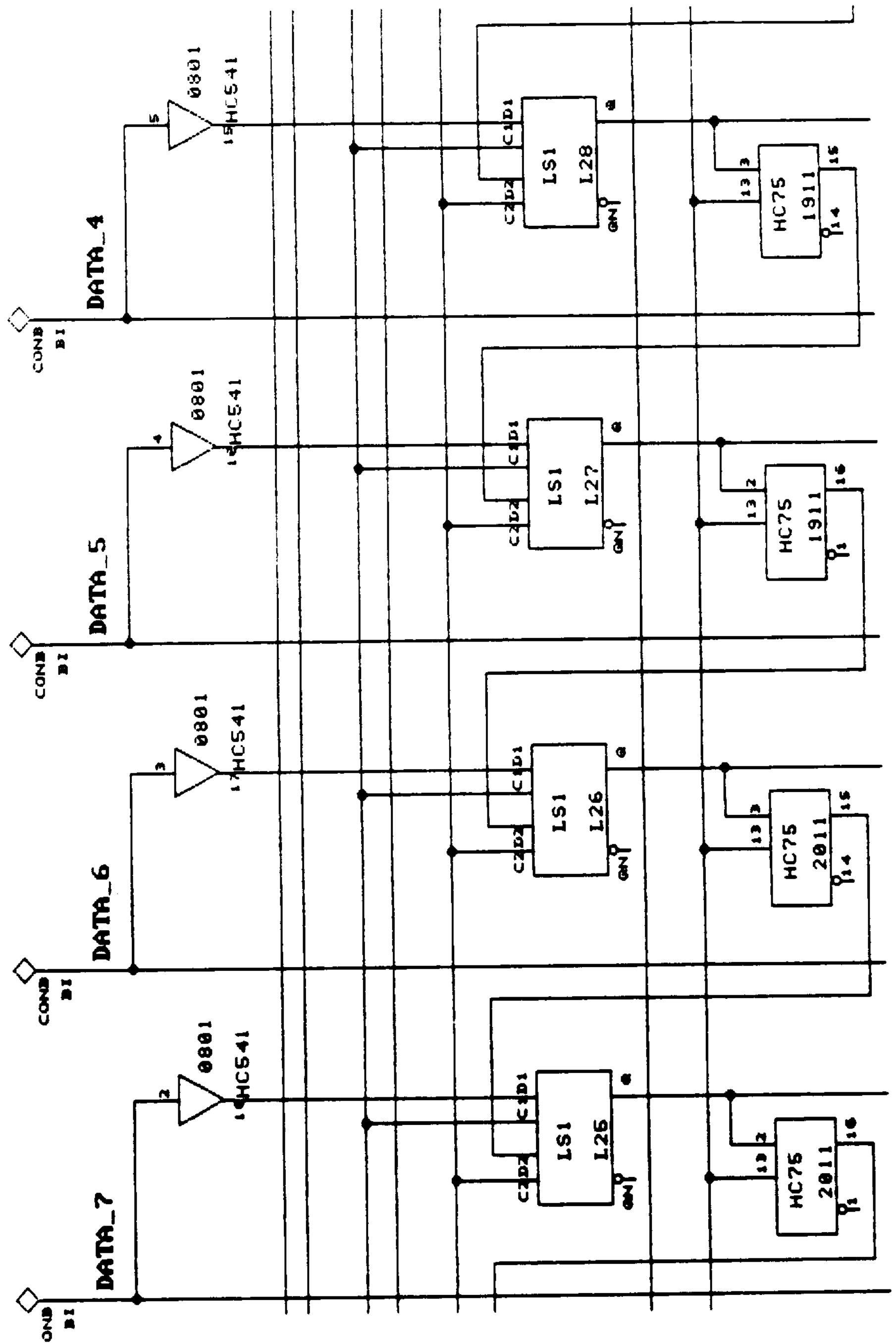

DATA_4
DATA_5
DATA_6
DATA_7
CONB
BI
0801
HC541
LS1
L28
L27
L26
L26
HC75
1911
1911
2011
2011
C1D1
C2D2
Q
GN

Figure 8

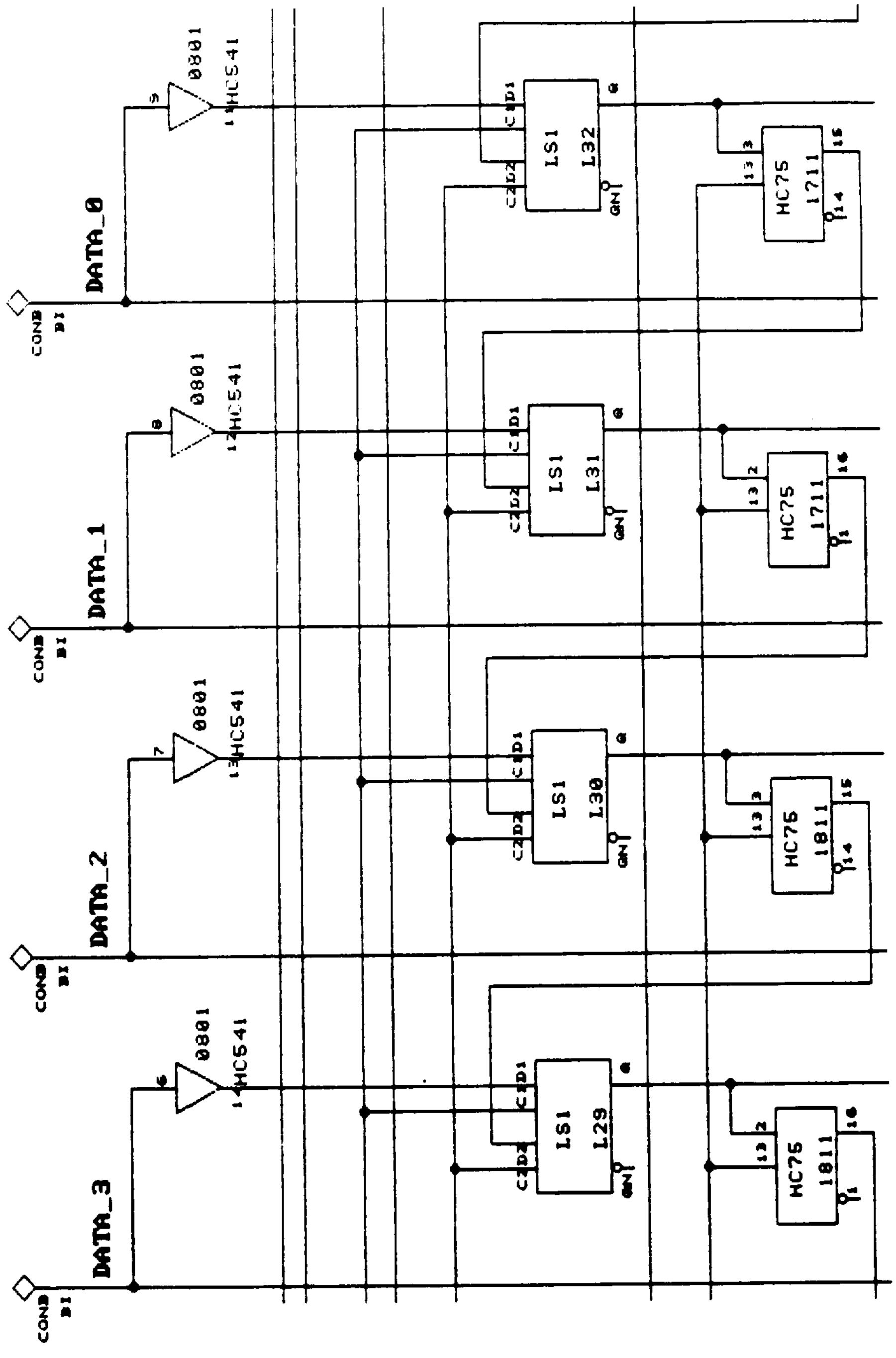

DATA_0
DATA_1
DATA_2
DATA_3
CONB
BI
0801
HC541
LS1
L32
L31
L30
L29
HC75
1711
1811
C1D1
C2D2
GN
Q

Figure 9

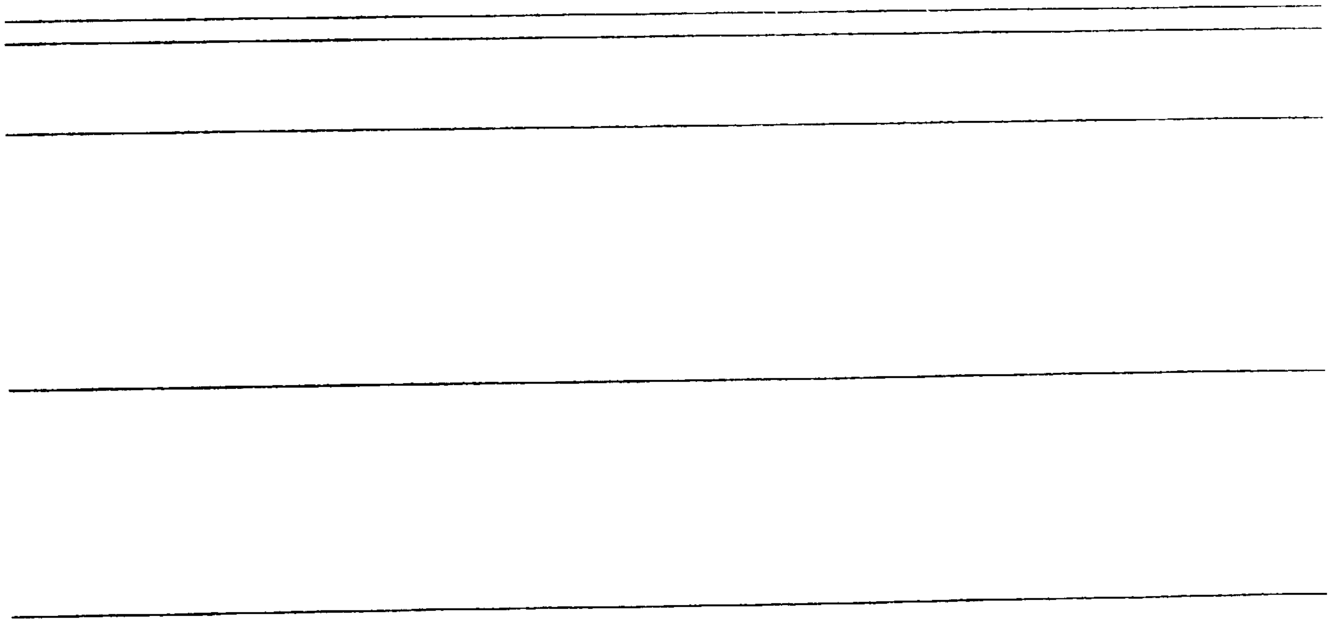


Figure 10

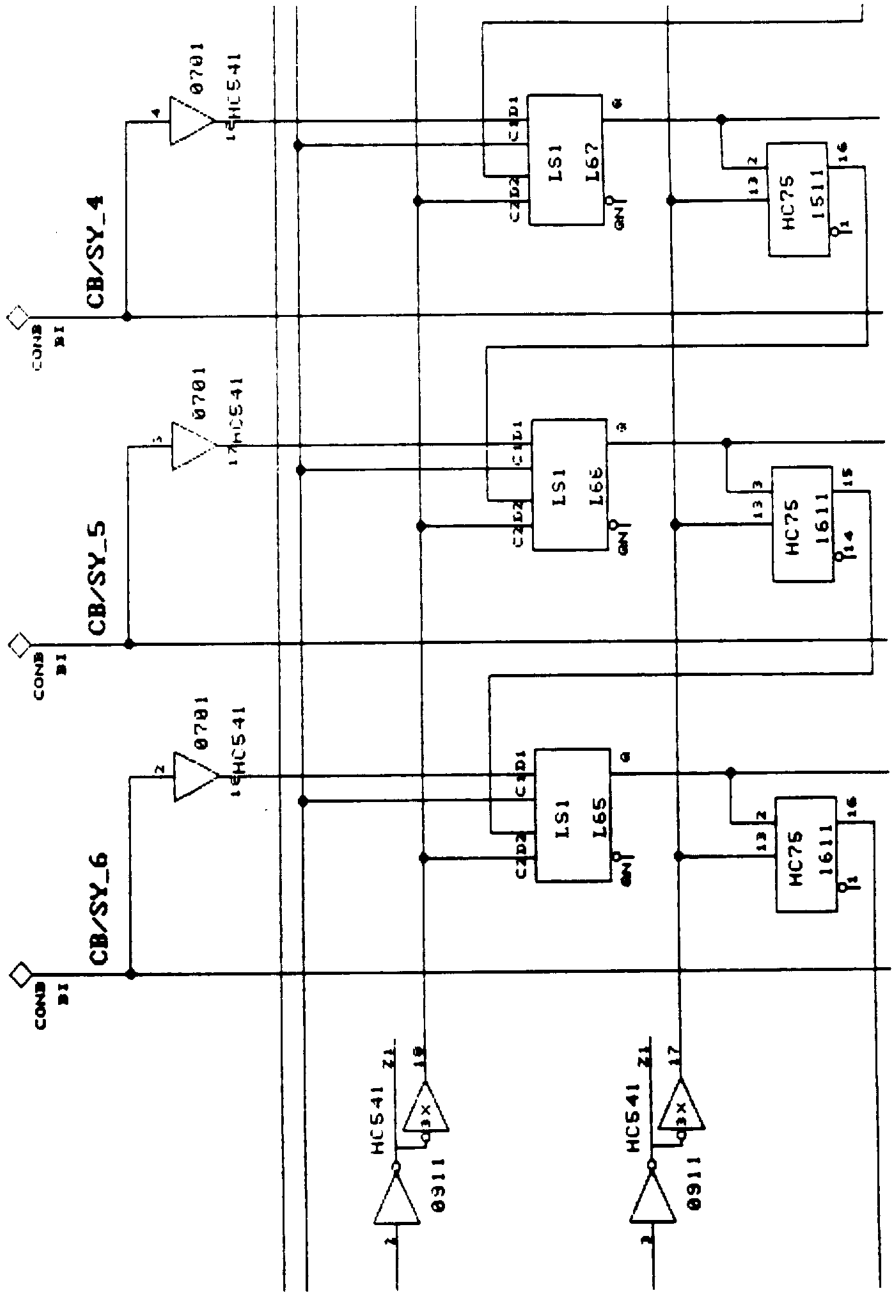

CB/SY_4
CB/SY_5
CB/SY_6
CONB
BI
0701
HC541
LS1
L67
L66
L65
C1D1
C2D2
Q
GN
HC75
1511
1611
HC541
0911
3X
Z1
18
17

Figure 11

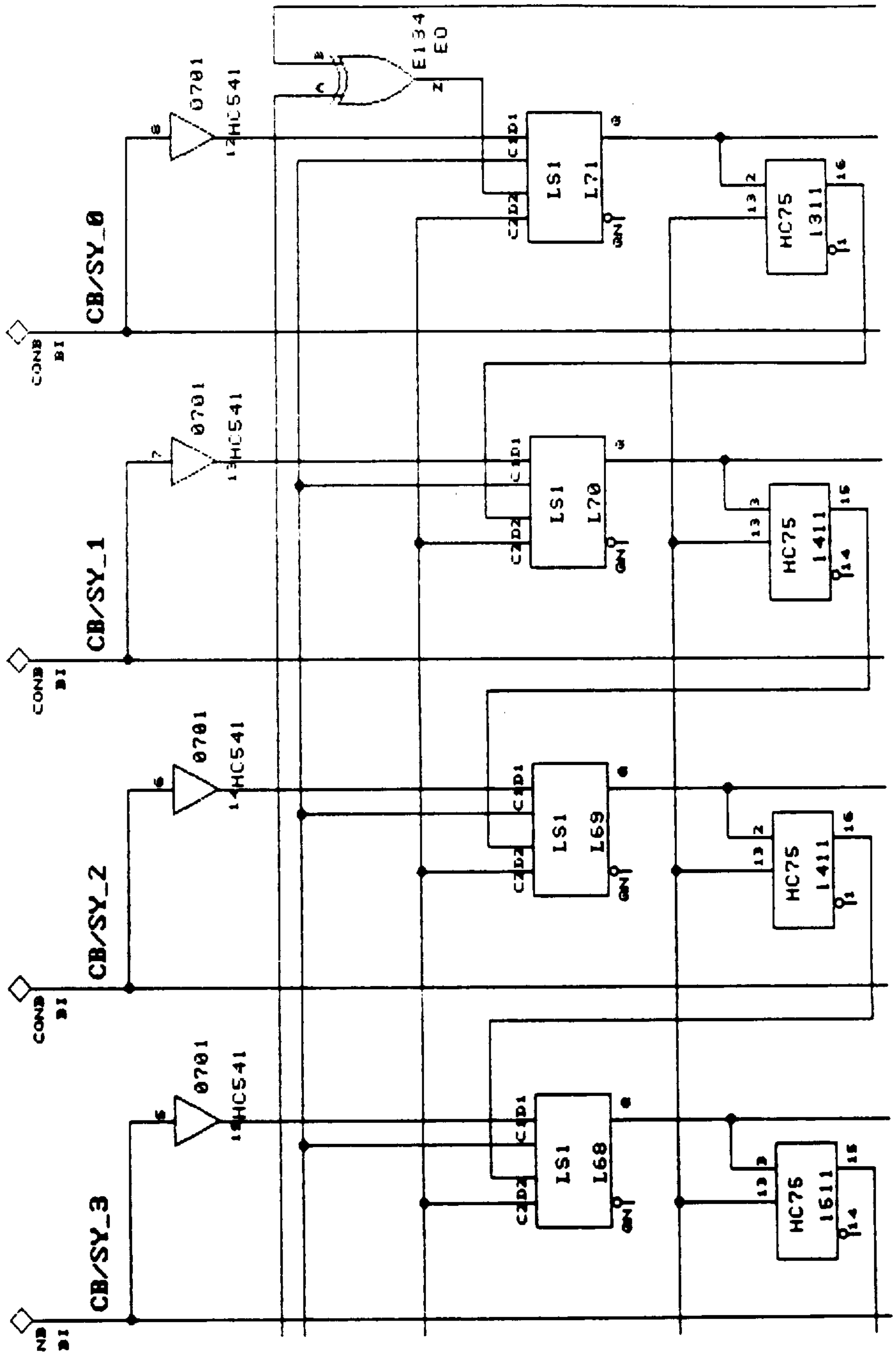

CB/SY_0
CB/SY_1
CB/SY_2
CB/SY_3
CONB
BI
0701
HC541
E134
EO
LS1
L71
L70
L69
L68
HC75
1311
1411
1511
C1D1
C2D2
GN

Figure 12

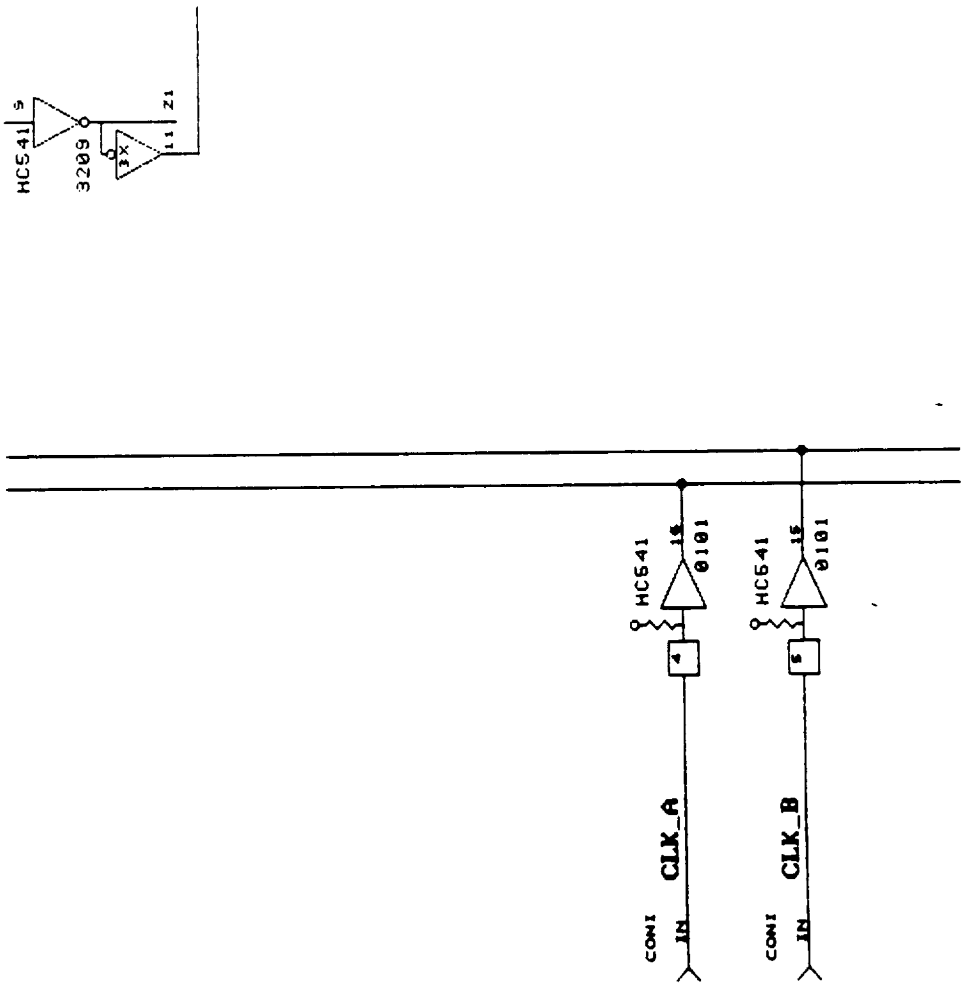

HC541
9
3209
3X
11
Z1
HC541
15
0101
4
CLK_A
CON1
IN
HC541
15
0101
5
CLK_B
CON1
IN

Figure 13

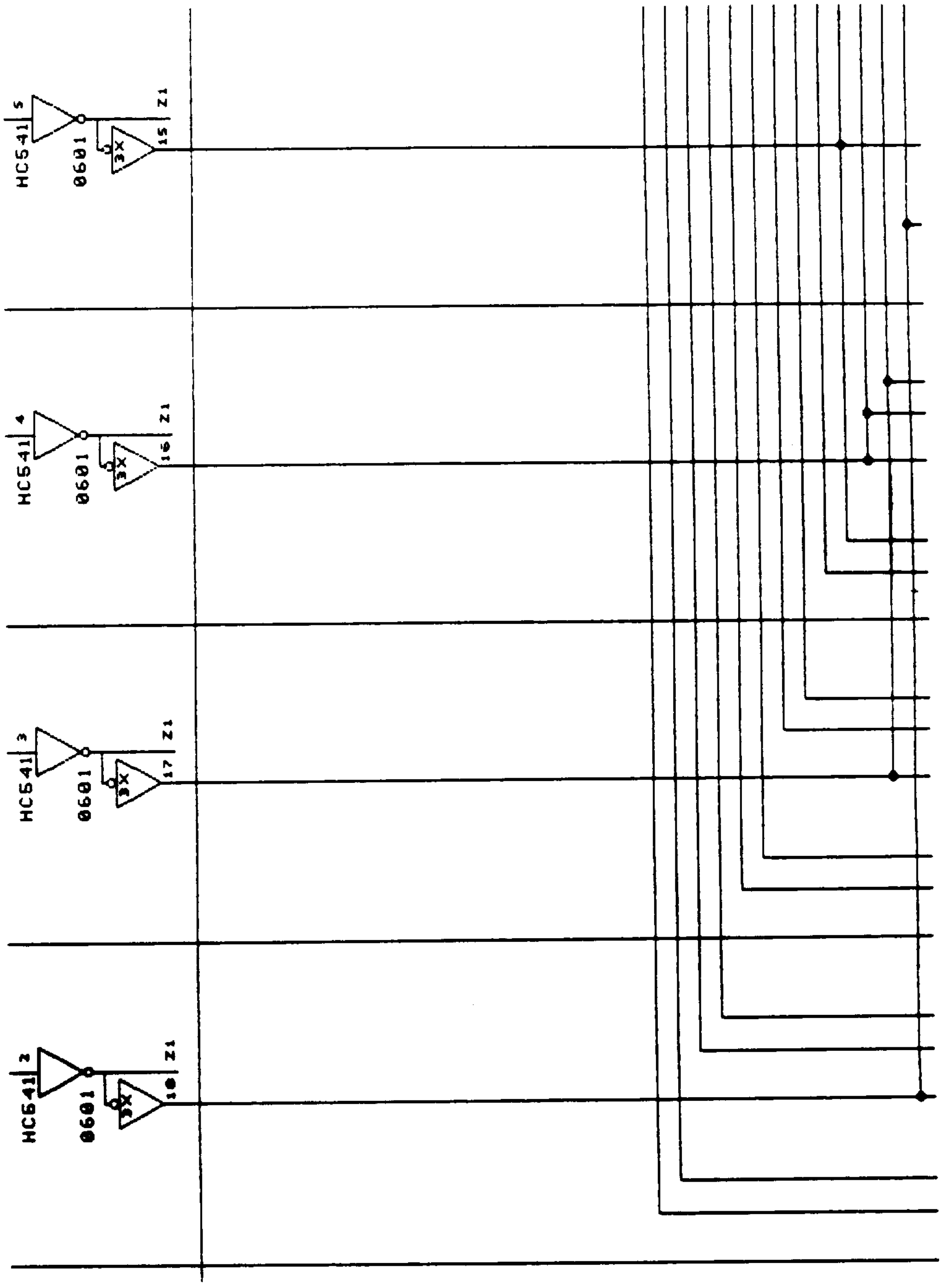

HC541
5
0601
3X
15
Z1
HC541
4
0601
3X
16
Z1
HC541
3
0601
3X
17
Z1
HC541
2
0601
3X
18
Z1

Figure 14

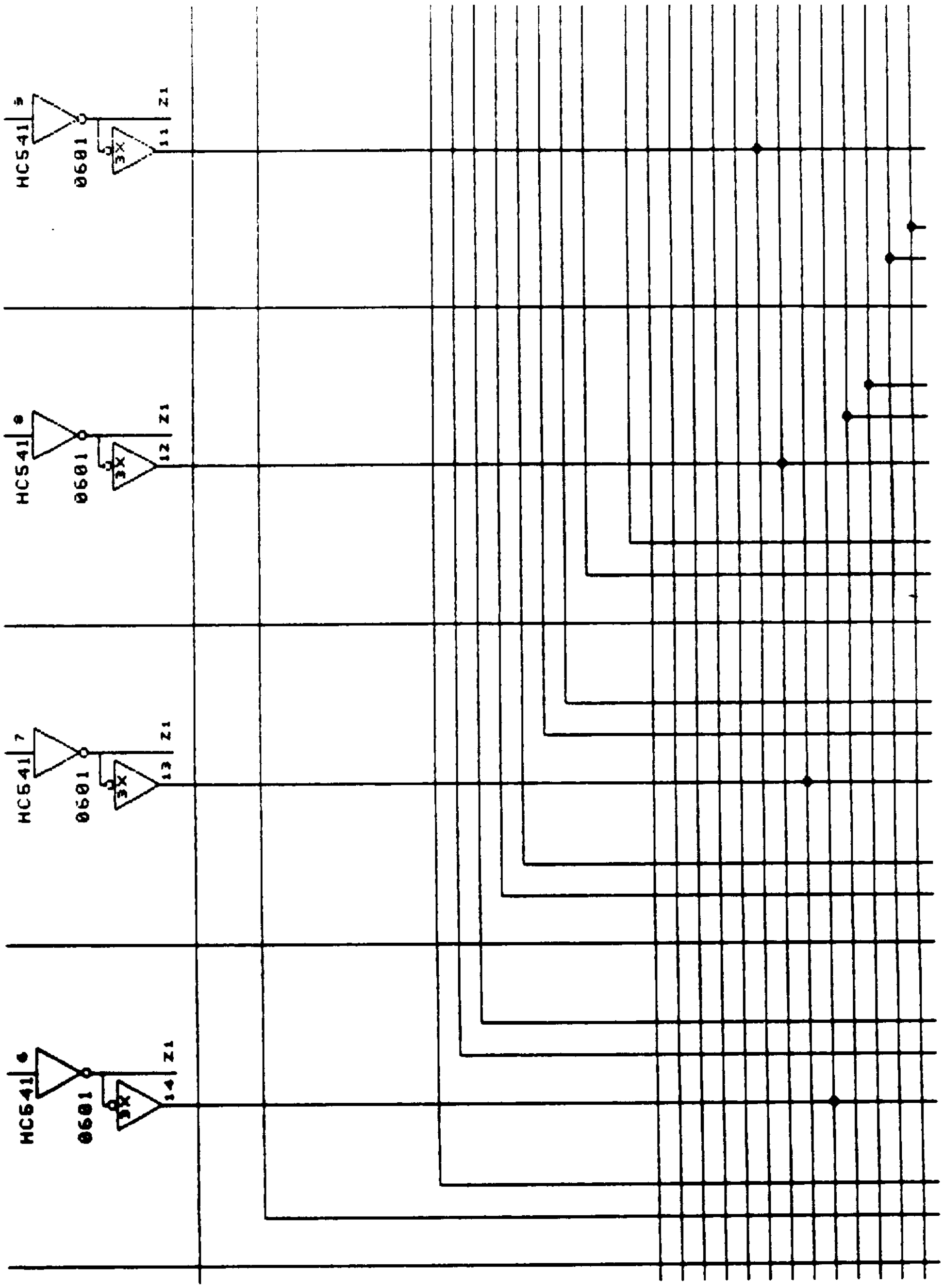

HC541
9
Ø601
3X
11
Z1
HC541
8
Ø601
3X
12
Z1
HC541
7
Ø601
3X
13
Z1
HC541
6
Ø601
3X
14
Z1

Figure 15

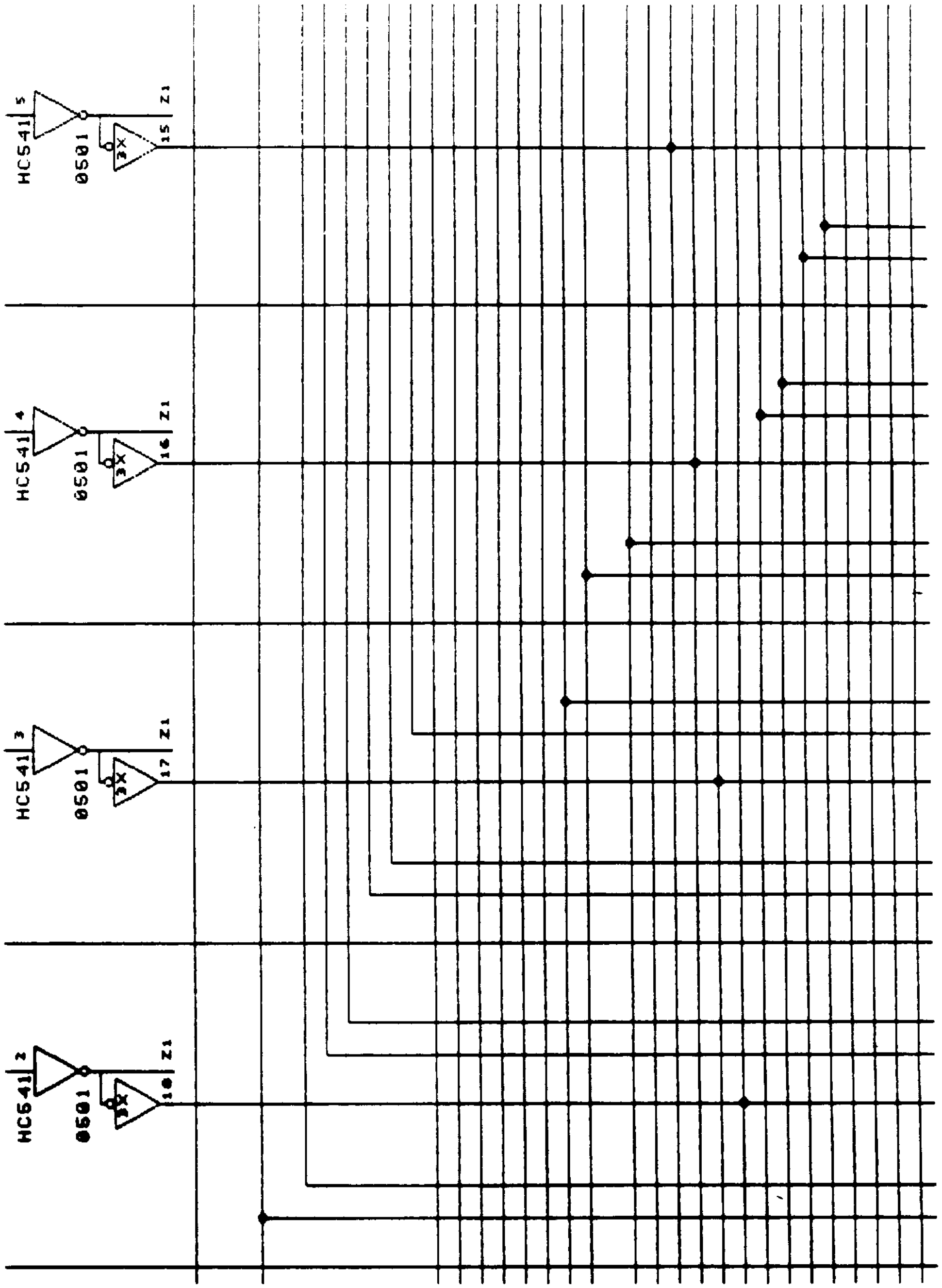

HC541
5
0501
3X
15
Z1
HC541
4
0501
3X
16
Z1
HC541
3
0501
3X
17
Z1
HC541
2
0501
3X
18
Z1

Figure 16

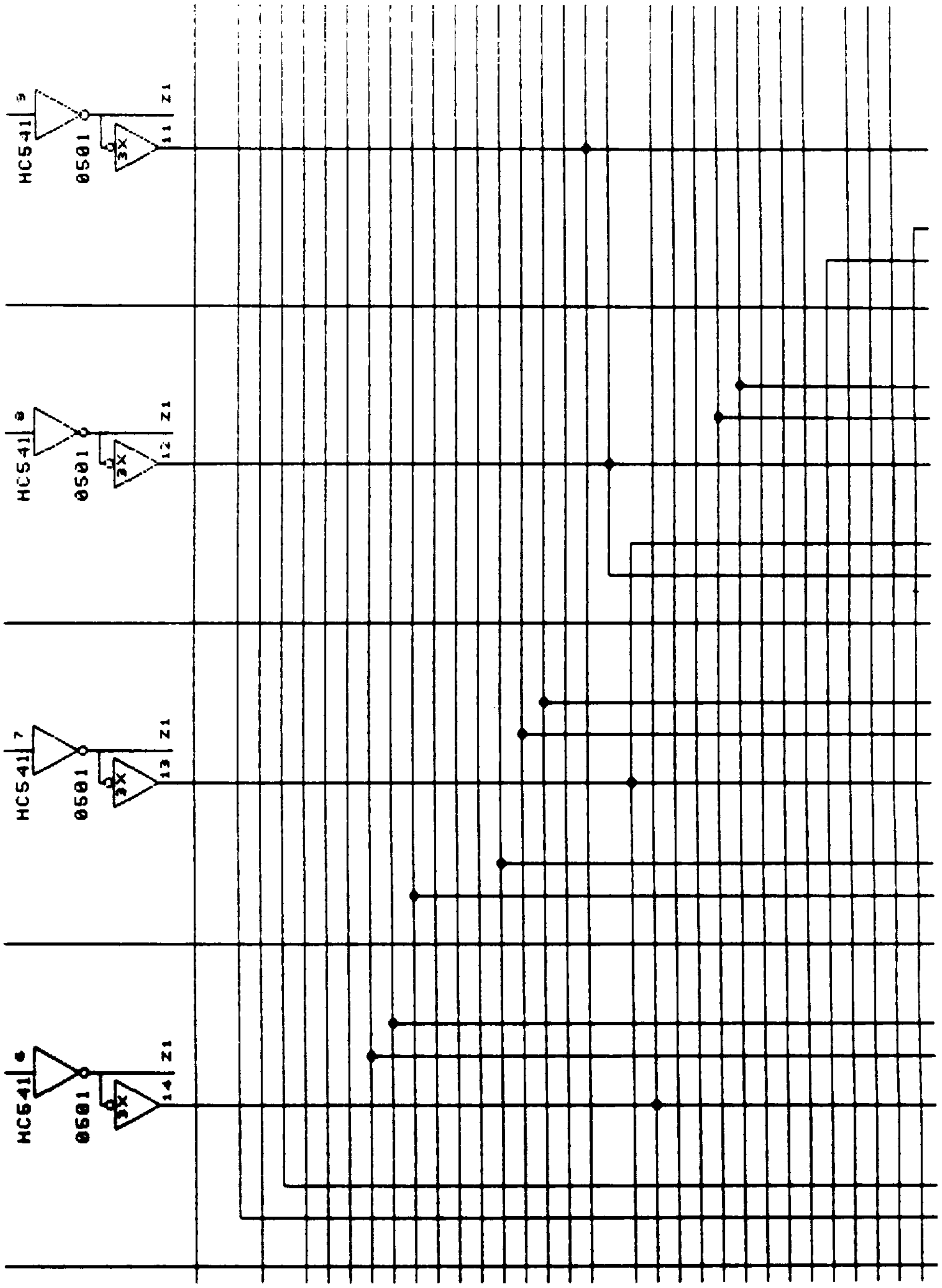

HC541
9
0501
3X
11
Z1
HC541
8
0501
3X
12
Z1
HC541
7
0501
3X
13
Z1
HC541
6
0601
3X
14
Z1

Figure 17

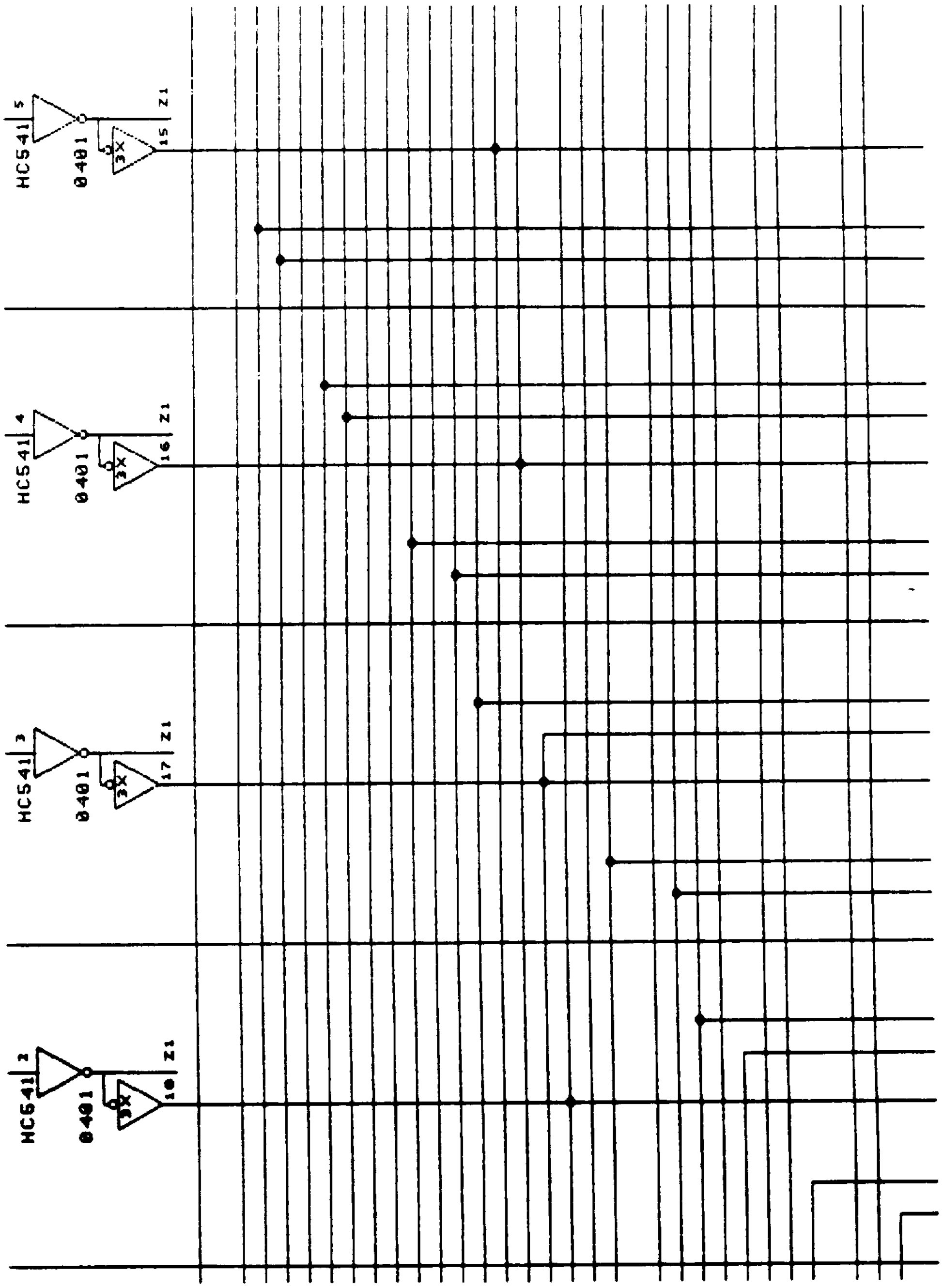

HC541
5
0401
3X
15
Z1
HC541
4
0401
3X
16
Z1
HC541
3
0401
3X
17
Z1
HC541
2
0401
3X
18
Z1

Figure 18

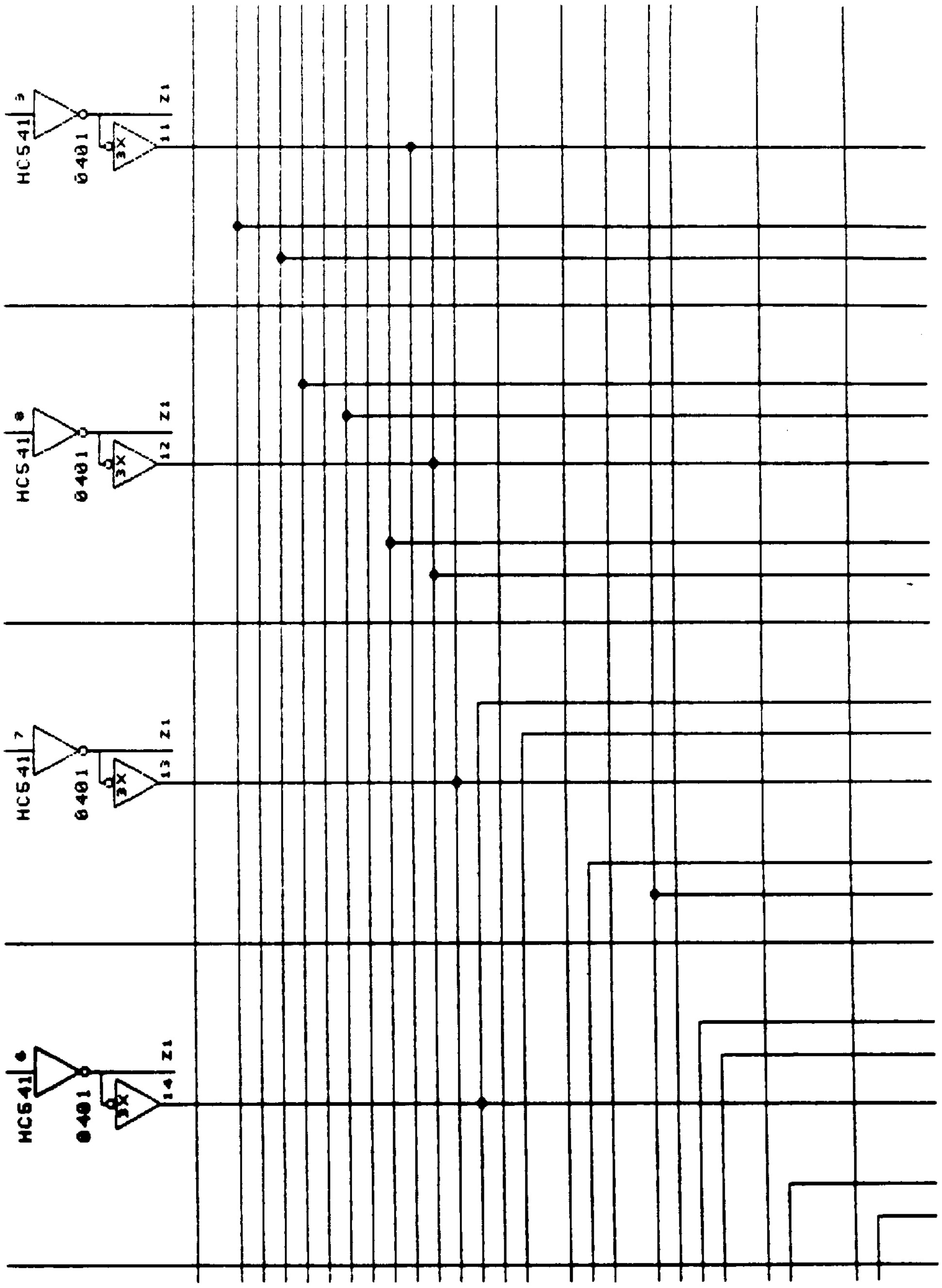

HC541
3
Ø401
3X
11
Z1
HC541
8
Ø401
3X
12
Z1
HC541
7
Ø401
3X
13
Z1
HC541
6
Ø401
3X
14
Z1

Figure 19

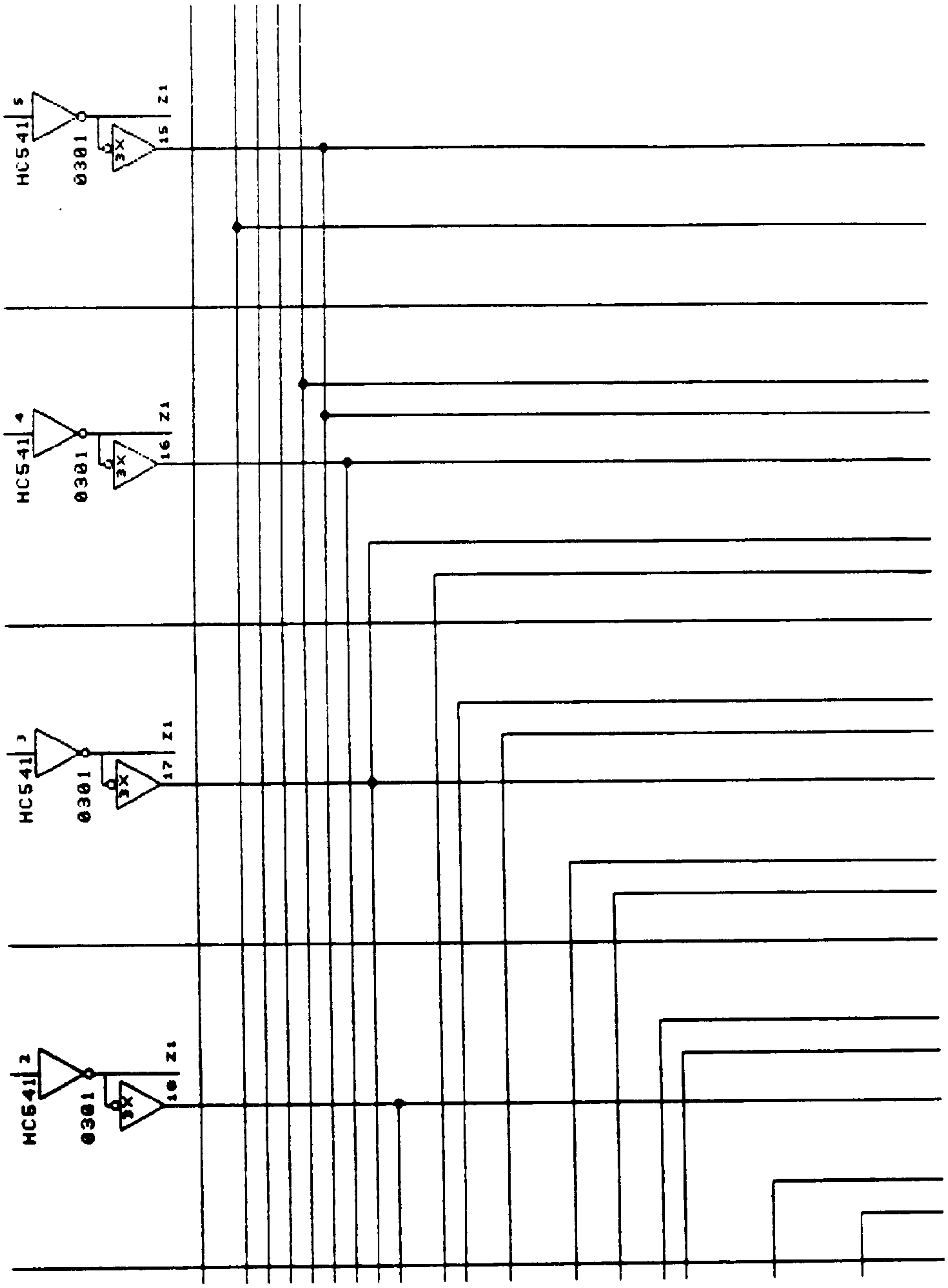

HC541
5
0301
3X
15
Z1
HC541
4
0301
3X
16
Z1
HC541
3
0301
3X
17
Z1
HC541
2
0301
3X
18
Z1

Figure 20

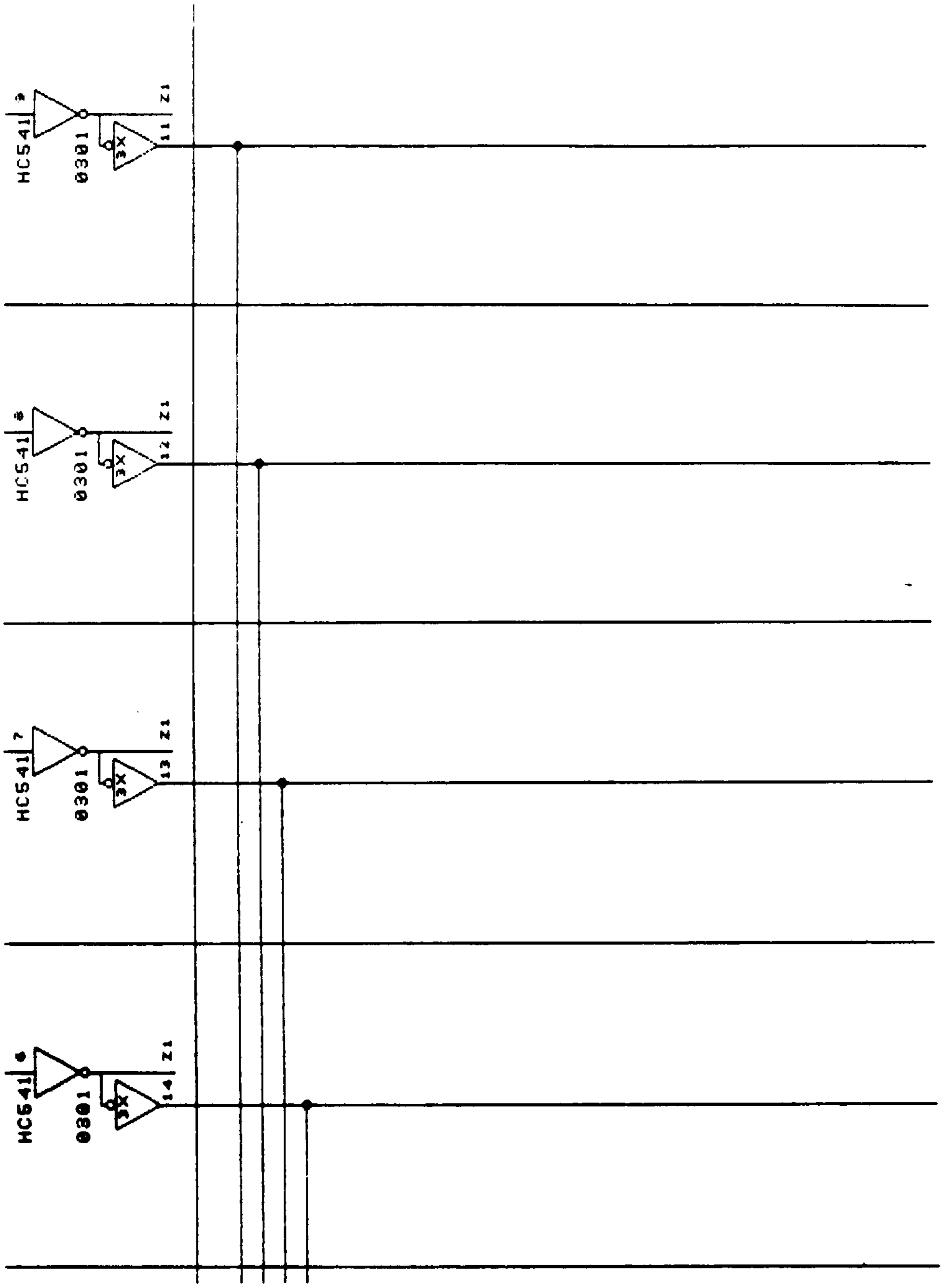

HC541
9
0301
3X
Z1
11
HC541
8
0301
3X
Z1
12
HC541
7
0301
3X
Z1
13
HC541
6
0301
3X
Z1
14

Figure 21

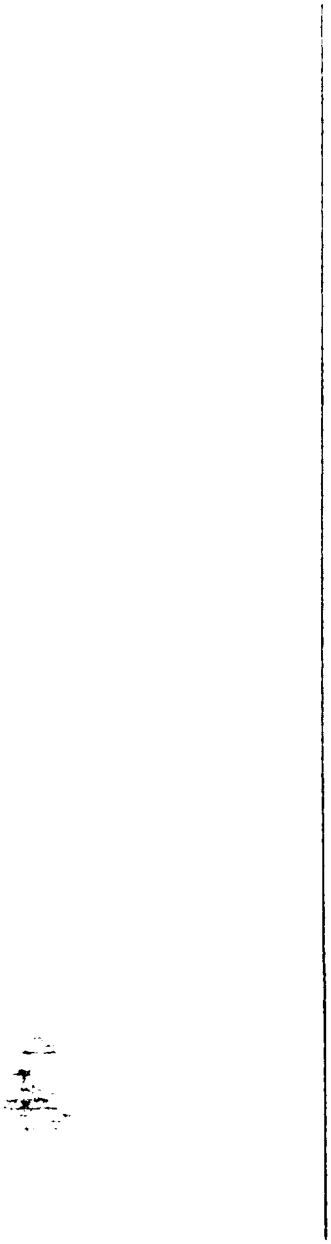


Figure 22

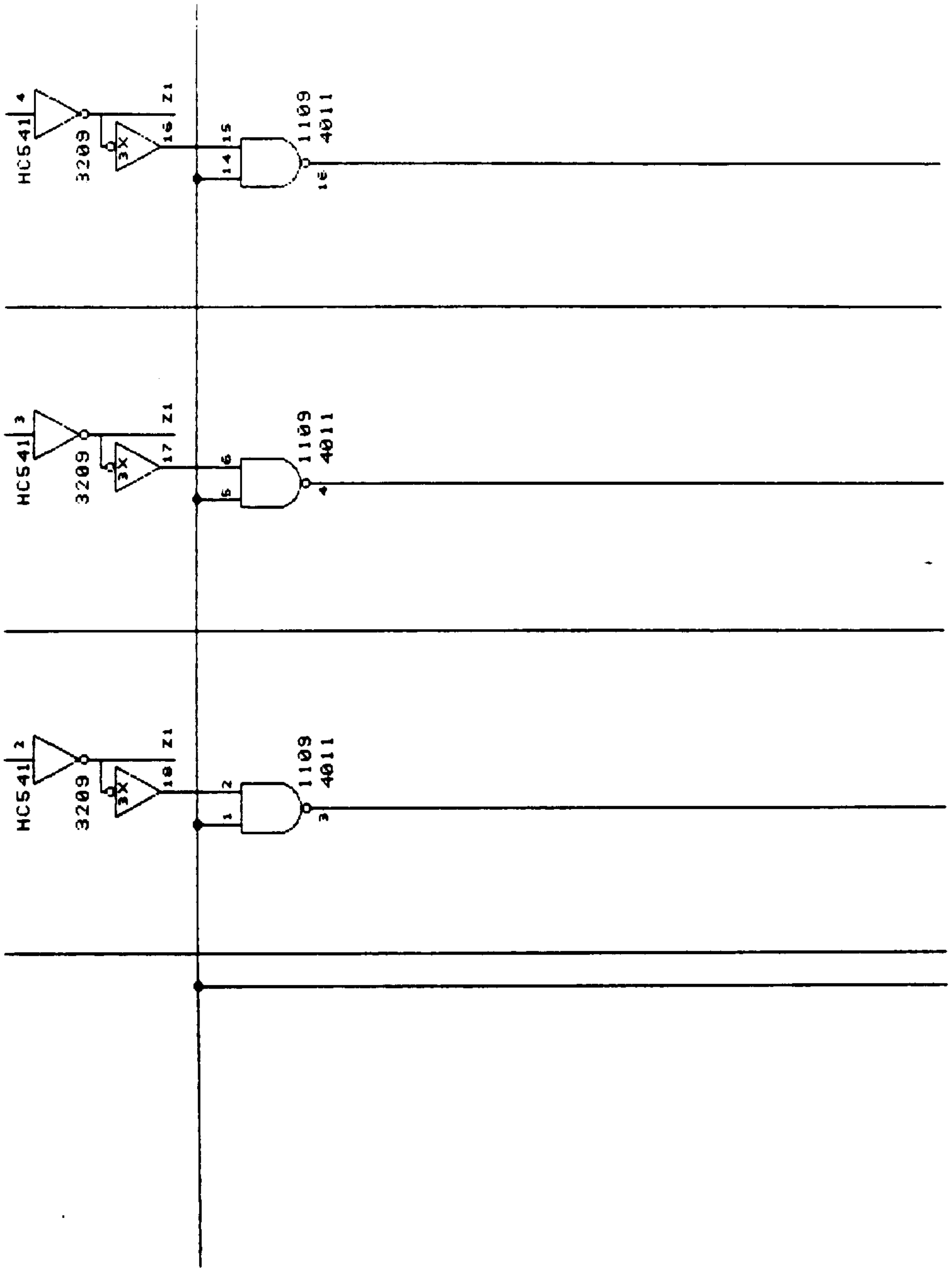

HC541
4
3209
3X
16
Z1
15
14
1109
4011
13
HC541
3
3209
3X
17
Z1
6
5
1109
4011
4
HC541
2
3209
3X
18
Z1
2
1
1109
4011
3

Figure 23

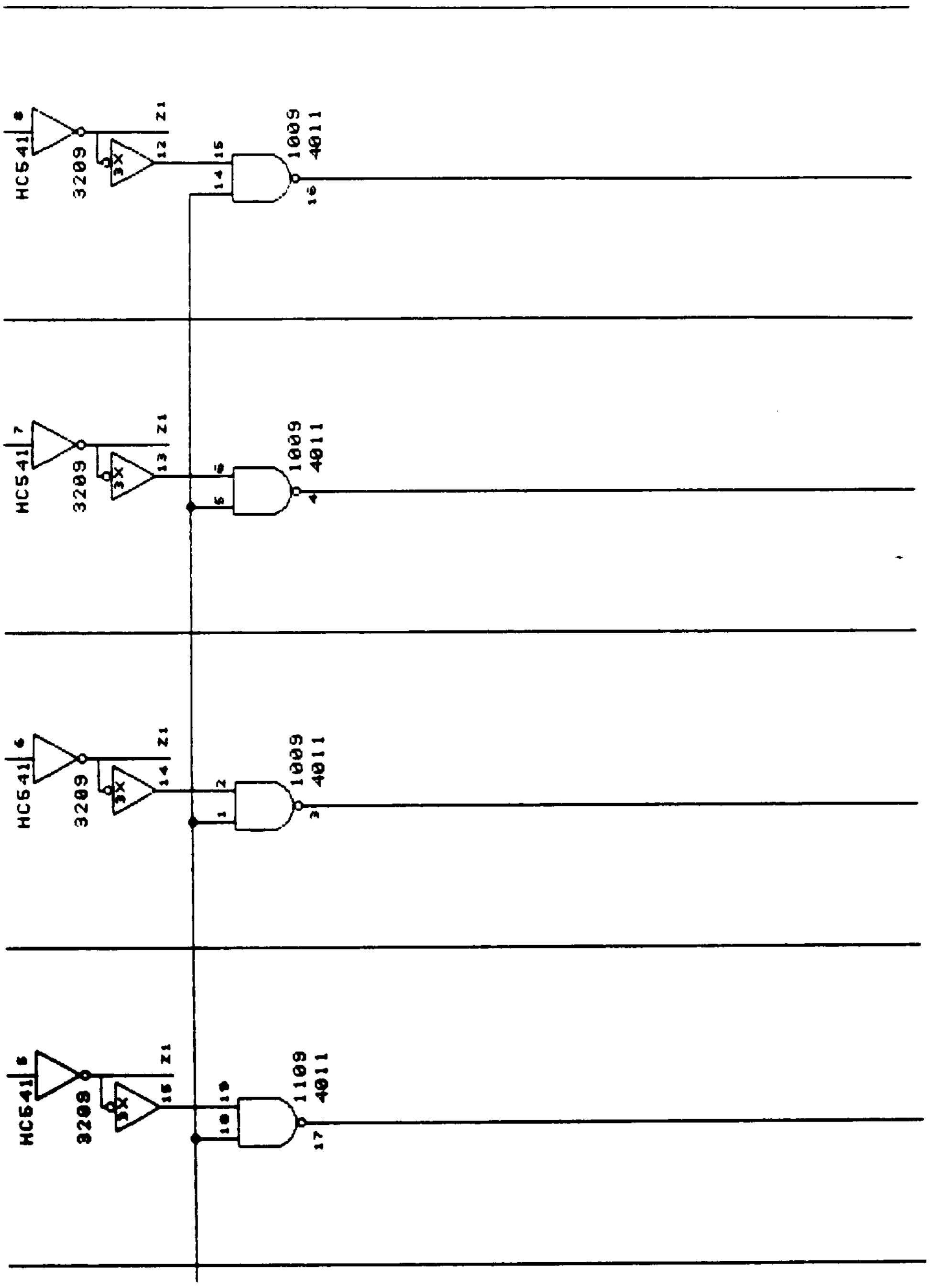

HC541
8
3209
3X
12
Z1
15
14
1009
4011
16
HC541
7
3209
3X
13
Z1
6
5
1009
4011
4
HC541
6
3209
3X
14
Z1
2
1
1009
4011
3
HC541
5
3209
3X
15
Z1
19
18
1109
4011
17

Figure 24

Figure 25

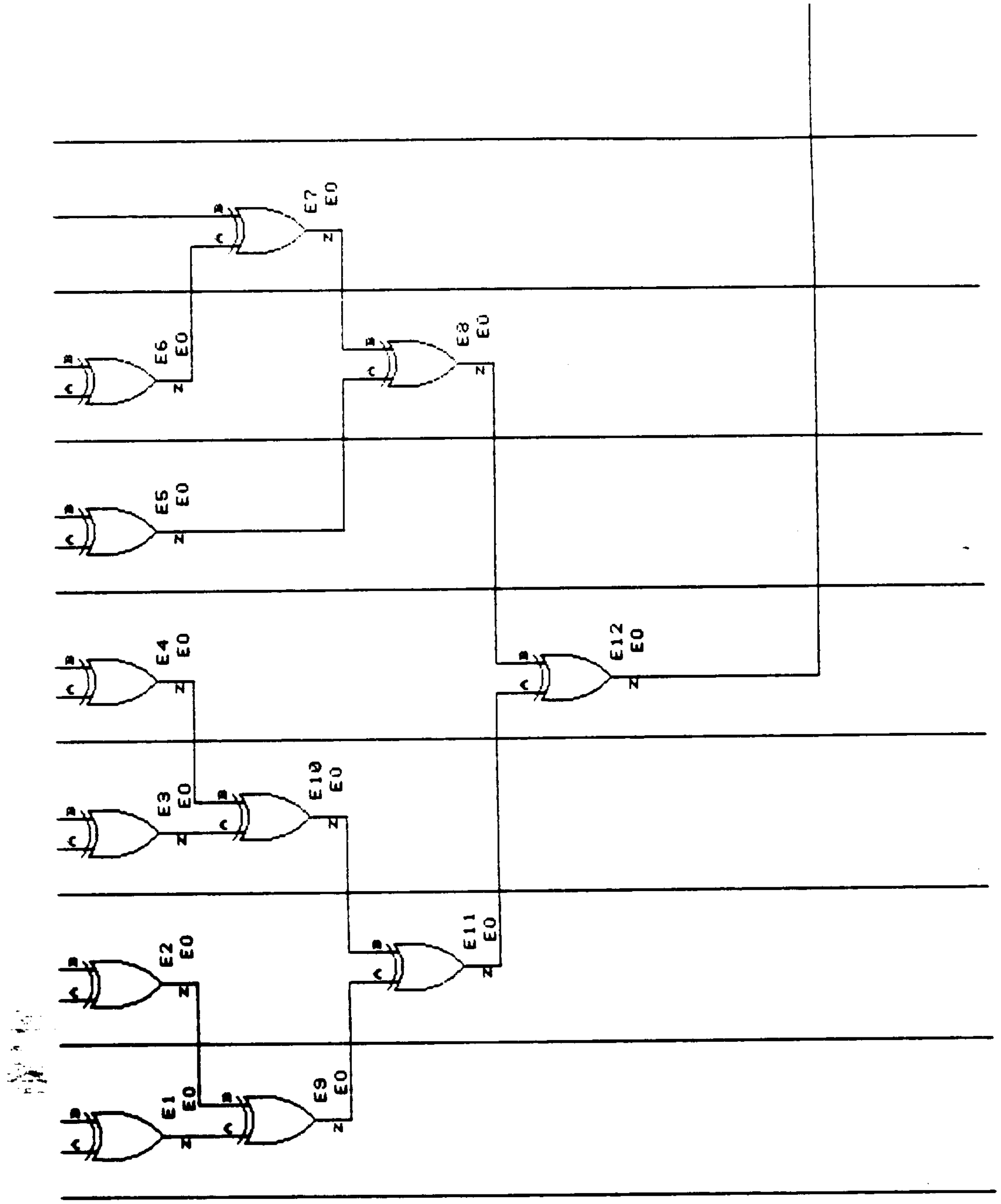

E1
E0
E2
E0
E3
E0
E4
E0
E5
E0
E6
E0
E7
E0
E8
E0
E9
E0
E10
E0
E11
E0
E12
E0
B
C
Z

Figure 26

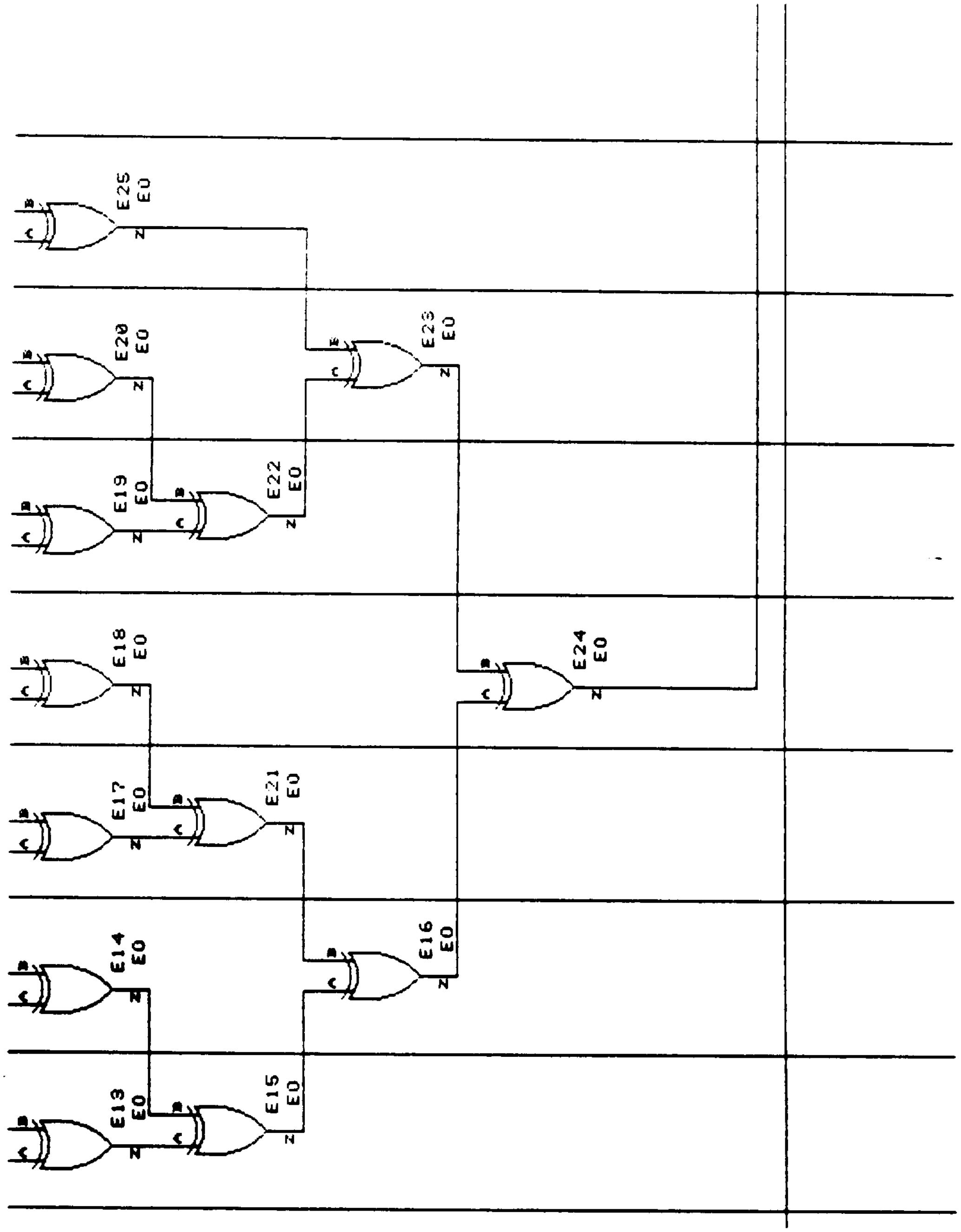

E25
E0
E20
E0
E19
E0
E22
E0
E23
E0
E18
E0
E24
E0
E17
E0
E21
E0
E14
E0
E16
E0
E13
E0
E15
E0
B
C
Z

Figure 27

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

NAS7-1028

PART 2

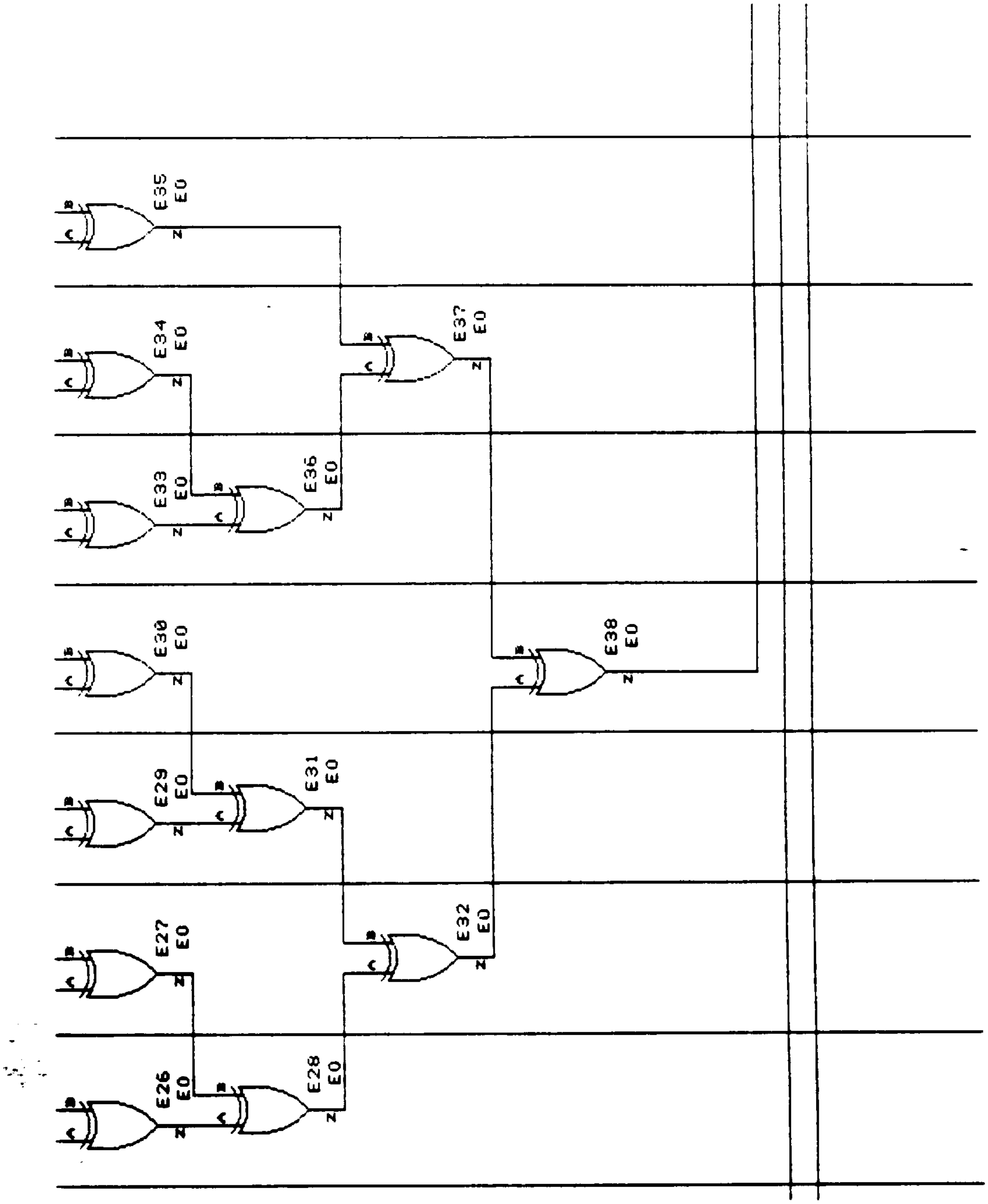

E35
EO
E34
EO
E33
EO
E36
EO
E37
EO
E30
EO
E29
EO
E31
EO
E38
EO
E27
EO
E32
EO
E26
EO
E28
EO
B
C
Z

Figure 28

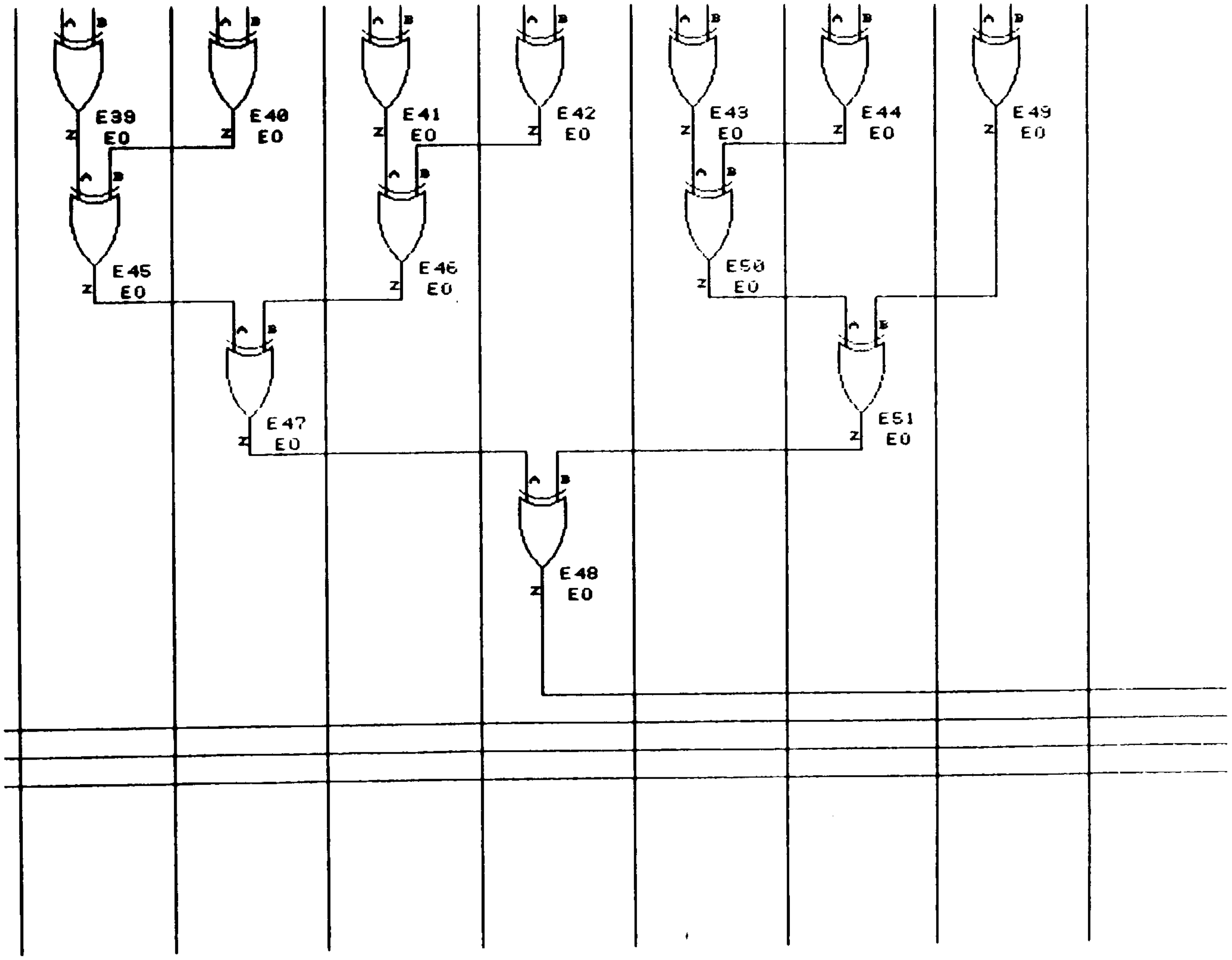

A
B
Z
E39
E0
E40
E0
E41
E0
E42
E0
E43
E0
E44
E0
E49
E0
E45
E0
E46
E0
E50
E0
E47
E0
E51
E0
E48
E0

Figure 29

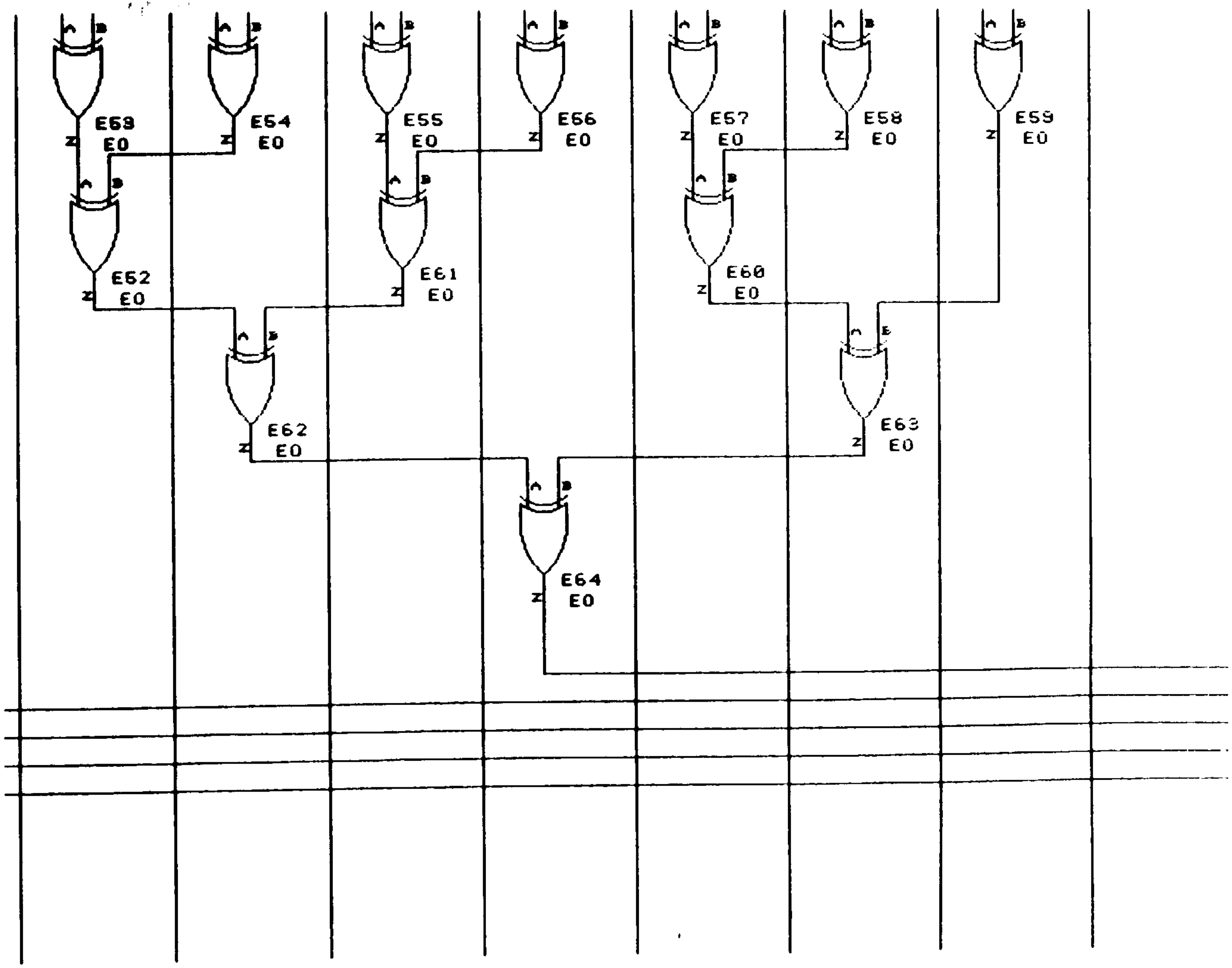

E53
E0
E54
E0
E55
E0
E56
E0
E57
E0
E58
E0
E59
E0
E52
E0
E61
E0
E60
E0
E62
E0
E63
E0
E64
E0
A
B
Z

Figure 30

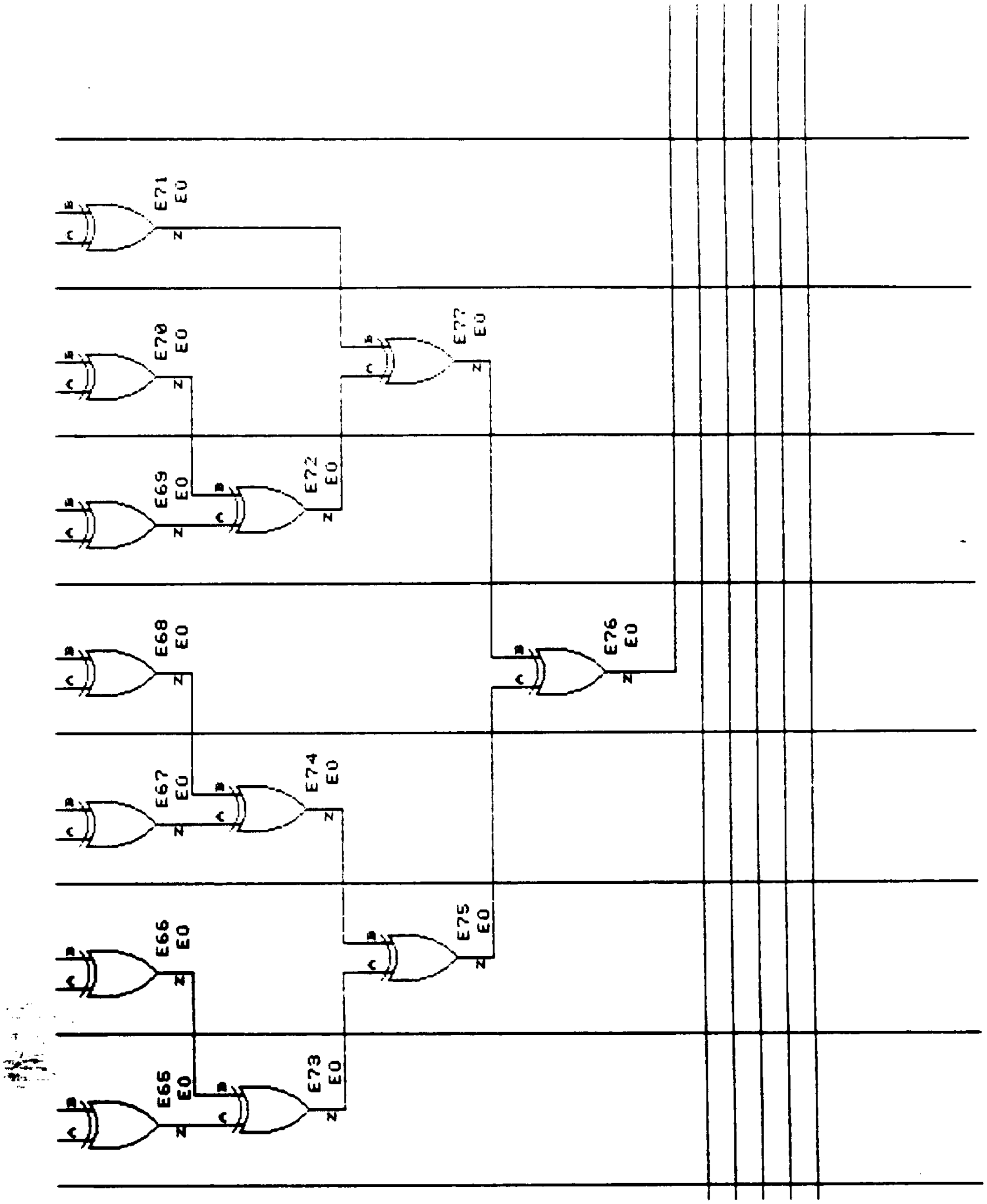

E71
E0
E70
E0
E77
E0
E69
E0
E72
E0
E68
E0
E76
E0
E67
E0
E74
E0
E66
E0
E75
E0
E65
E0
E73
E0
B
C
Z

Figure 31

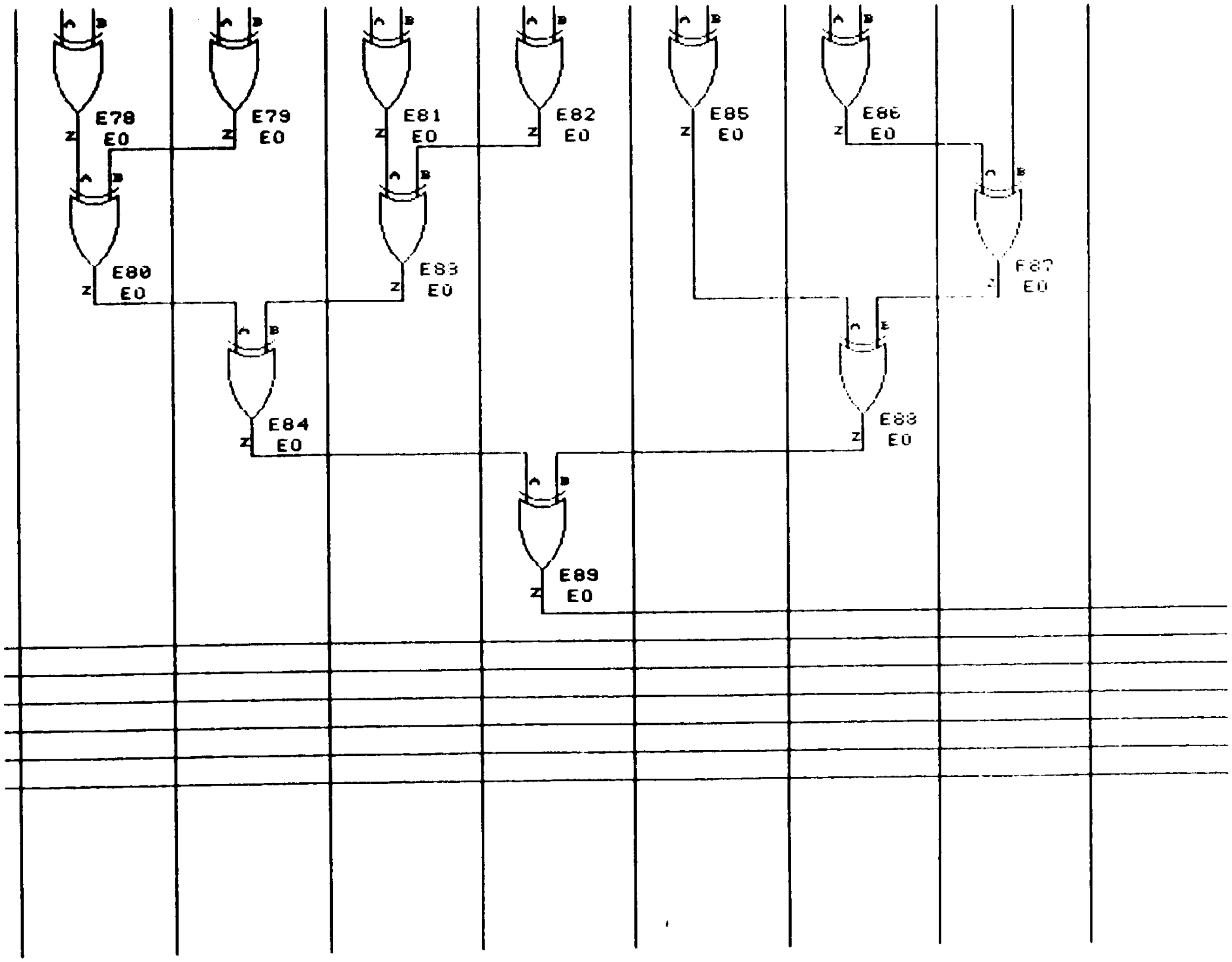

A
B
Z
E78
E0
E79
E0
E81
E0
E82
E0
E85
E0
E86
E0
E80
E0
E83
E0
E87
E0
E84
E0
E88
E0
E89
E0

Figure 32

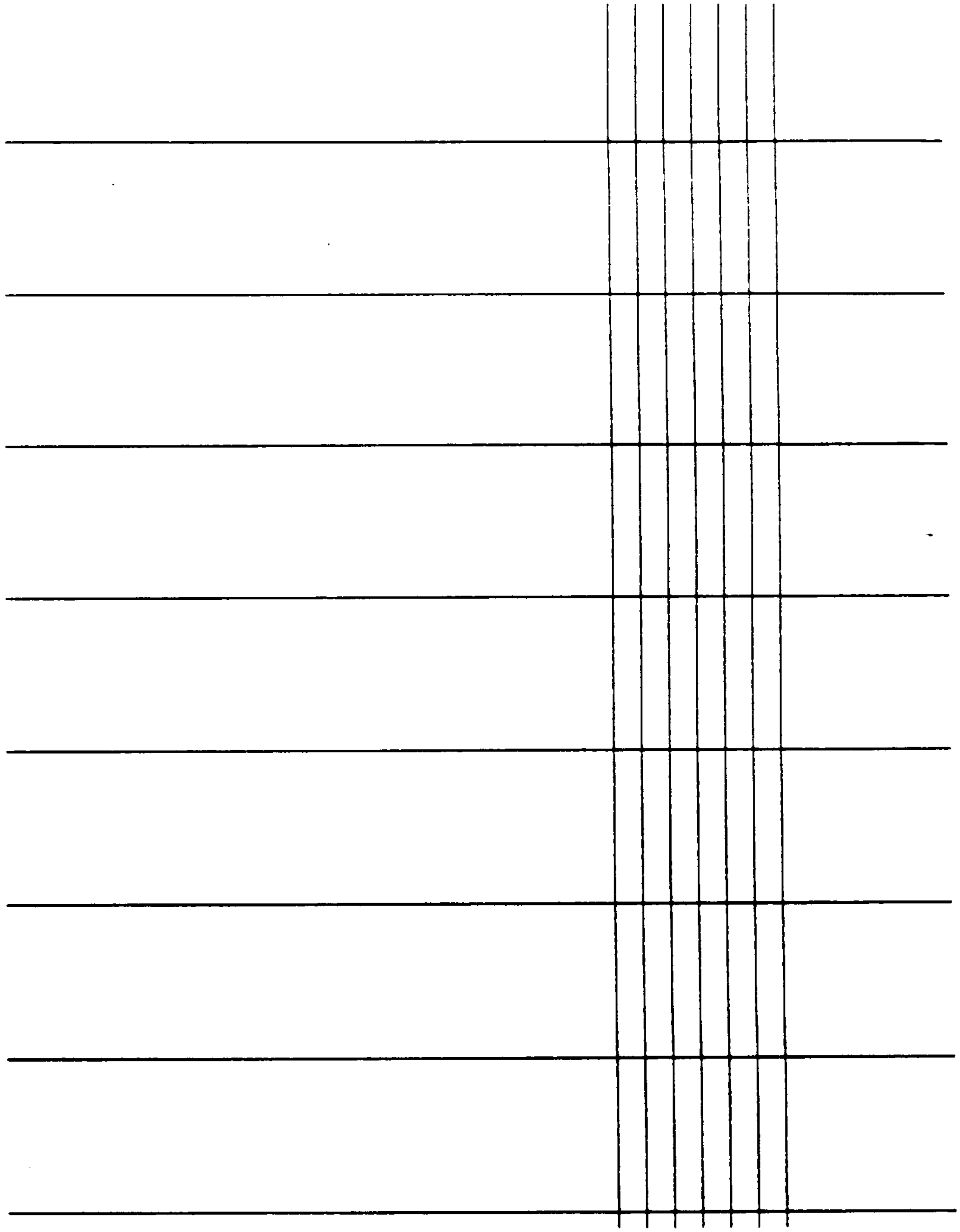


Figure 33

Figure 34

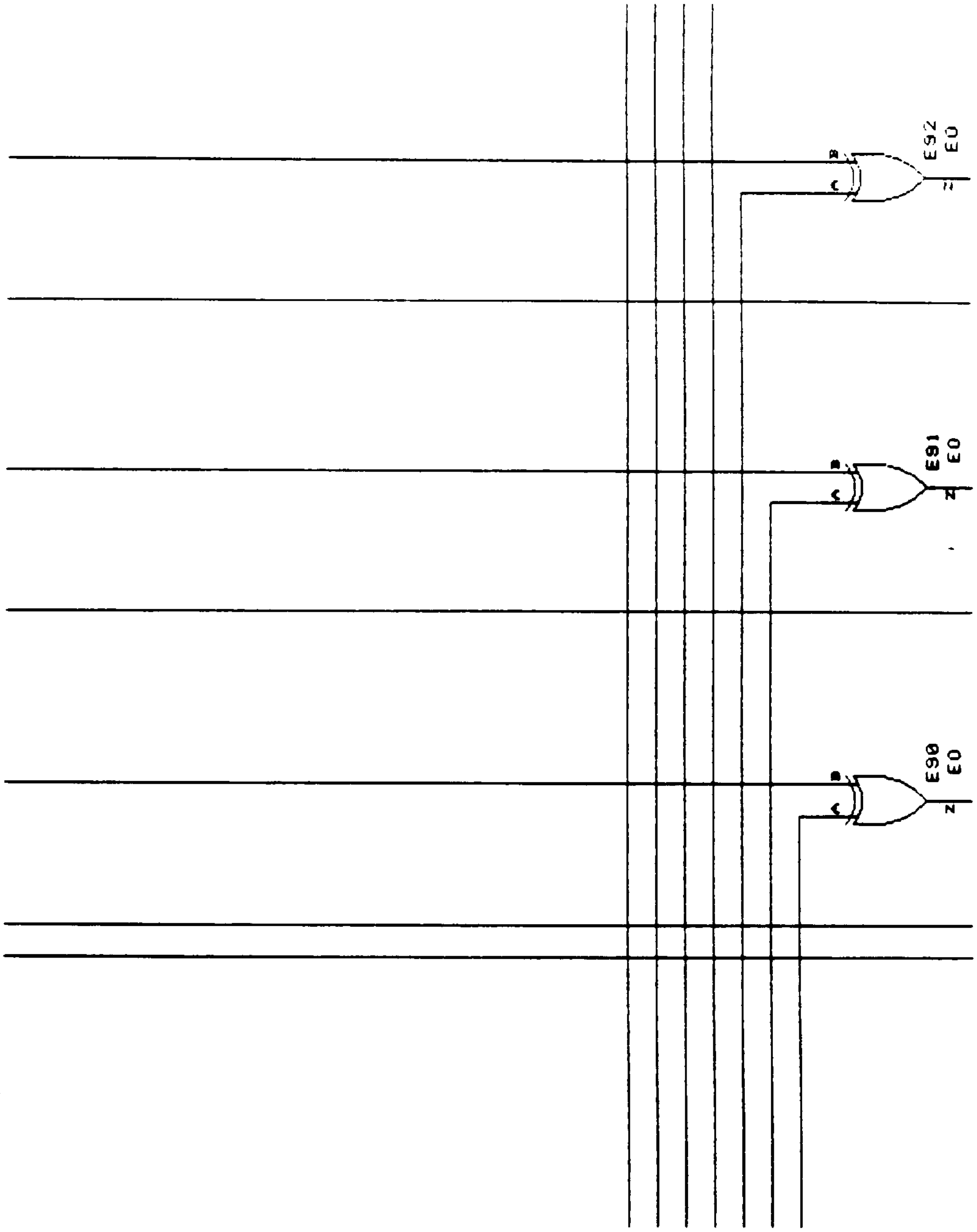


Figure 36

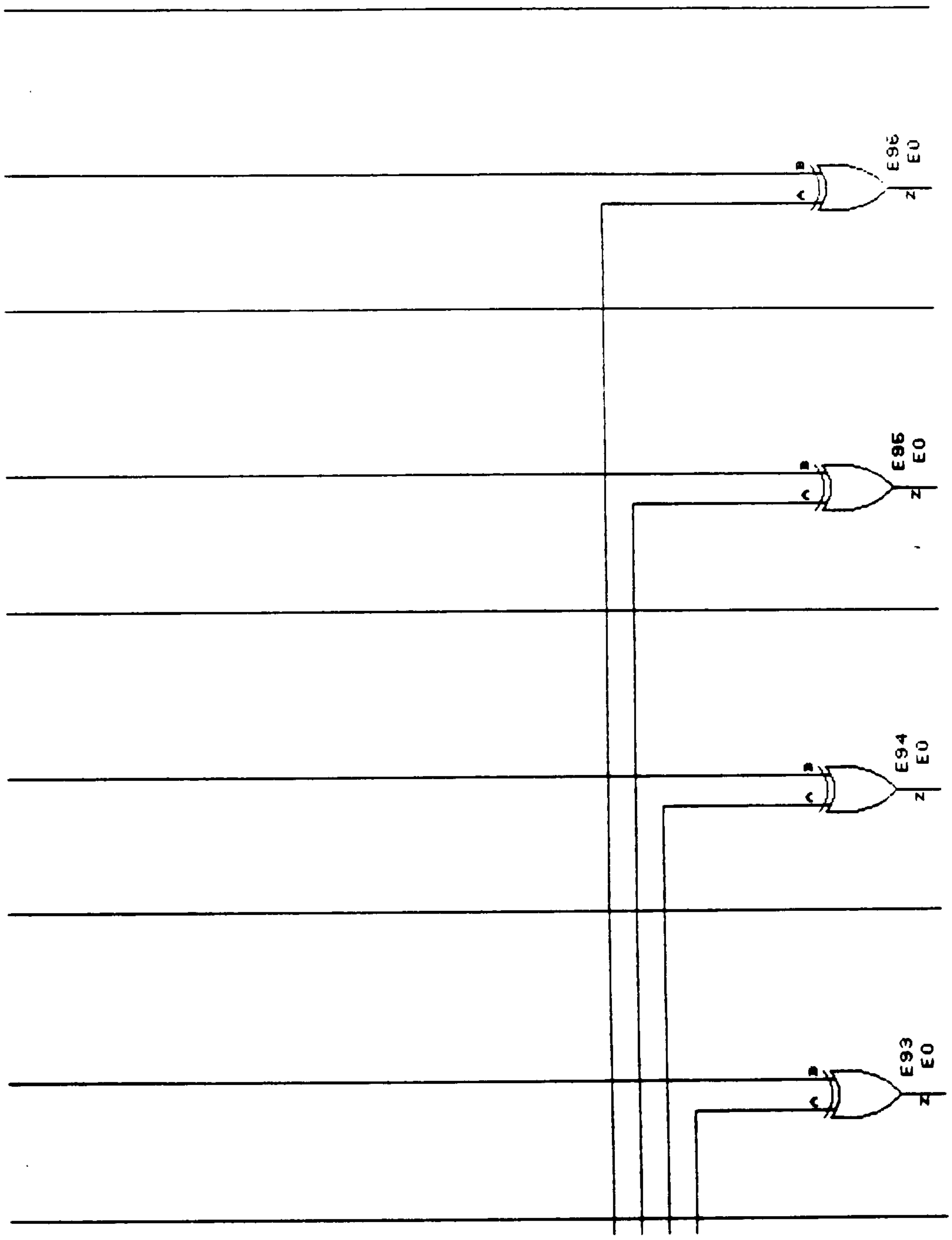


Figure 36

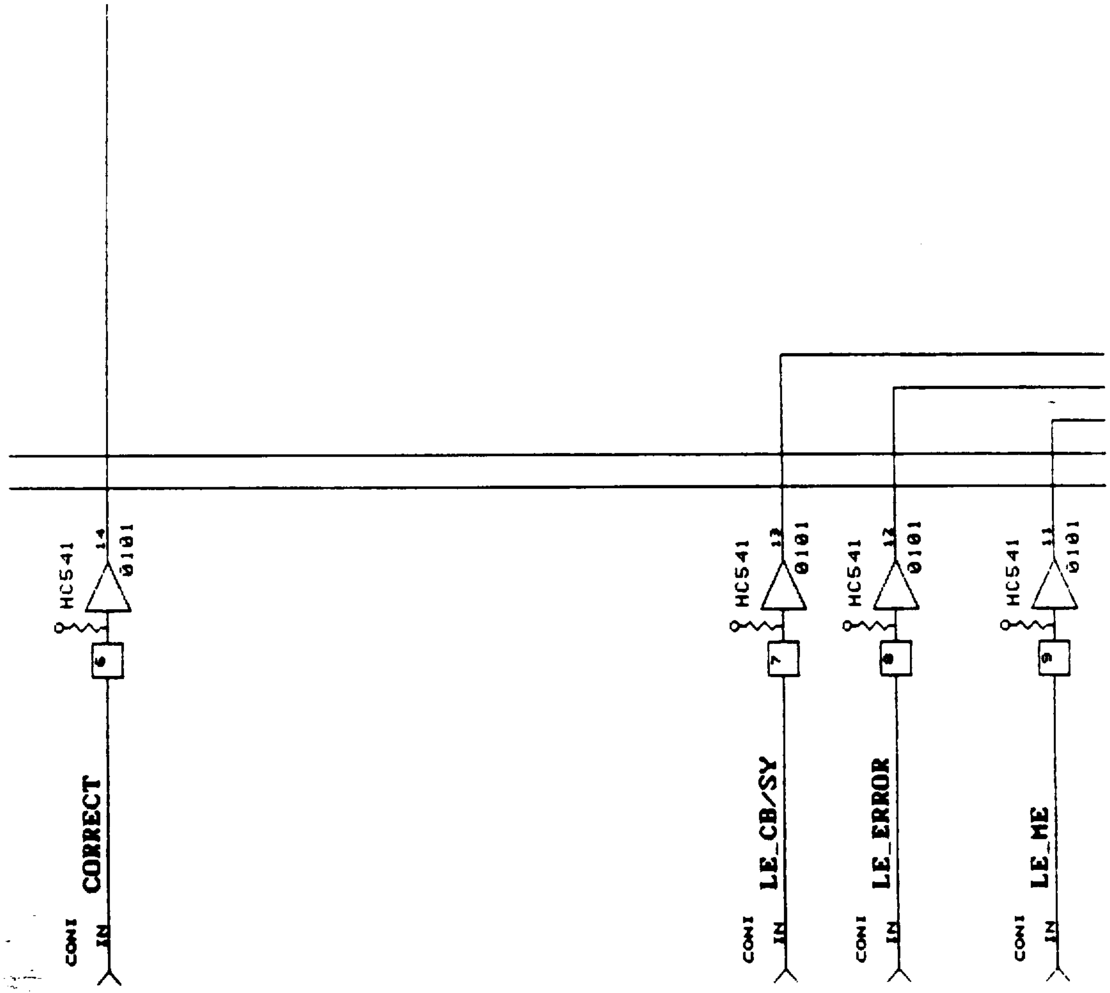

CON1
IN
CORRECT
6
HC541
14
U101
CON1
IN
LE_CB/SY
7
HC541
13
U101
CON1
IN
LE_ERROR
8
HC541
12
U101
CON1
IN
LE_ME
9
HC541
11
U101

Figure 37

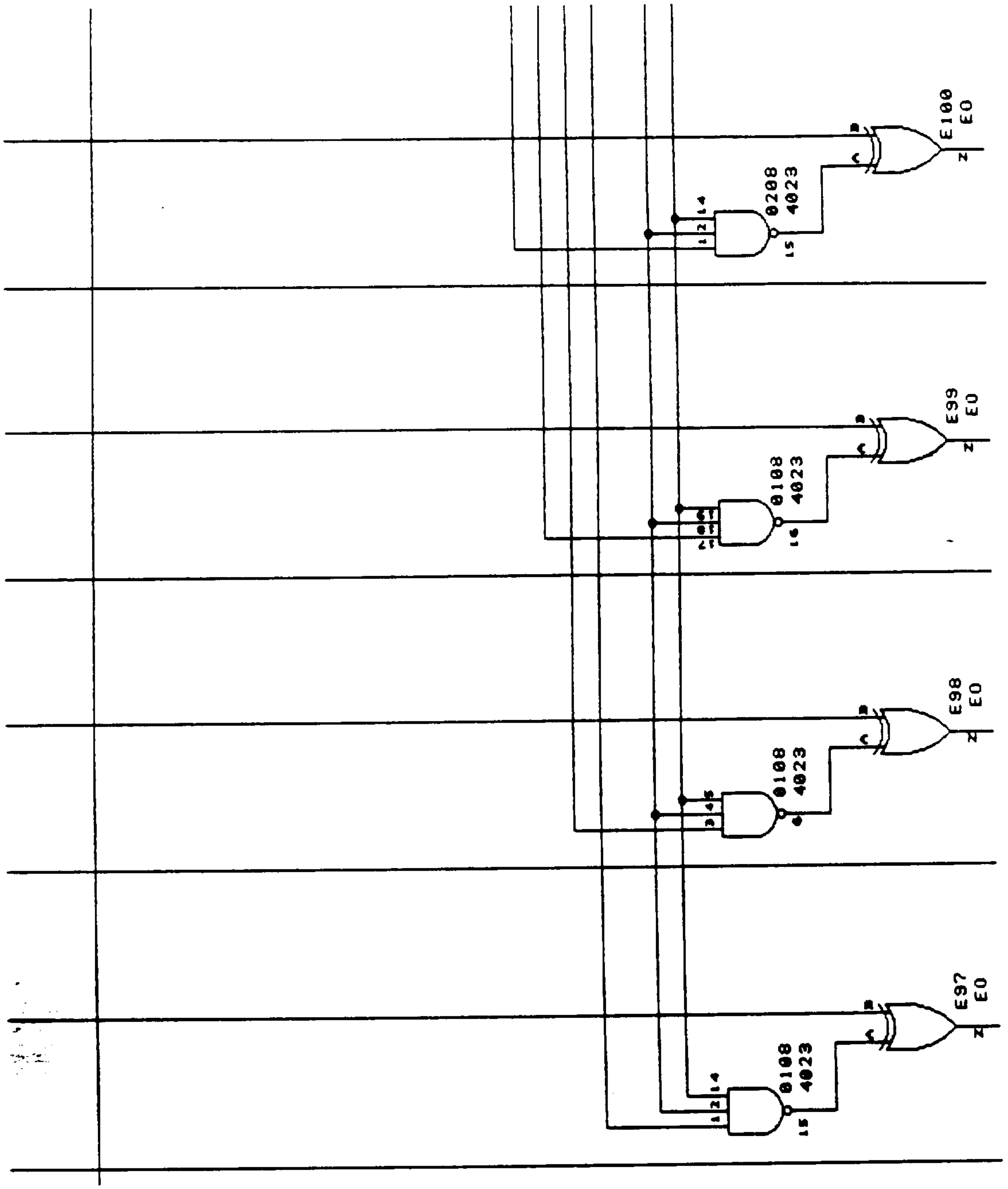

E100
EO
0208
4023
E99
EO
0108
4023
E98
EO
0108
4023
E97
EO
0108
4023
A
B
Z

Figure 38

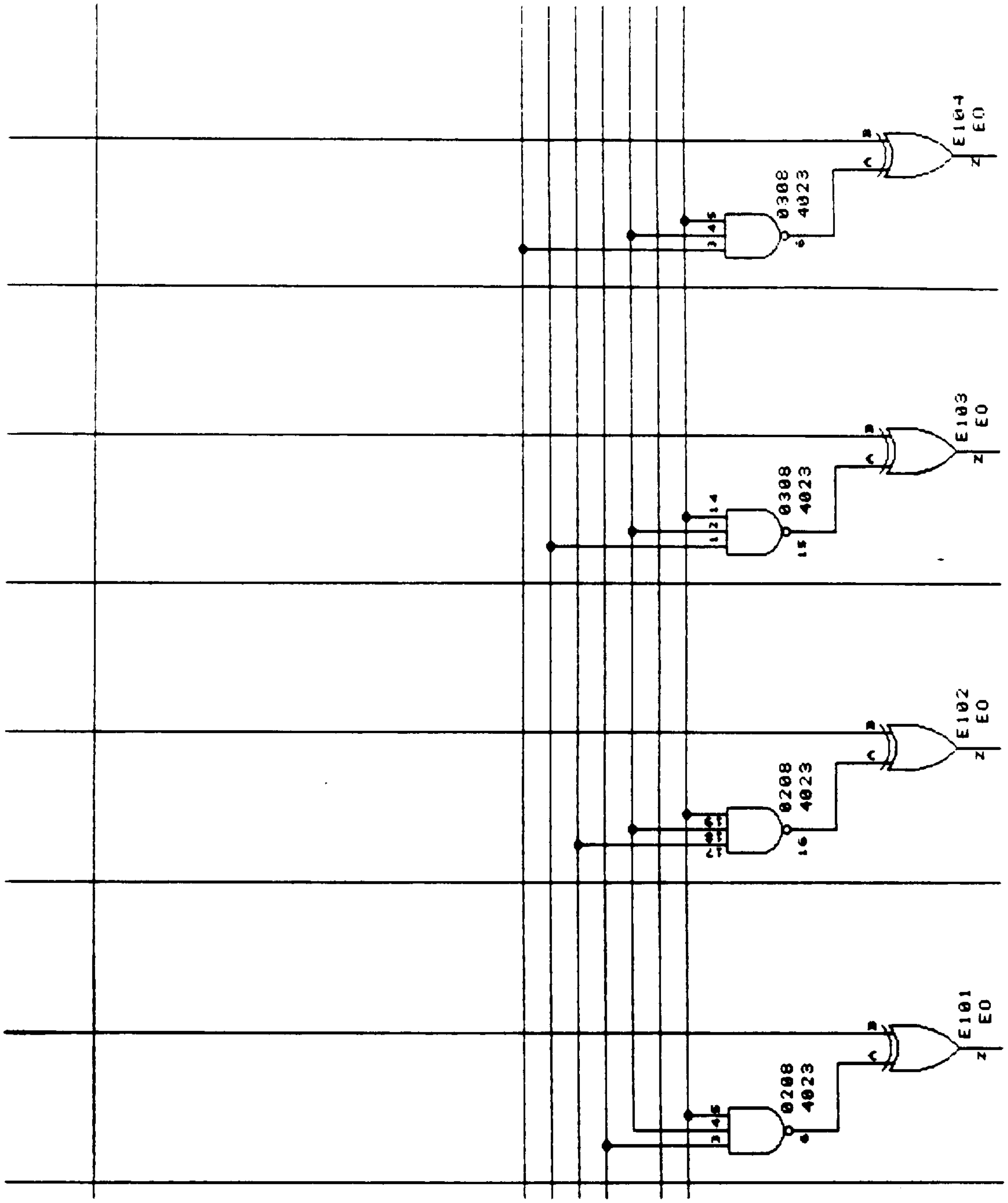

3 4 5
0208
4023
6
B
A
E101
EO
Z
2 3 4
0208
4023
16
B
A
E102
EO
Z
1 2 14
0308
4023
15
B
A
E103
EO
Z
3 4 5
0308
4023
6
B
A
E104
EO
Z

Figure 39

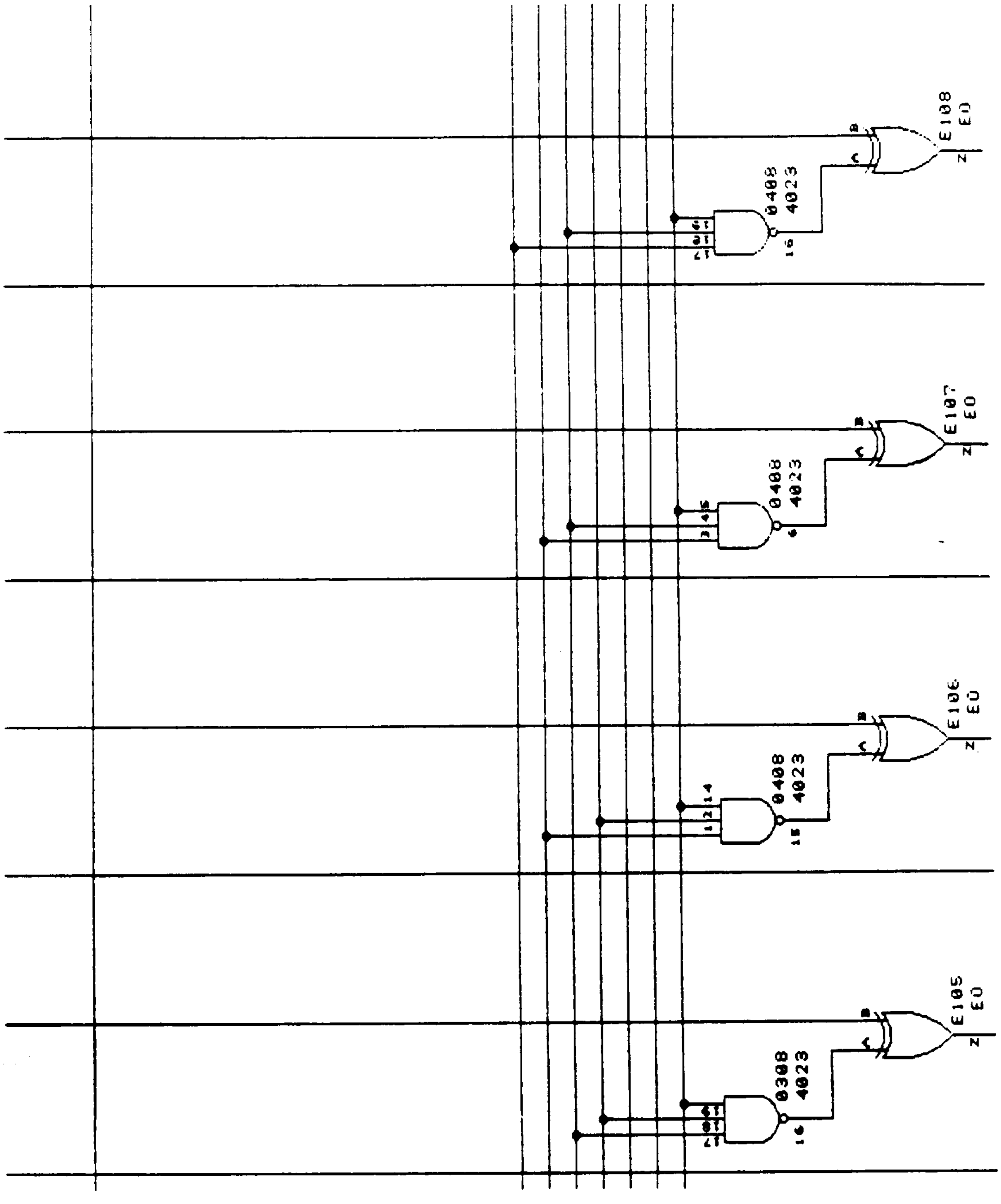

E108
EO
0408
4023
E107
EO
0408
4023
E106
EO
0408
4023
E105
EO
0308
4023
B
A
Z

Figure 40

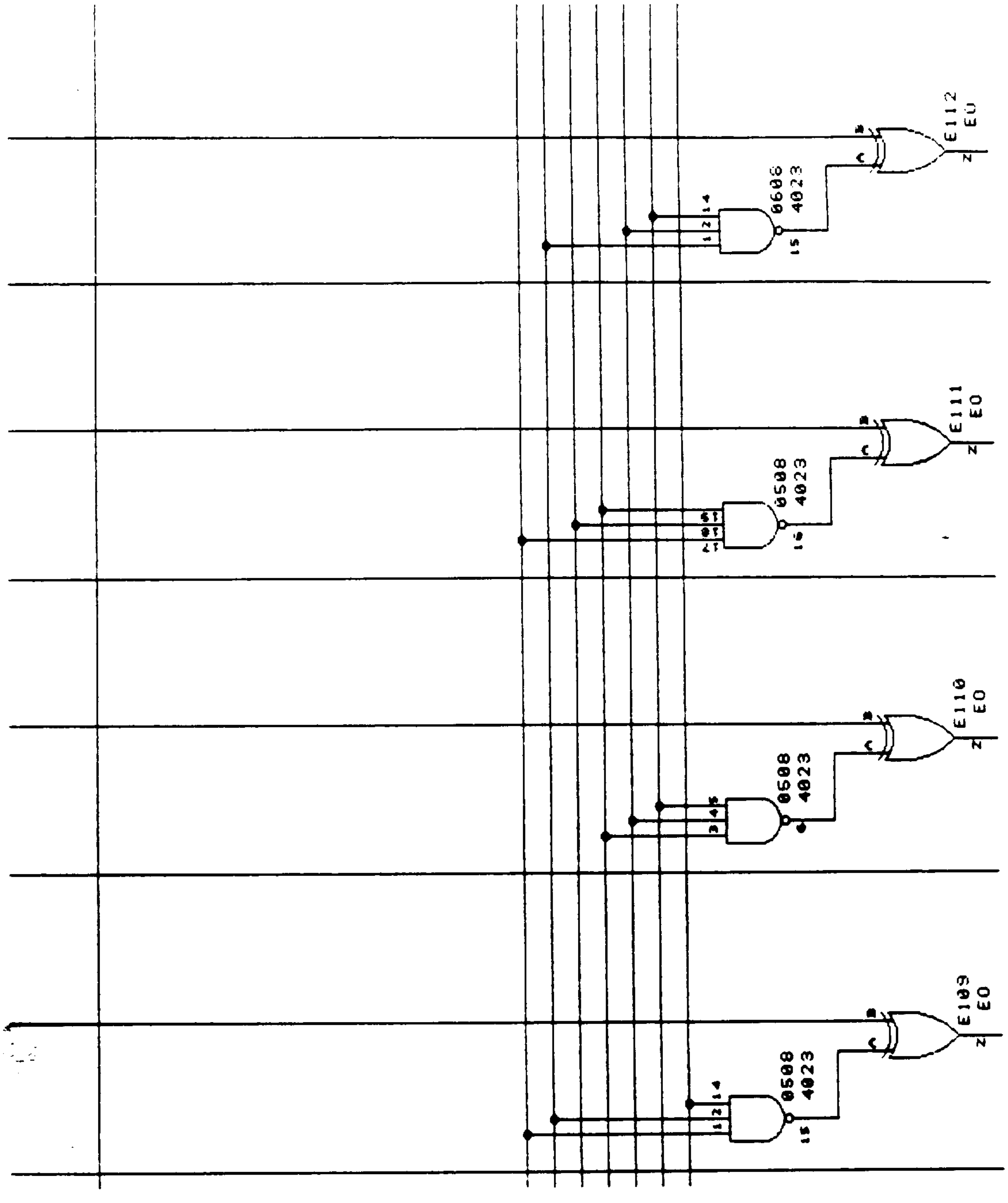

E112
EO
B
C
Z
0606
4023
1
2
14
15
E111
EO
B
C
Z
0508
4023
13
12
11
10
E110
EO
B
C
Z
0608
4023
3
4
5
6
E109
EO
B
C
Z
0508
4023
1
2
14
15

Figure 41

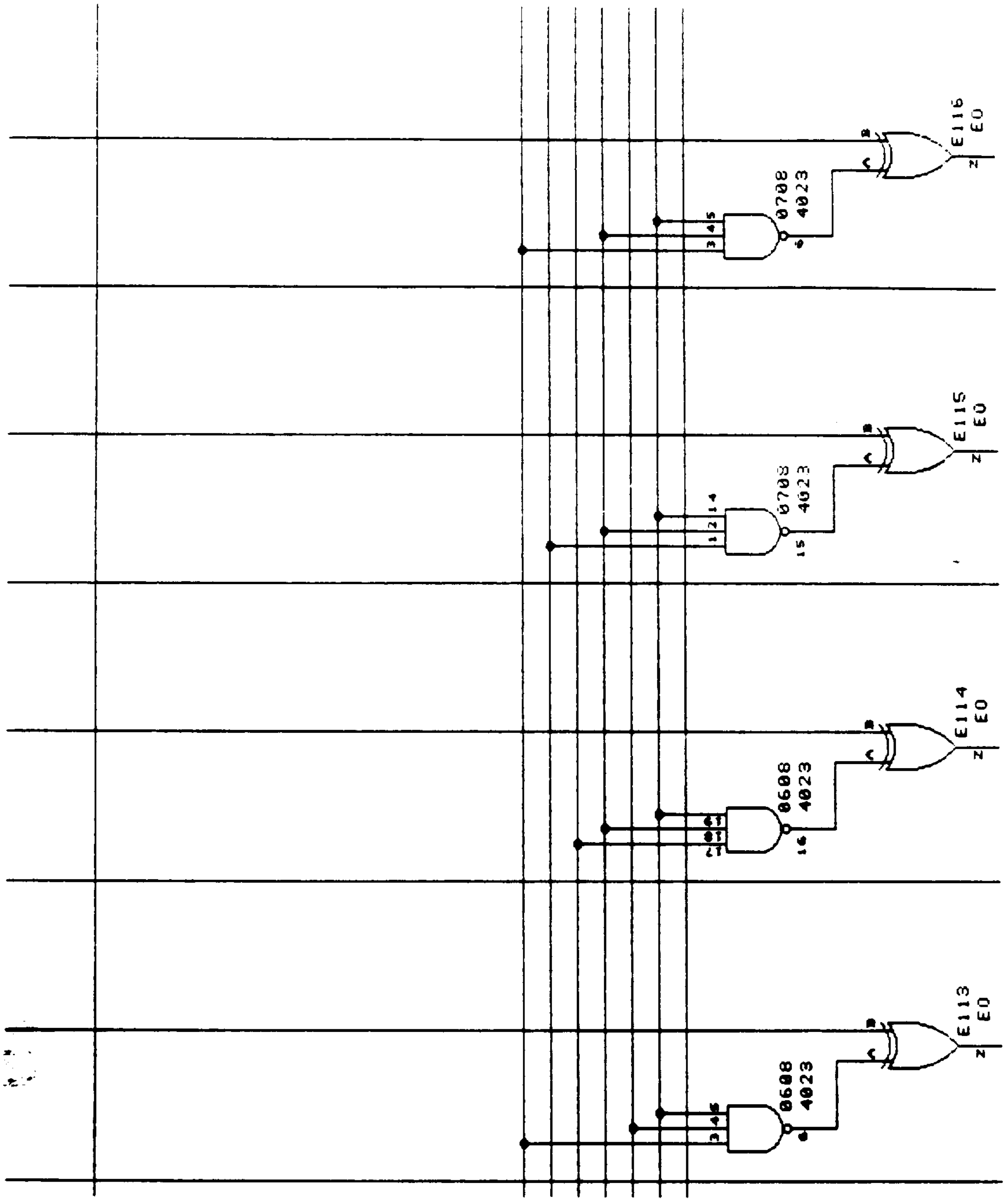

E116
EO
0708
4023
3 4 5
6
A
B
Z
E115
EO
0708
4023
12 13 14
15
E114
EO
0608
4023
16
E113
EO
0608
4023
3 4 5
6

Figure 42

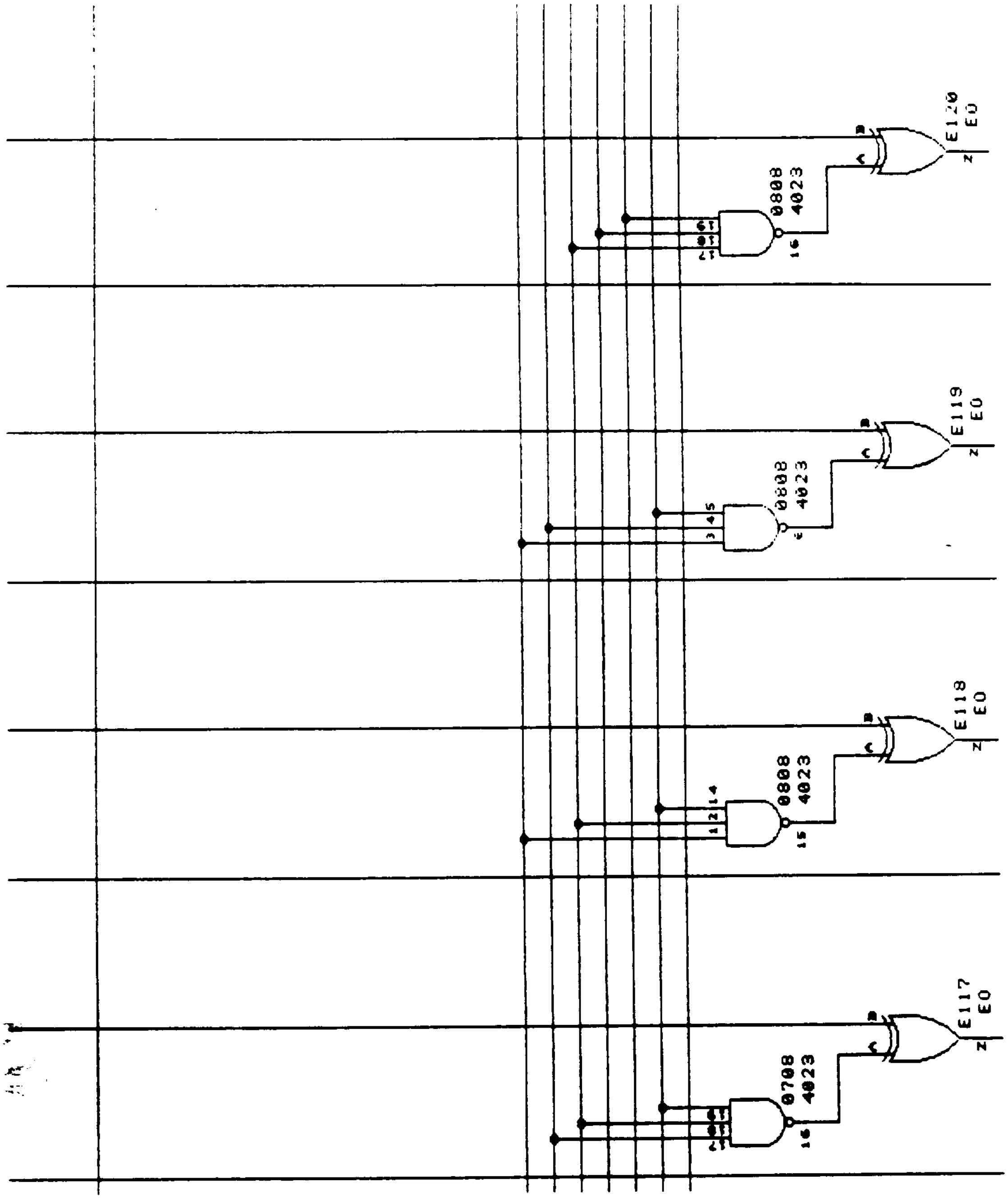

E120
EO
0808
4023
16
E119
EO
0808
4023
3
4
5
E118
EO
0808
4023
12
14
15
E117
EO
0708
4023
16
A
B
Z

Figure 43

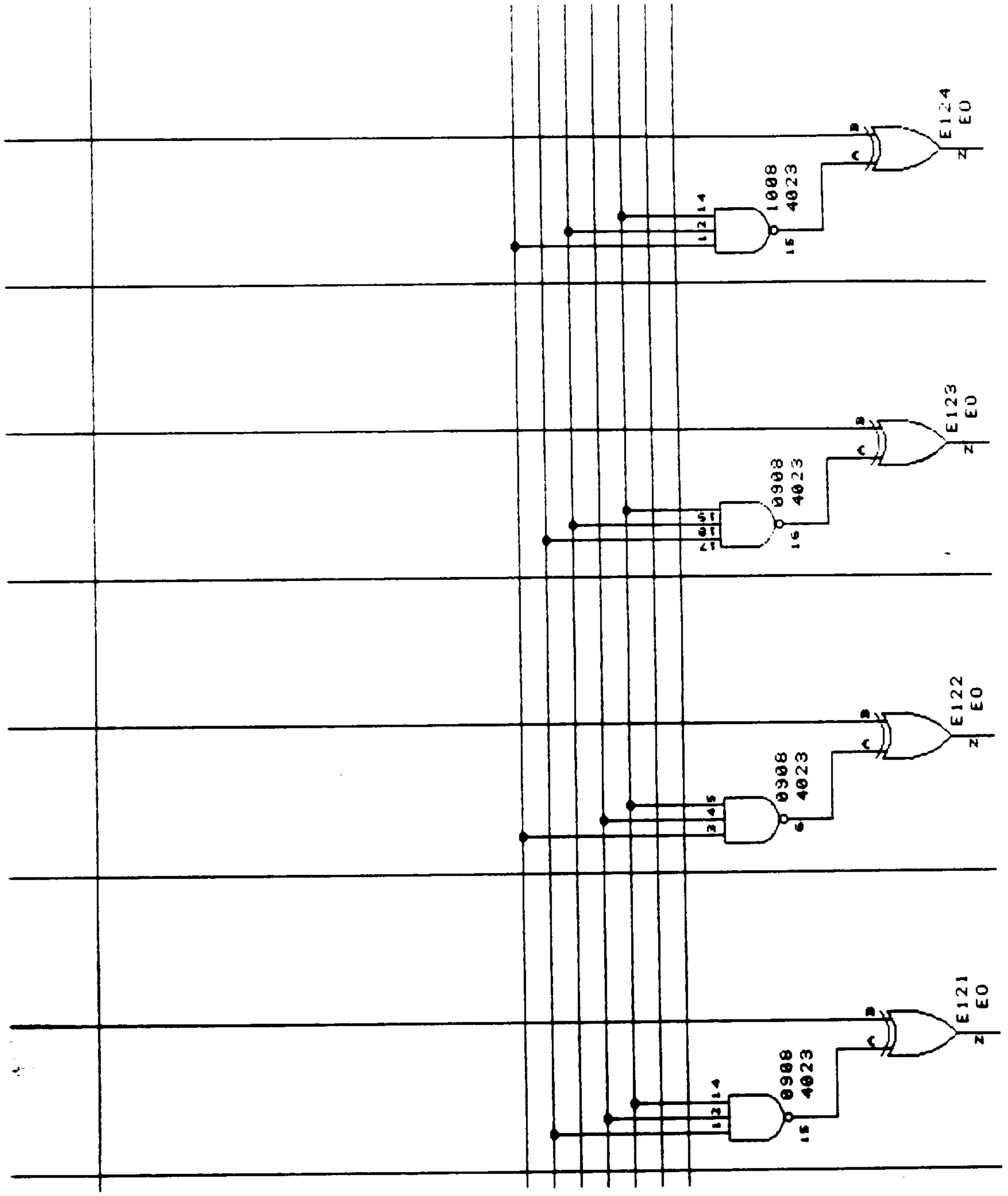

E124
EO
1008
4023
B
C
Z
E123
EO
0908
4023
16
E122
EO
0908
4023
6
E121
EO
0908
4023
15

Figure 44

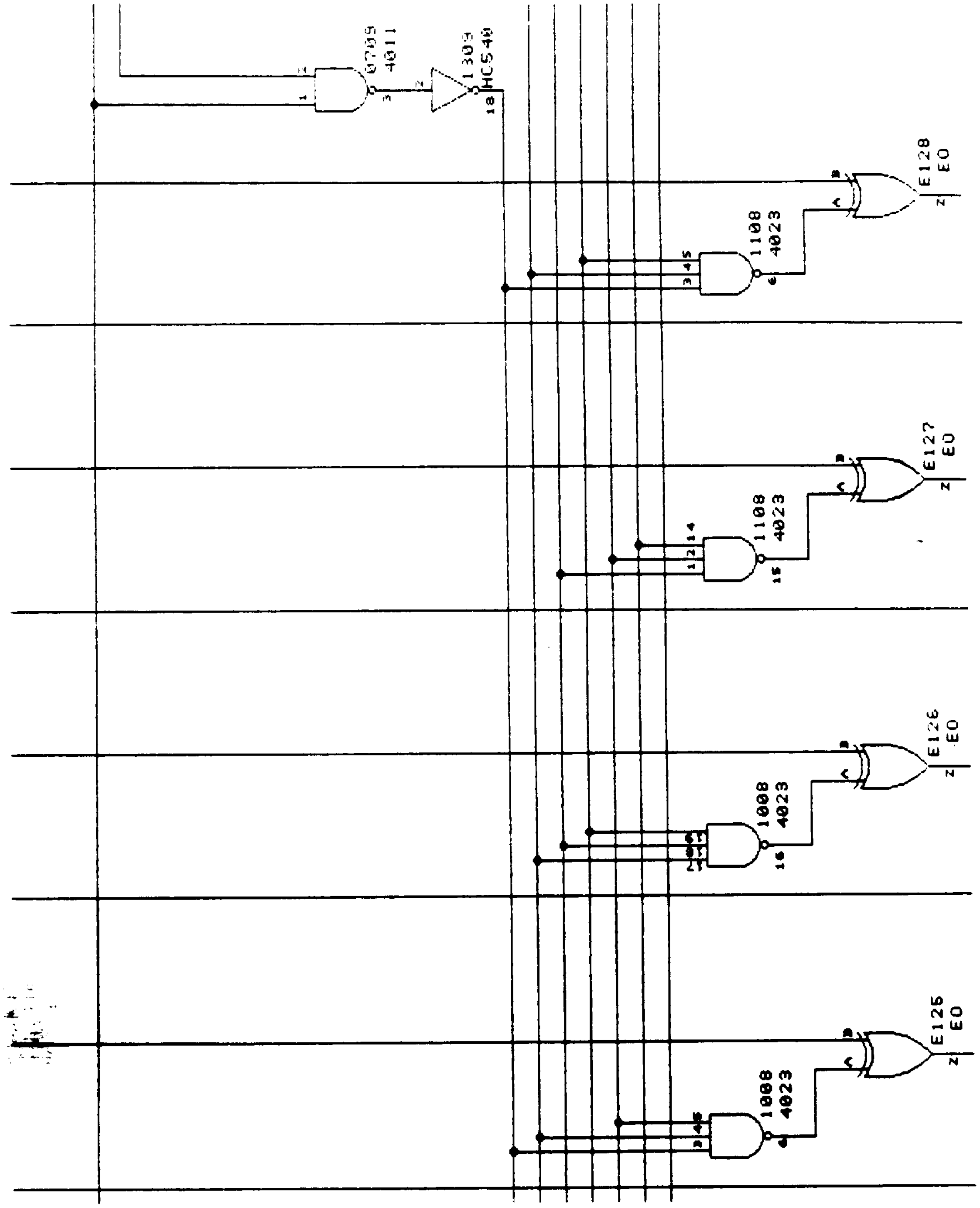

0709
4011
1309
HC540
1108
4023
1108
4023
1008
4023
1008
4023
E128
EO
E127
EO
E126
EO
E125
EO
A
B
Z

Figure 45

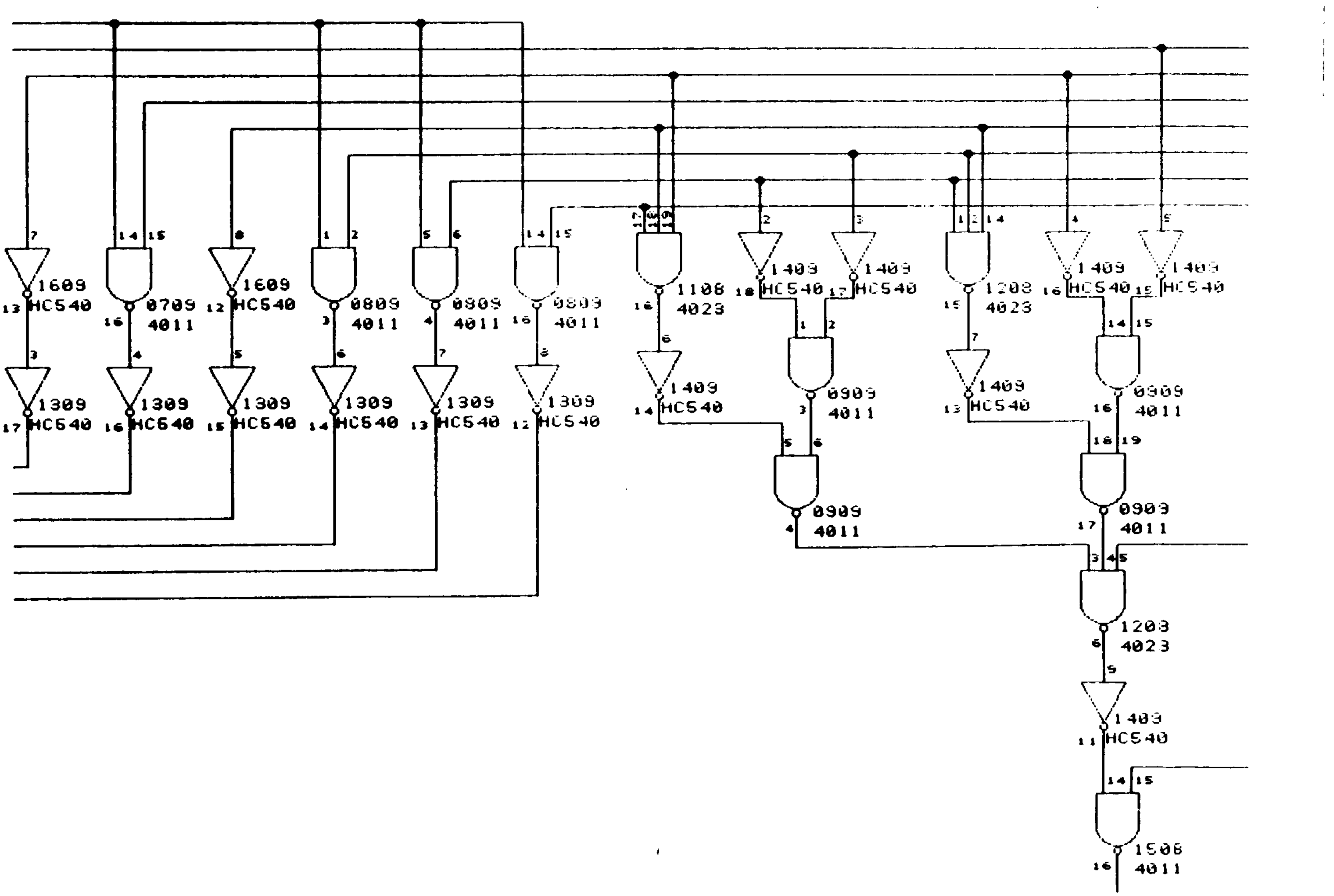

1609 HC540
0709 4011
1609 HC540
0809 4011
0809 4011
0809 4011
1309 HC540
1309 HC540
1309 HC540
1309 HC540
1309 HC540
1309 HC540
1108 4023
1409 HC540
1409 HC540
1409 HC540
0909 4011
0909 4011
1208 4023
1409 HC540
1409 HC540
1409 HC540
0909 4011
0909 4011
1208 4023
1408 HC540
1508 4011

Figure 46

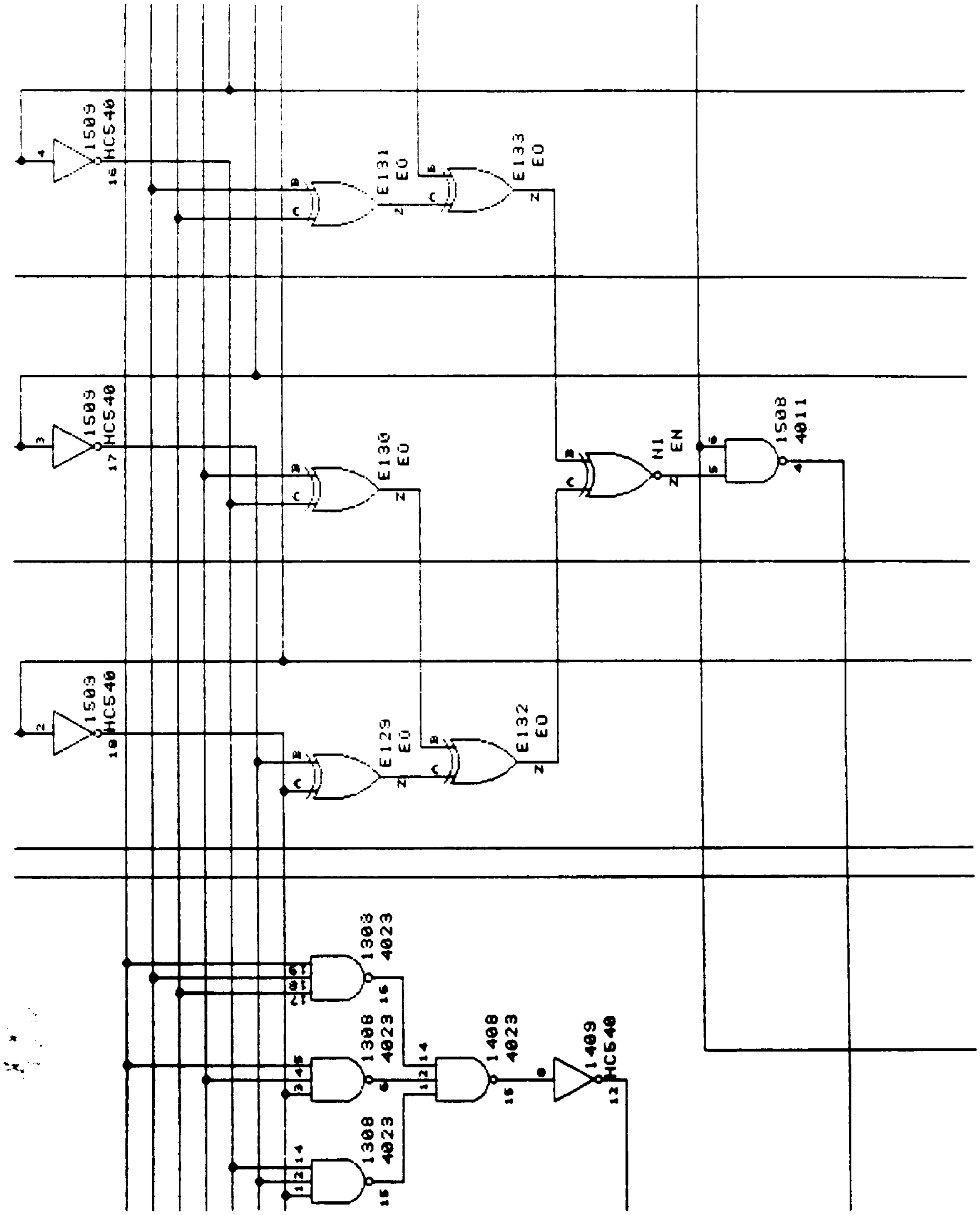

1509
HC540
1509
HC540
1509
HC540
E131
EO
E133
EO
E130
EO
N1
EN
1508
4011
E129
EO
E132
EO
1308
4023
1308
4023
1308
4023
1408
4023
1409
HC540

Figure 47

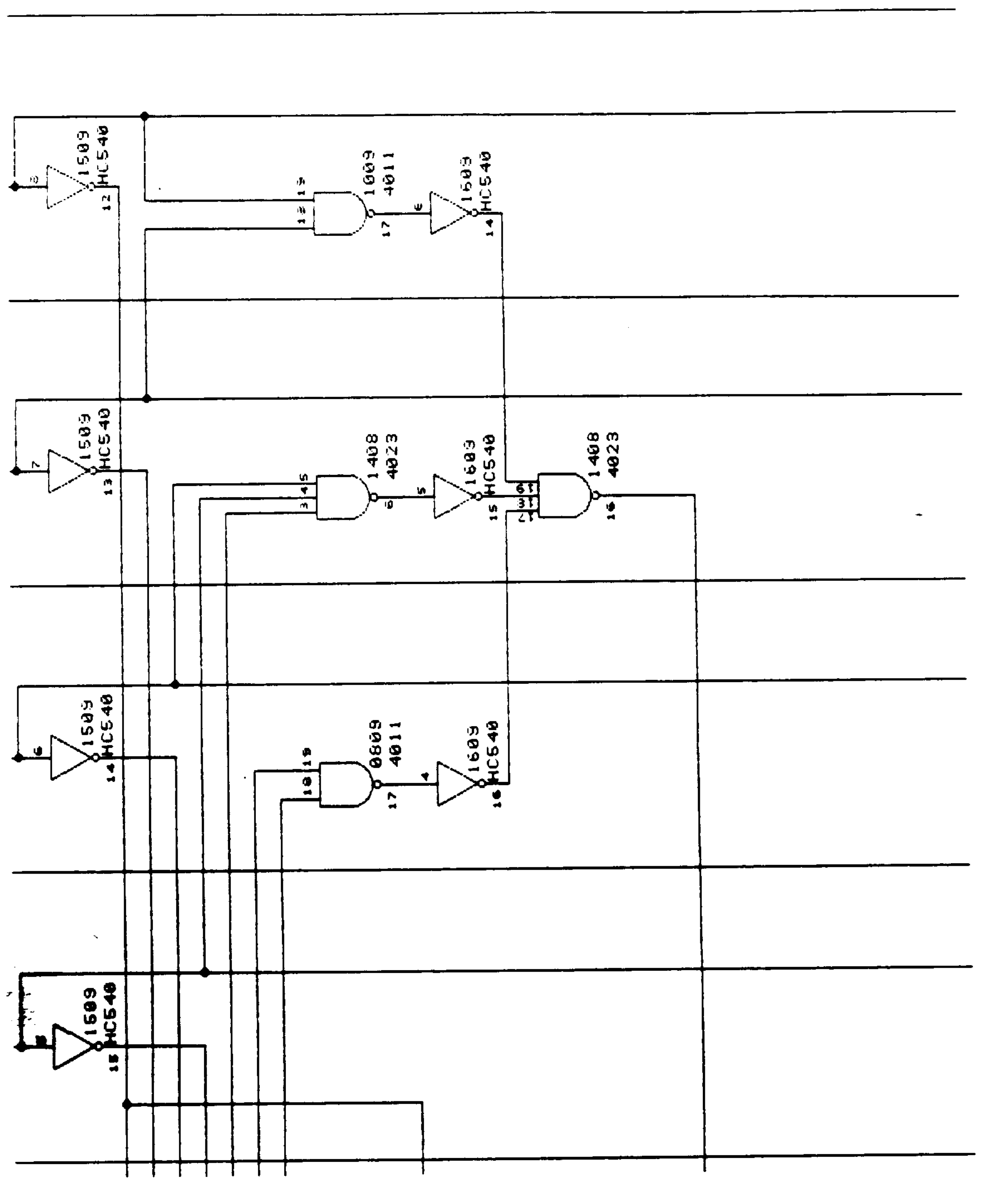

1509
HC540
1009
4011
1609
HC540
1509
HC540
1408
4023
1609
HC540
1408
4023
1509
HC540
0809
4011
1609
HC540
1609
HC540

Figure 48

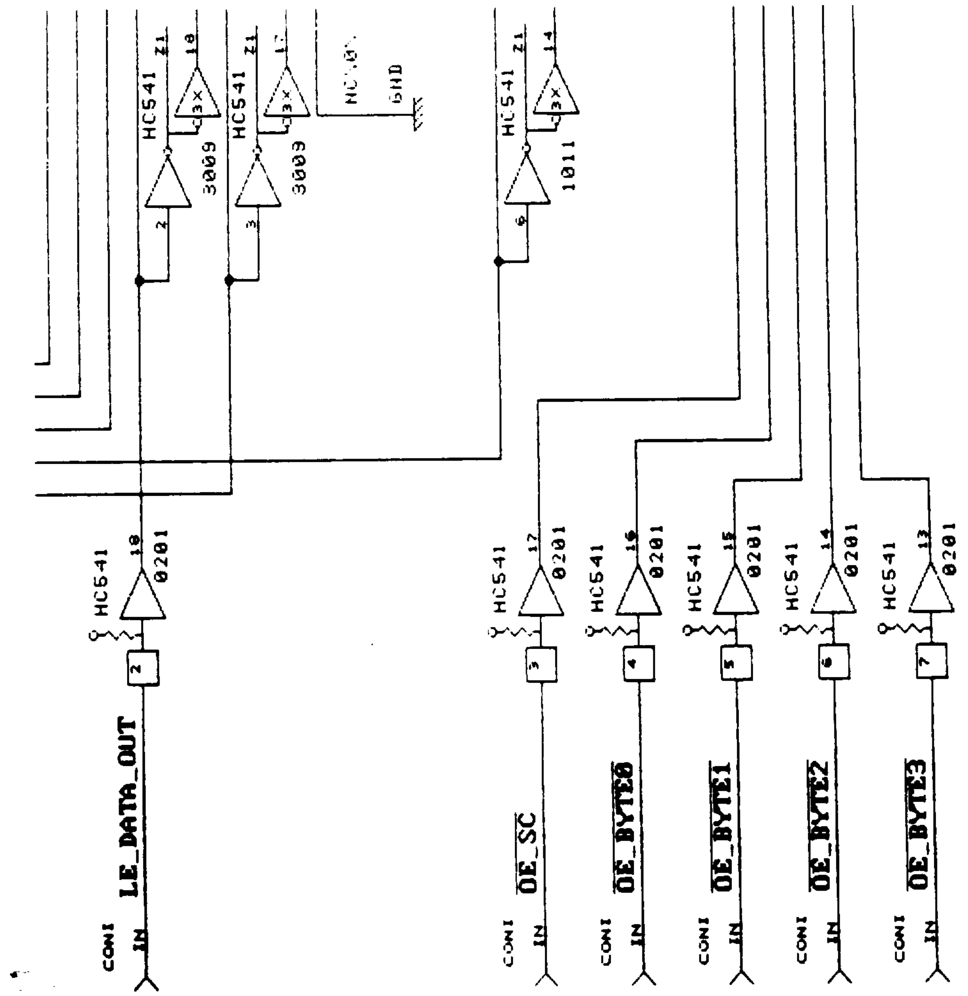

CON1
IN
LE_DATA_OUT
2
HC541
10
0201
HC541
Z1
18
3009
2
HC541
Z1
17
3009
3
NC/0
GND
HC541
Z1
14
1011
6
CON1
IN
OE_SC
3
HC541
17
0201
CON1
IN
OE_BYTE0
4
HC541
16
0201
CON1
IN
OE_BYTE1
5
HC541
15
0201
CON1
IN
OE_BYTE2
6
HC541
14
0201
CON1
IN
OE_BYTE3
7
HC541
13
0201
3x

Figure 49

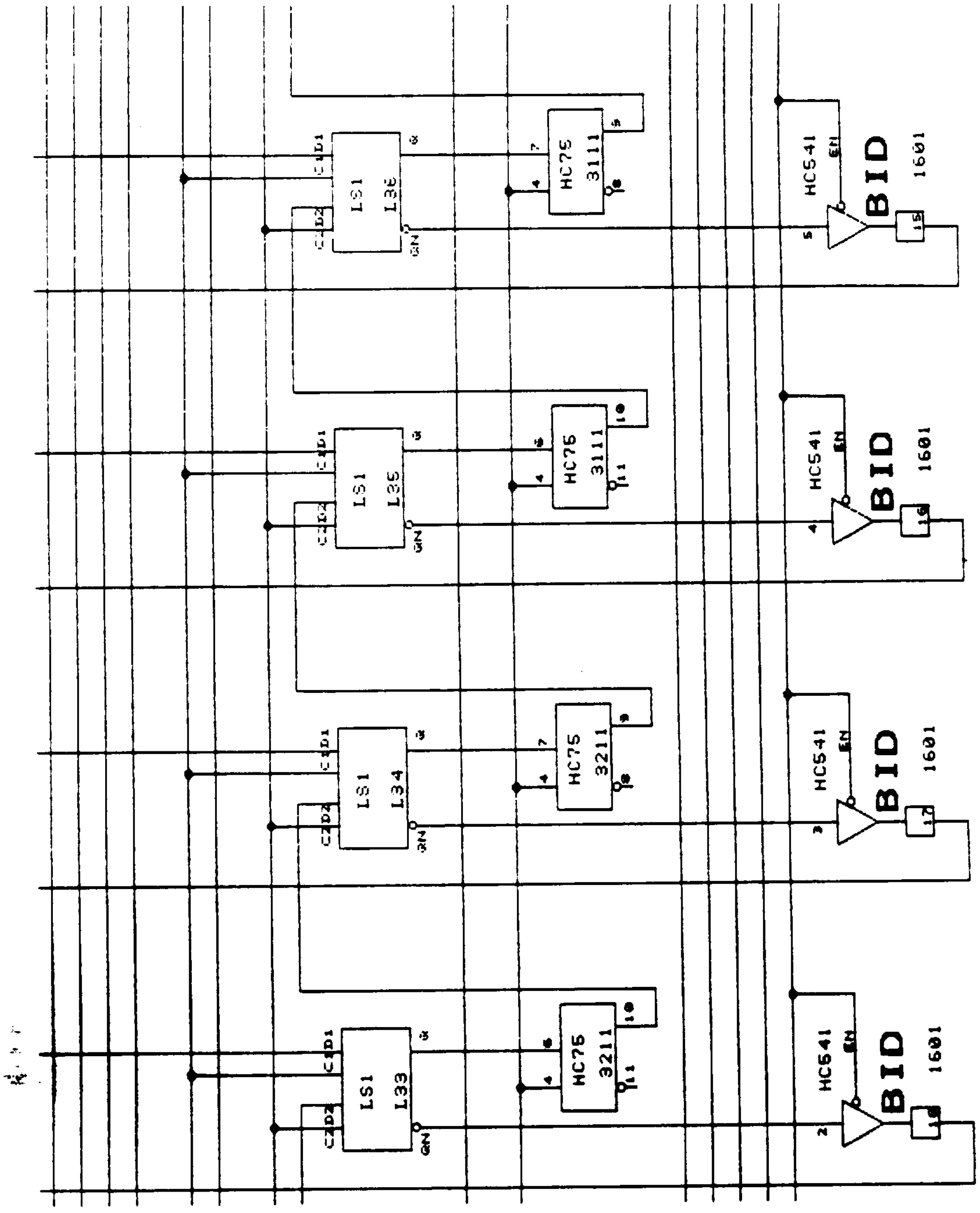

LS1
L33
L34
L35
L36
C1D1
C2D2
Q
QN
HC75
3211
3111
HC541
EN
BID
1601

Figure 50

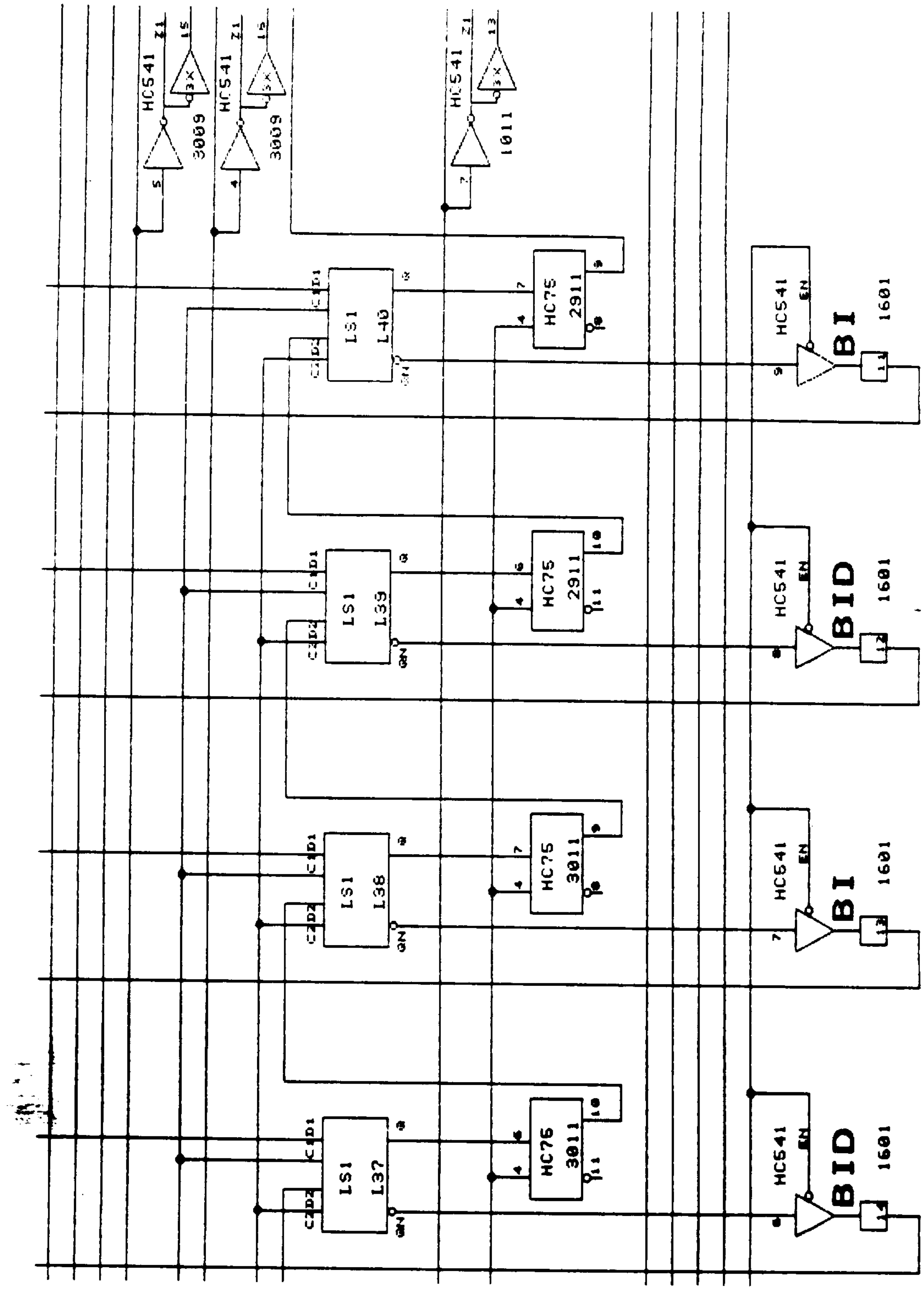

HC541
Z1
15
3X
3009
5
4
HC541
Z1
13
1011
7
LS1
L40
C1D1
C2D2
GN
Q
HC75
2911
7
4
9
8
HC541
EN
BI
1601
9
11
LS1
L39
HC75
2911
BID
12
LS1
L38
HC75
3011
BI
13
LS1
L37
HC75
3011
BID
14
10
11

Figure 51

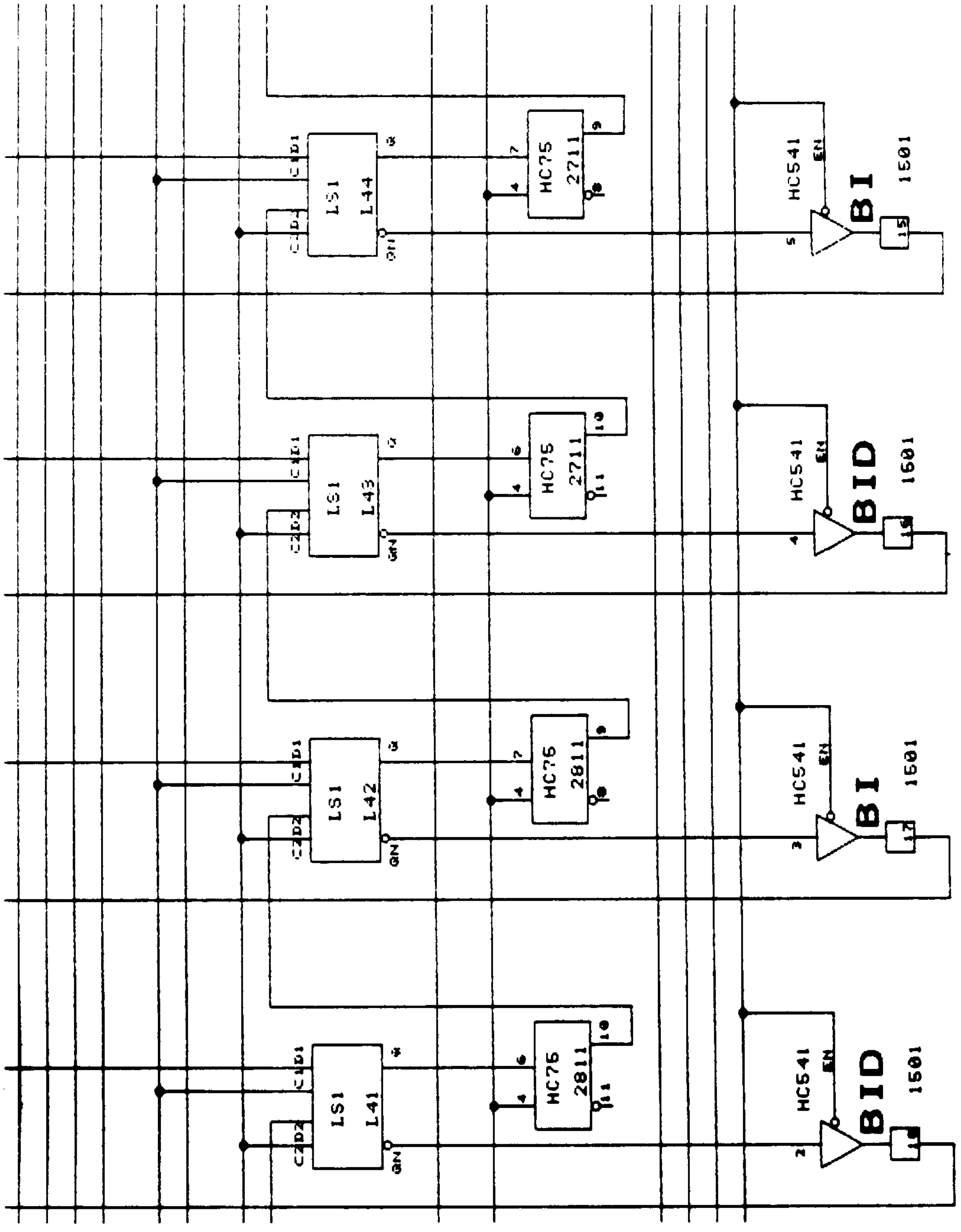
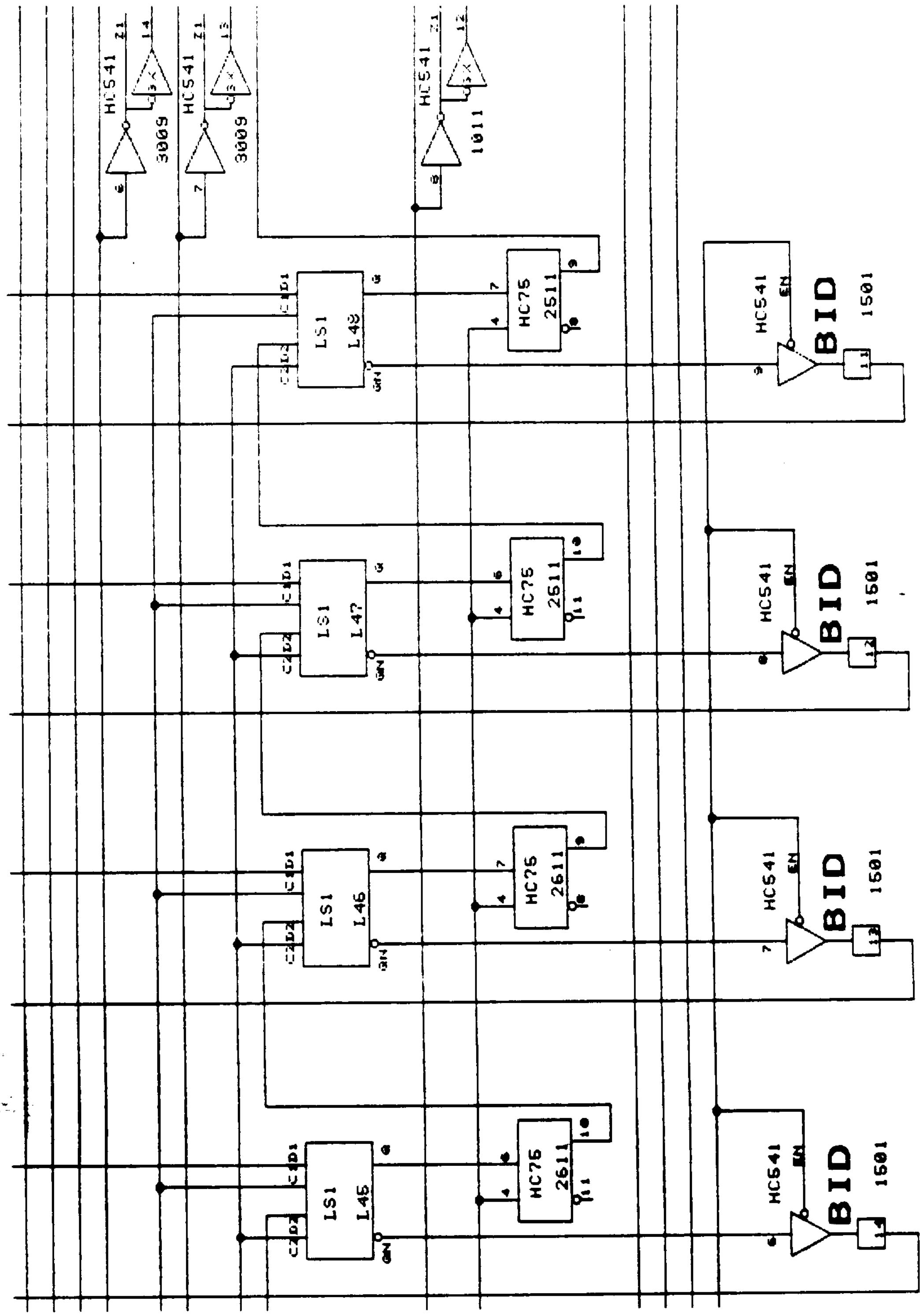
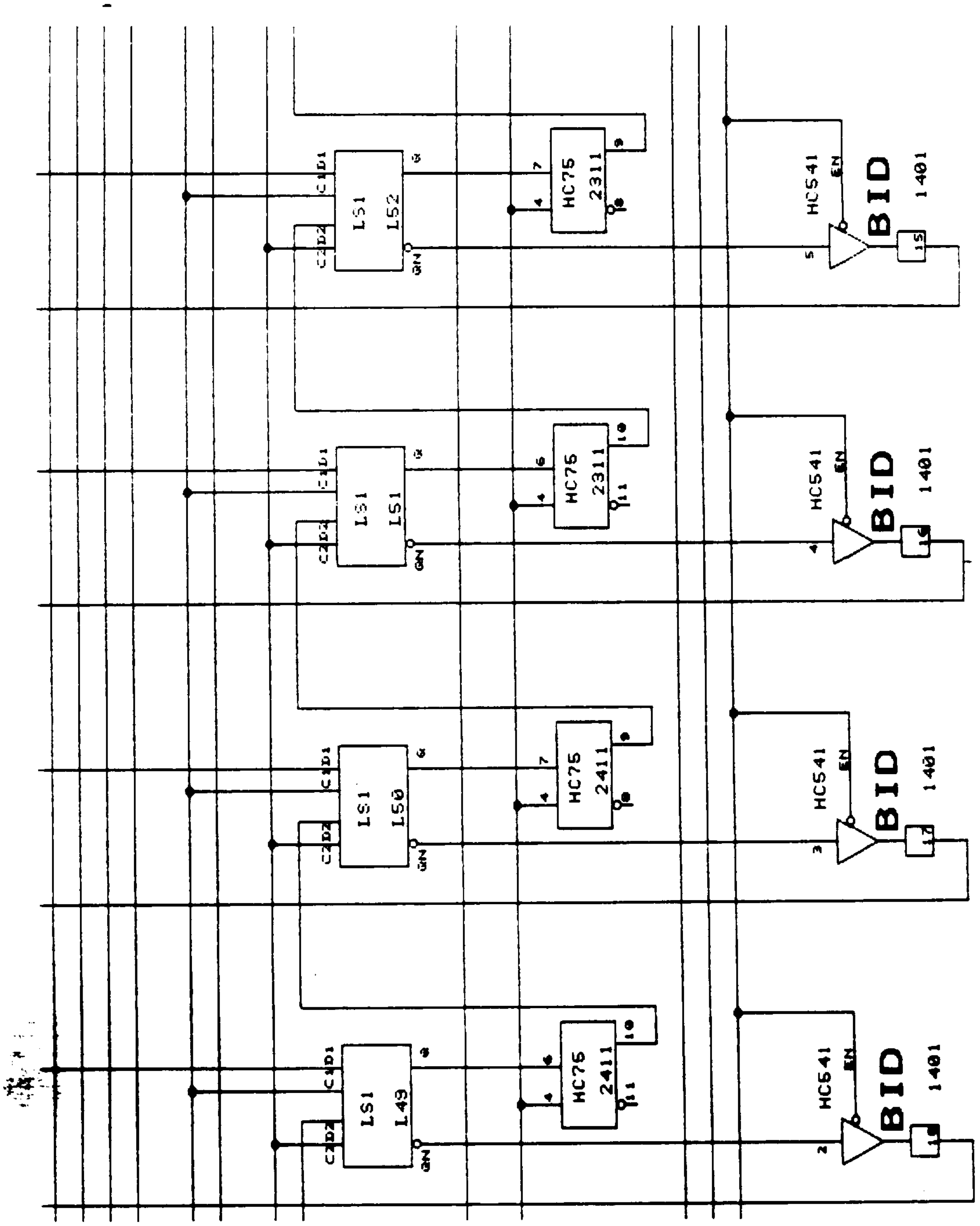

LS1
L41
L42
L43
L44
C1D1
C2D2
Q
QN
HC75
2811
2711
HC541
EN
BID
BI
1501
2
3
4
5
4
6
7
9
10
11
12
13
14
15

Figure 52



HC541
3009
HC541
3009
HC541
1011
LS1
L48
C1D1
C2D2
GN
HC75
2511
HC541
EN
BID
1501
LS1
L47
HC75
2511
BID
1501
LS1
L46
HC75
2611
BID
1501
LS1
L45
HC75
2611
BID
1501

Figure 53



LS1
L52
L51
L50
L49
C1D1
C2D2
Q
QN
HC75
2311
2311
2411
2411
HC541
EN
BID
1401

Figure 54

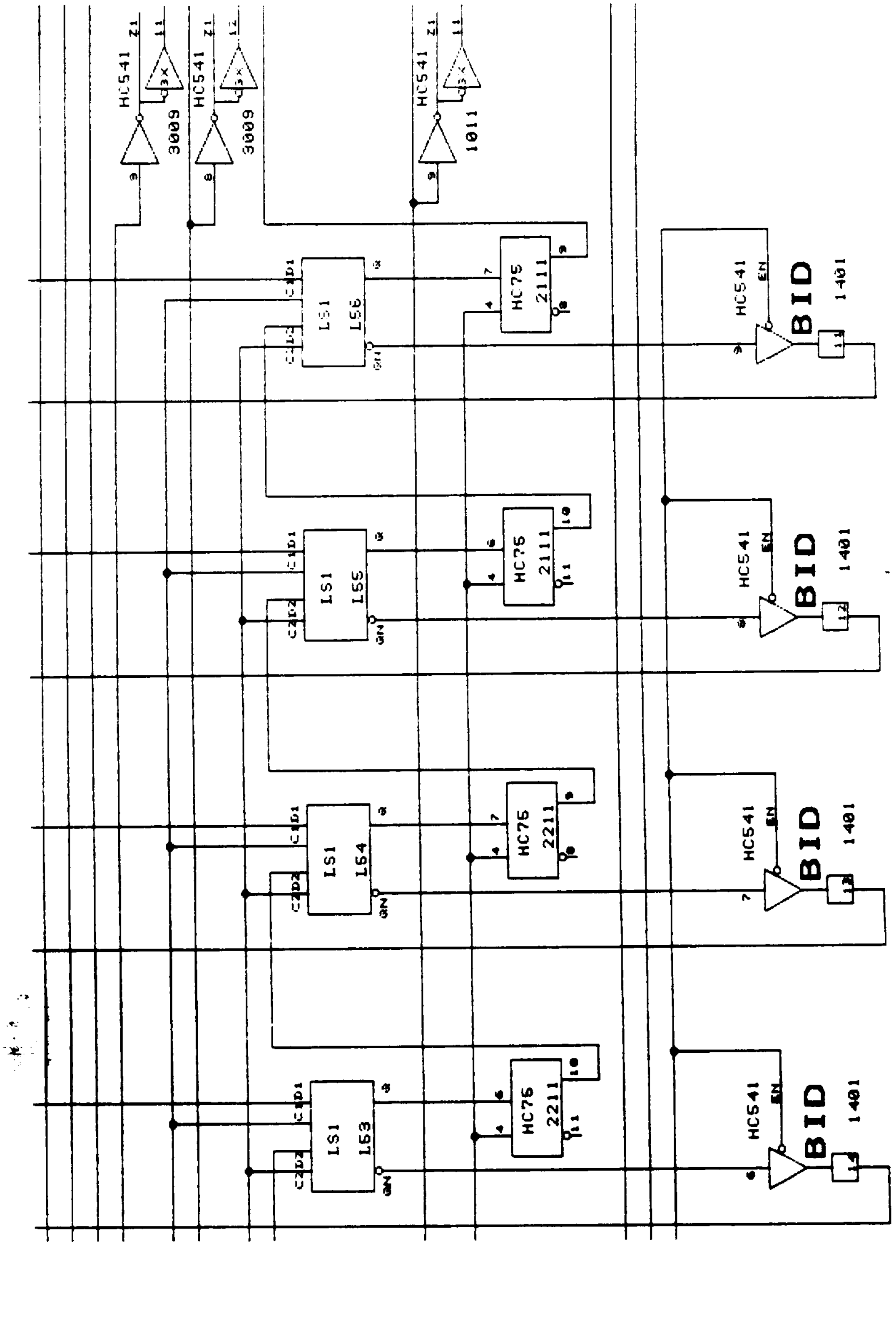

HC541
3009
Z1
I1
HC541
3009
Z1
I2
HC541
1011
Z1
I1
LS1
L56
C1D1
C2D2
Q
QN
HC75
2111
LS1
L55
C1D1
C2D2
Q
QN
HC75
2111
LS1
L54
C1D1
C2D2
Q
QN
HC75
2211
LS1
L53
C1D1
C2D2
Q
QN
HC75
2211
HC541
EN
BID
1401
HC541
EN
BID
1401
HC541
EN
BID
1401
HC541
EN
BID
1401

Figure 55

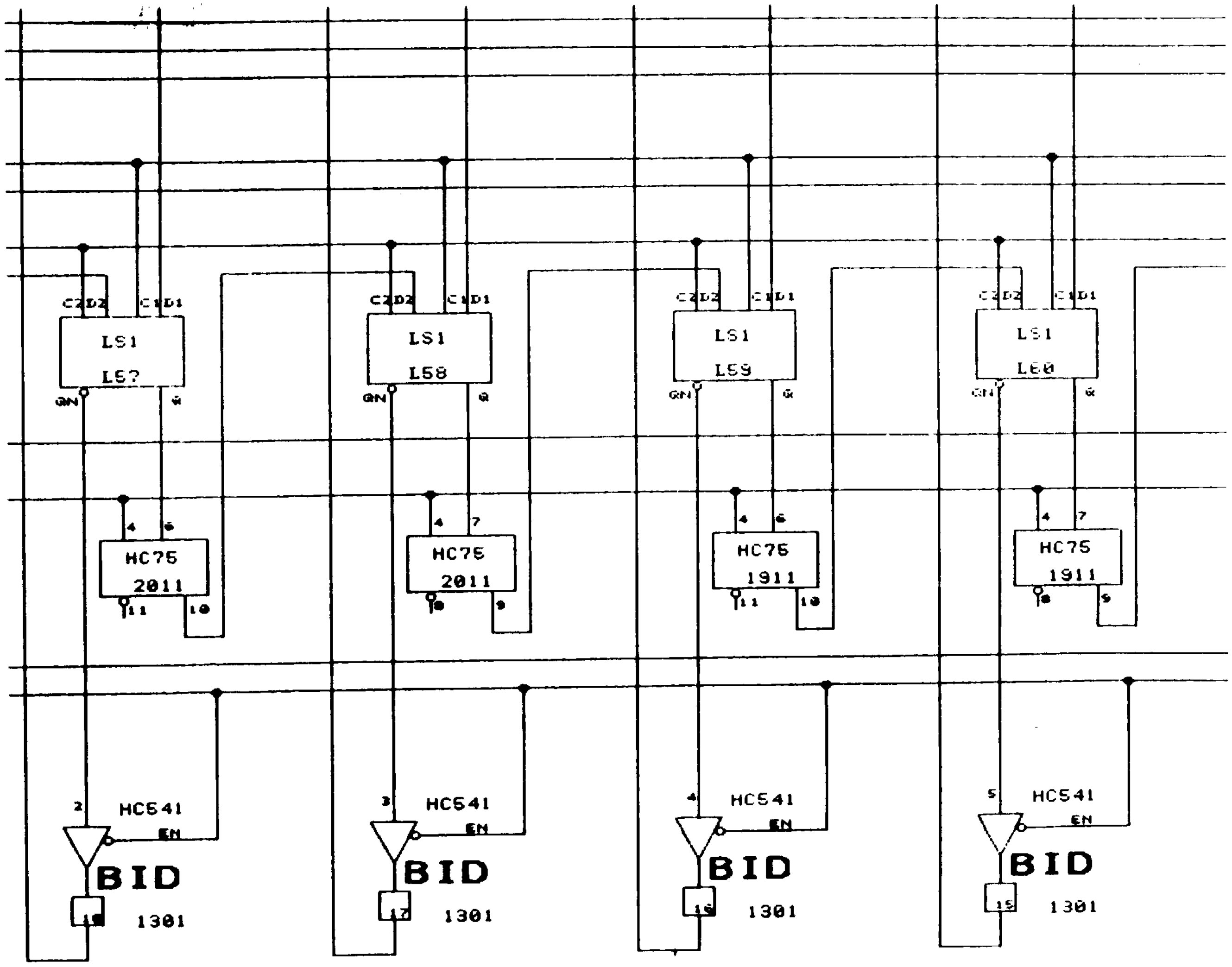
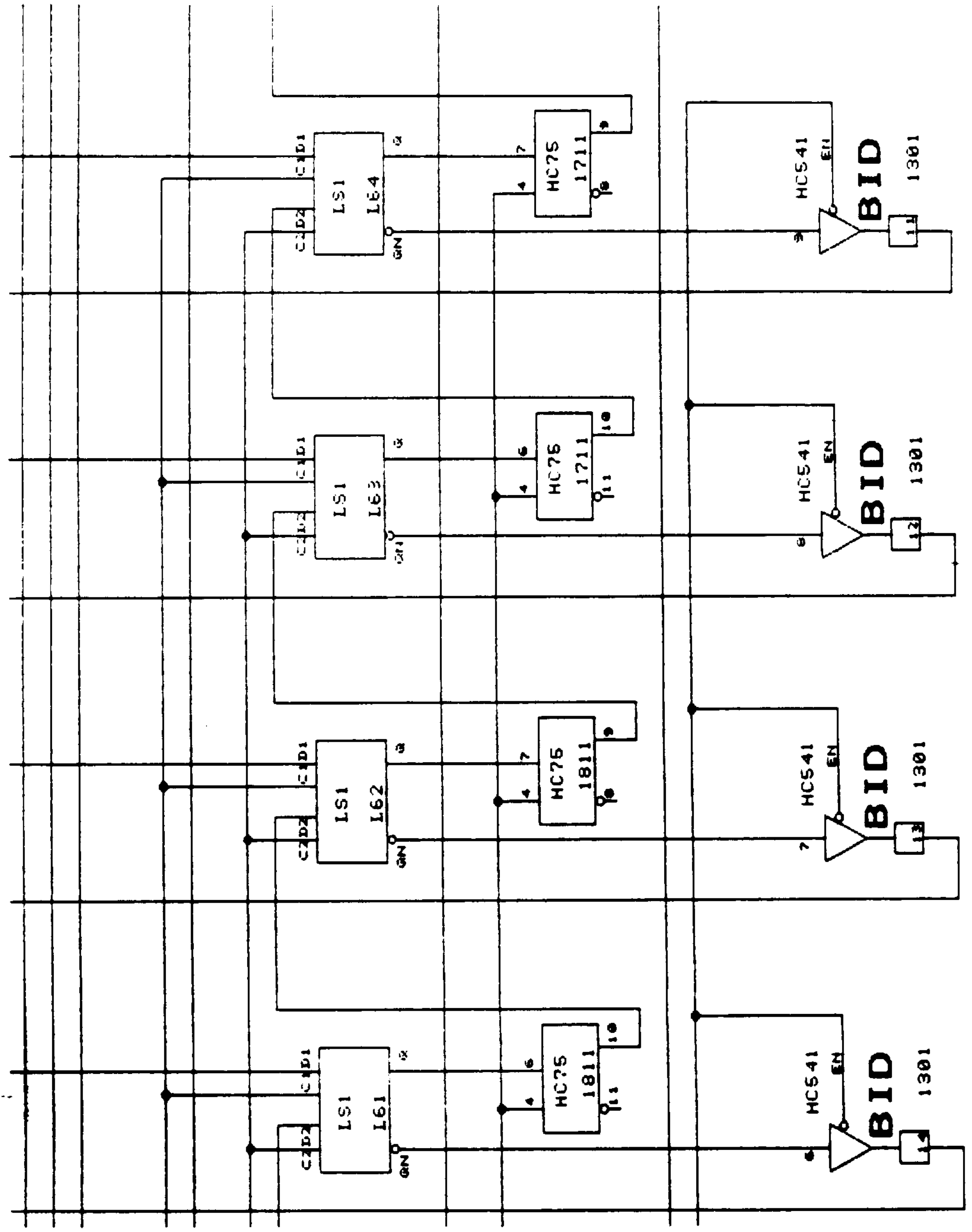

C2D2
C1D1
LS1
LS?
GN
Q
4
6
HC75
2011
11
10
2
HC541
EN
BID
1301
C2D2
C1D1
LS1
L58
GN
Q
4
7
HC75
2011
9
3
HC541
EN
BID
17
1301
C2D2
C1D1
LS1
L59
GN
Q
4
6
HC75
1911
11
10
4
HC541
EN
BID
1301
C2D2
C1D1
LS1
L60
GN
Q
4
7
HC75
1911
9
5
HC541
EN
BID
15
1301

Figure 56



LS1
L61
L62
L63
L64
C1D1
C2D2
Q
QN
HC75
1811
1711
HC541
EN
BID
1301

Figure 57

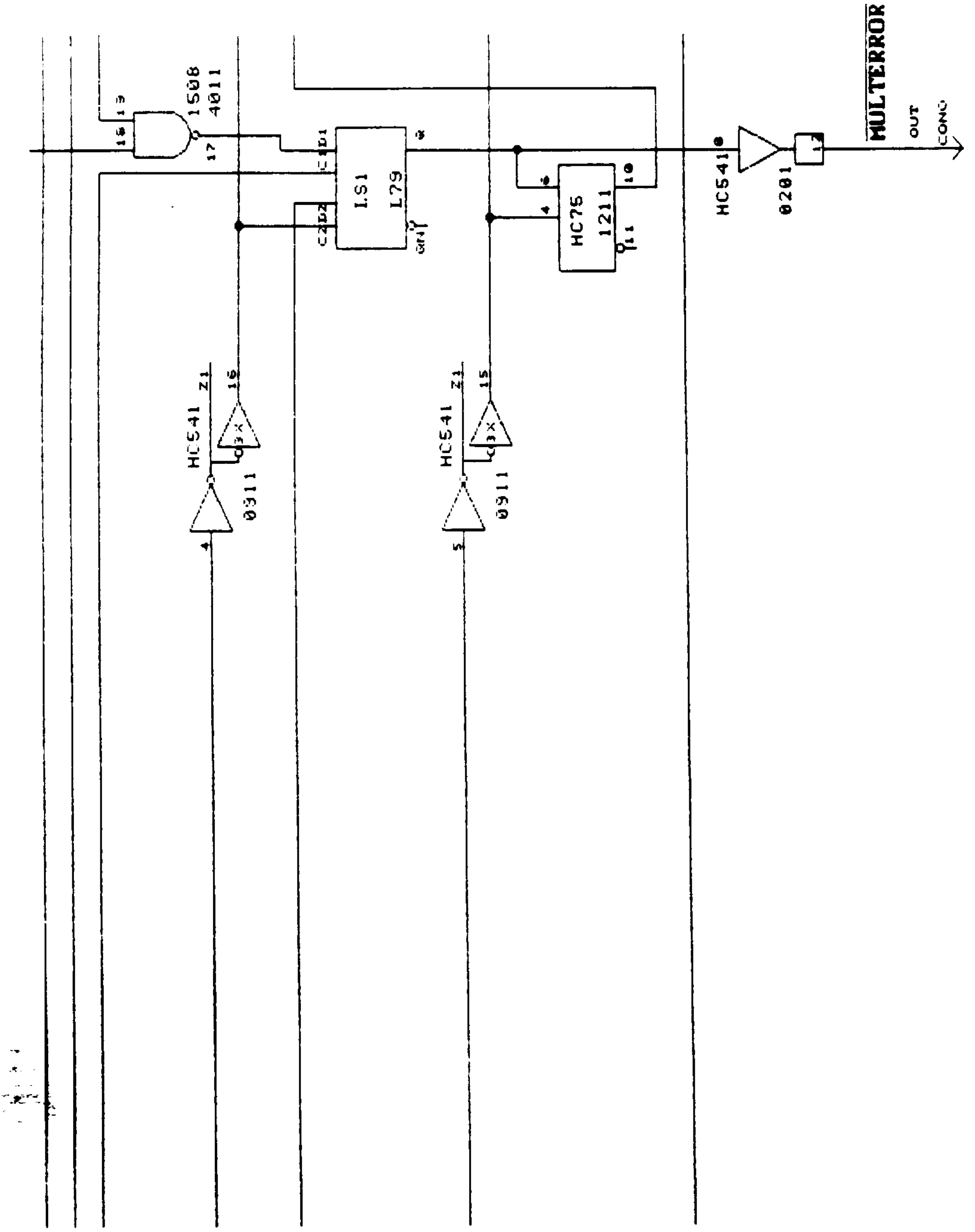

MULTERROR
OUT
CONO
1508
4011
18
19
17
C1D1
C2D2
LS1
L79
Q
HC75
1211
4
6
10
11
HC541
0201
HC541
Z1
16
0311
4
HC541
Z1
15
0911
5

Figure 58

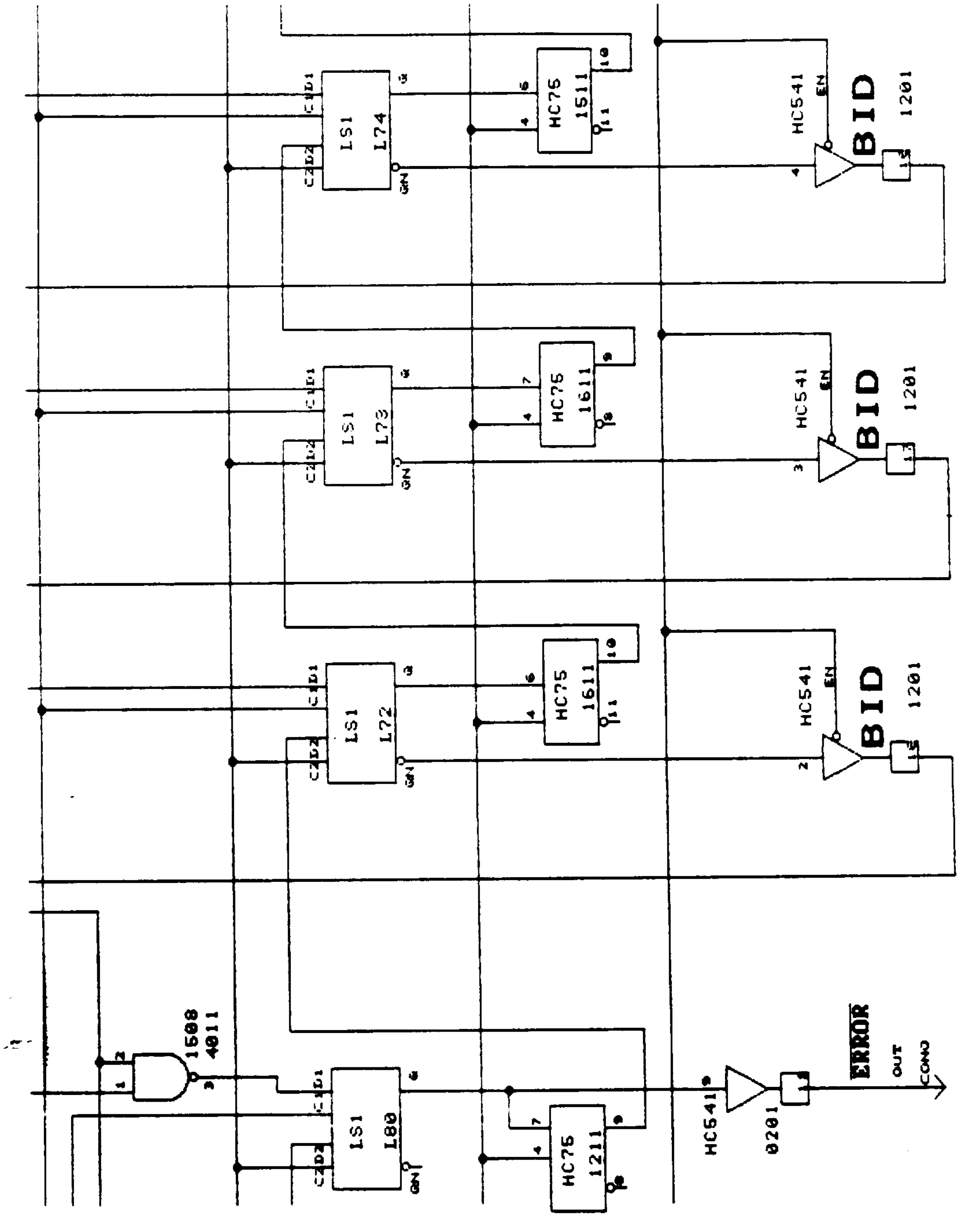
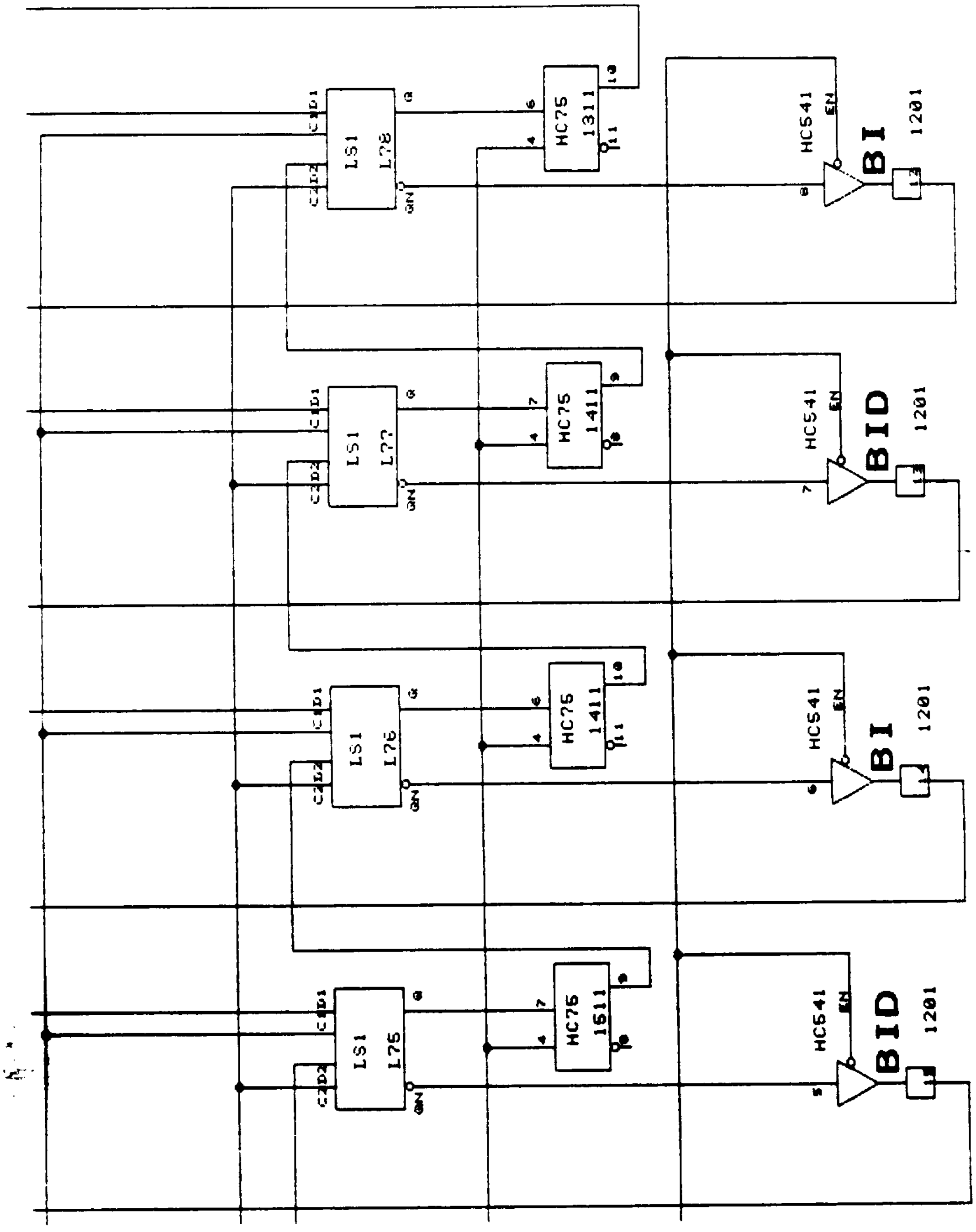

LS1
L74
HC75
1511
HC541
EN
BID
1201
LS1
L73
HC75
1611
HC541
EN
BID
1201
LS1
L72
HC75
1611
HC541
EN
BID
1201
1508
4011
LS1
L80
HC75
1211
HC541
0201
ERROR
OUT
COND

Figure 59



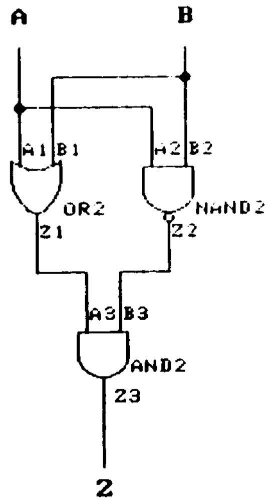
March 28 1989

EDC-BR DWG

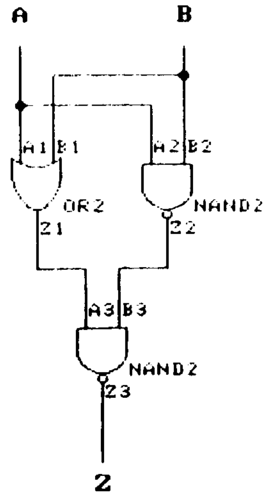
Figure 60

Realization of Exclusive-OR and Exclusive-NOR

Exclusive OR



Exclusive NOR



Position of the Exclusive-OR gates:

	OR: MS4071				NAND: MS4011				AND: MS4081			
Nr.	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	2907	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

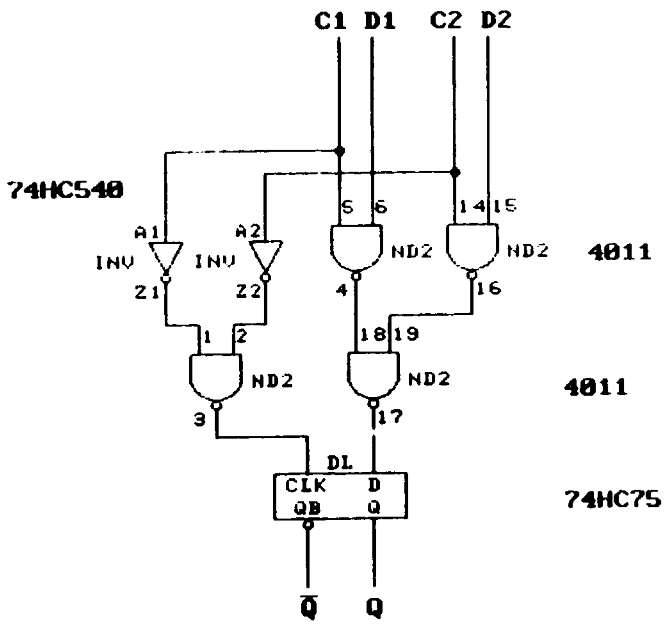
Nr.	loc.	A1	B1	Z1	loc.	A2	B2	Z2	loc.	A3	B3	Z3
37	2305	1	2	3	2306	1	2	3	2307	1	2	3
38	2305	5	6	4	2306	5	6	4	2307	5	6	4
39	2305	14	15	16	2306	14	15	16	2307	14	15	16
40	2305	18	19	17	2306	18	19	17	2307	18	19	17
41	2205	1	2	3	2206	1	2	3	2207	1	2	3
42	2205	5	6	4	2206	5	6	4	2207	5	6	4
43	2205	14	15	16	2206	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	1907	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

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}	$\overline{Q}_{n+1}$
0	0	Q_n	$\overline{Q}_n$
0	1	D_2	$\overline{D}_2$
1	0	D_1	$\overline{D}_1$
1	1	0	1

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

$$CLK = C_1 + C_2$$

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

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

CONNECTOR C1

Pin	Signal
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	

CONN 2B

CONNECTOR C2

Pin	Signal
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	

CONN 2C

CONNECTOR C3

Pin	Signal
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	

CONN 1A

CONNECTOR C4

Pin	Signal
1	LE DATA IN
2	LE R/$\overline{W}$
3	LE DATA OUT
4	LE CB/SY
5	LE ERROR
6	LE ME
7	FDI
8	FCLK
9	reserved
10	$\overline{OE}$ DATA
11	$\overline{OE}$ SC
12	CLK A
13	CLK B
14	CORRECT
15	R/$\overline{W}$
16	reserved
17	FE1
18	FE2

CONN 1B

NO EXT. CONNECTOR

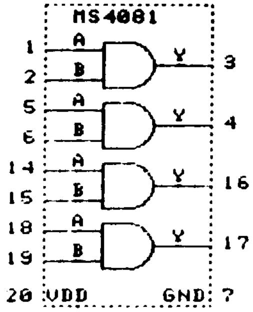
Pin	Signal
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	$\overline{OE}$ BYTE 0
11	$\overline{OE}$ BYTE 1
12	$\overline{OE}$ BYTE 2
13	$\overline{OE}$ BYTE 3
14	$\overline{OE}$ SC
15	
16	
17	
18	

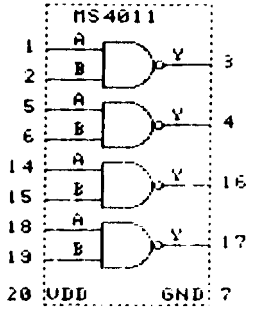
CONN 1C

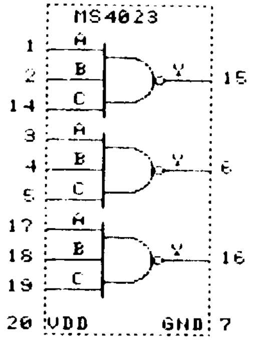
CONNECTOR C5

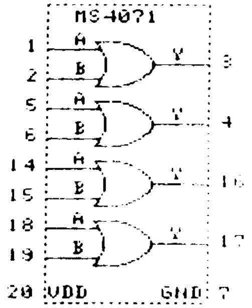
Pin	Signal
1	$\overline{\text{MULT ERROR}}$
2	$\overline{\text{ERROR}}$
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	

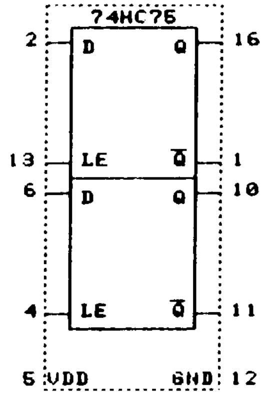
Pin Assignments

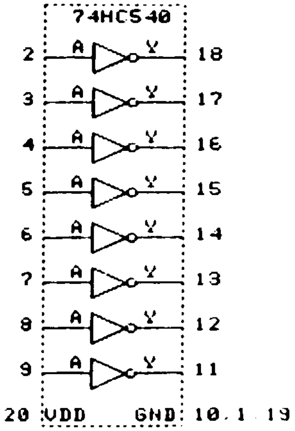


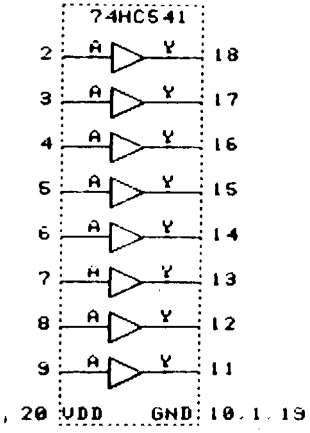


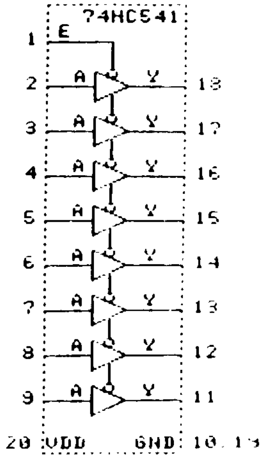




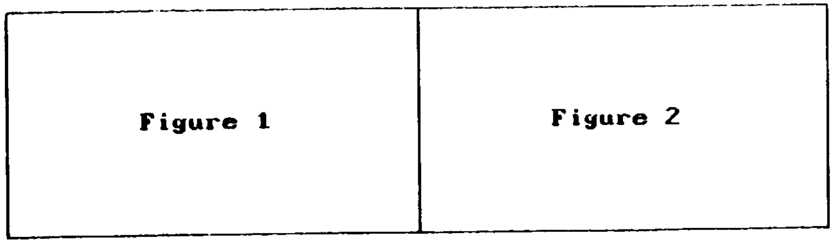








Map of IDT 49C460 Adapter



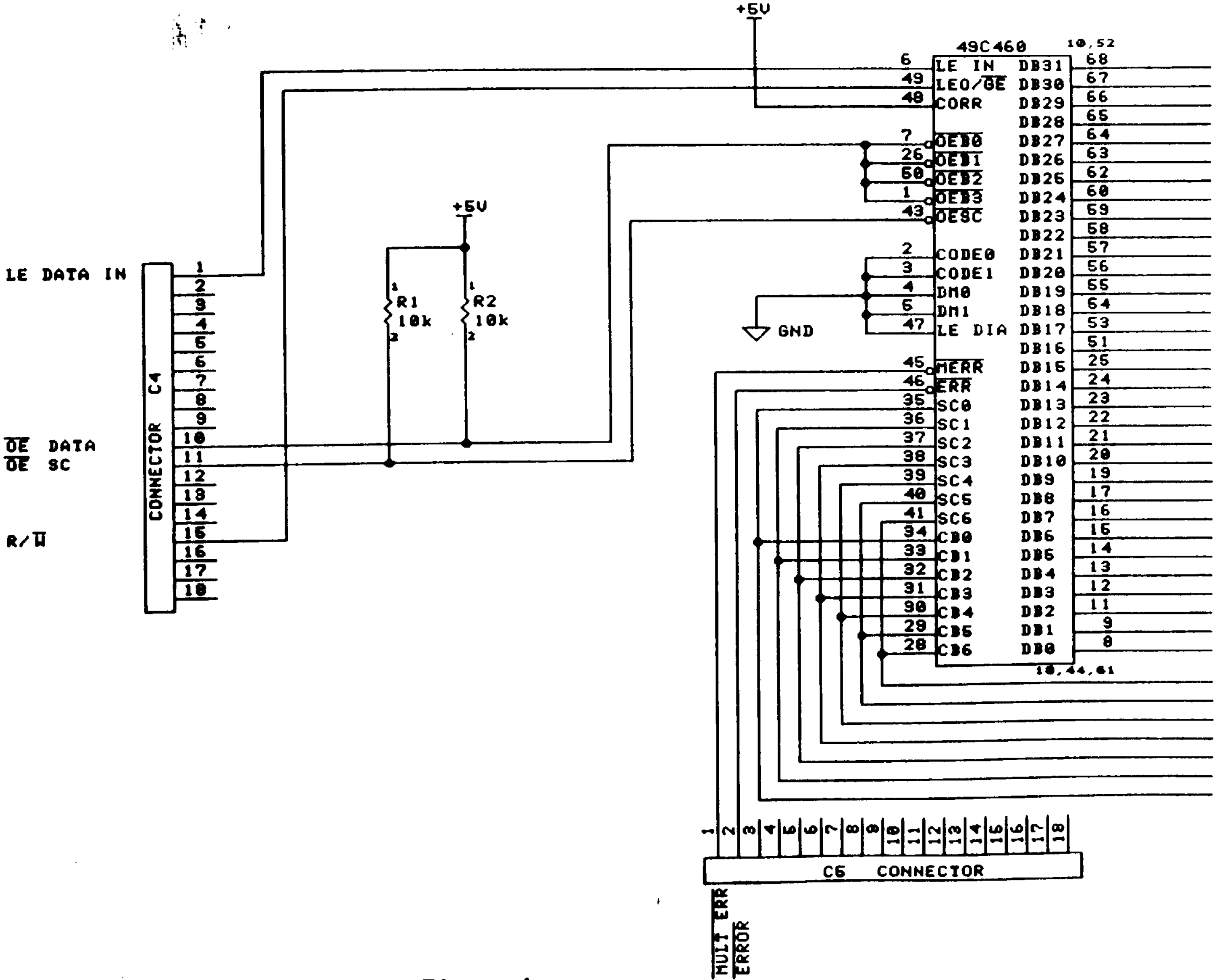

49C460
+5V
LE IN
LEO/$\overline{OE}$
CORR
$\overline{OEB0}$
$\overline{OEB1}$
$\overline{OEB2}$
$\overline{OEB3}$
$\overline{OESC}$
CODE0
CODE1
DM0
DM1
LE DIA
$\overline{MERR}$
$\overline{ERR}$
SC0
SC1
SC2
SC3
SC4
SC5
SC6
CB0
CB1
CB2
CB3
CB4
CB5
CB6
DB31
DB30
DB29
DB28
DB27
DB26
DB25
DB24
DB23
DB22
DB21
DB20
DB19
DB18
DB17
DB16
DB15
DB14
DB13
DB12
DB11
DB10
DB9
DB8
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0
10,52
10,44,61
GND
R1
10k
R2
10k
+5V
LE DATA IN
$\overline{OE}$ DATA
$\overline{OE}$ SC
R/$\overline{W}$
CONNECTOR C4
C6 CONNECTOR
$\overline{MULT\ ERR}$
$\overline{ERROR}$

Figure 1

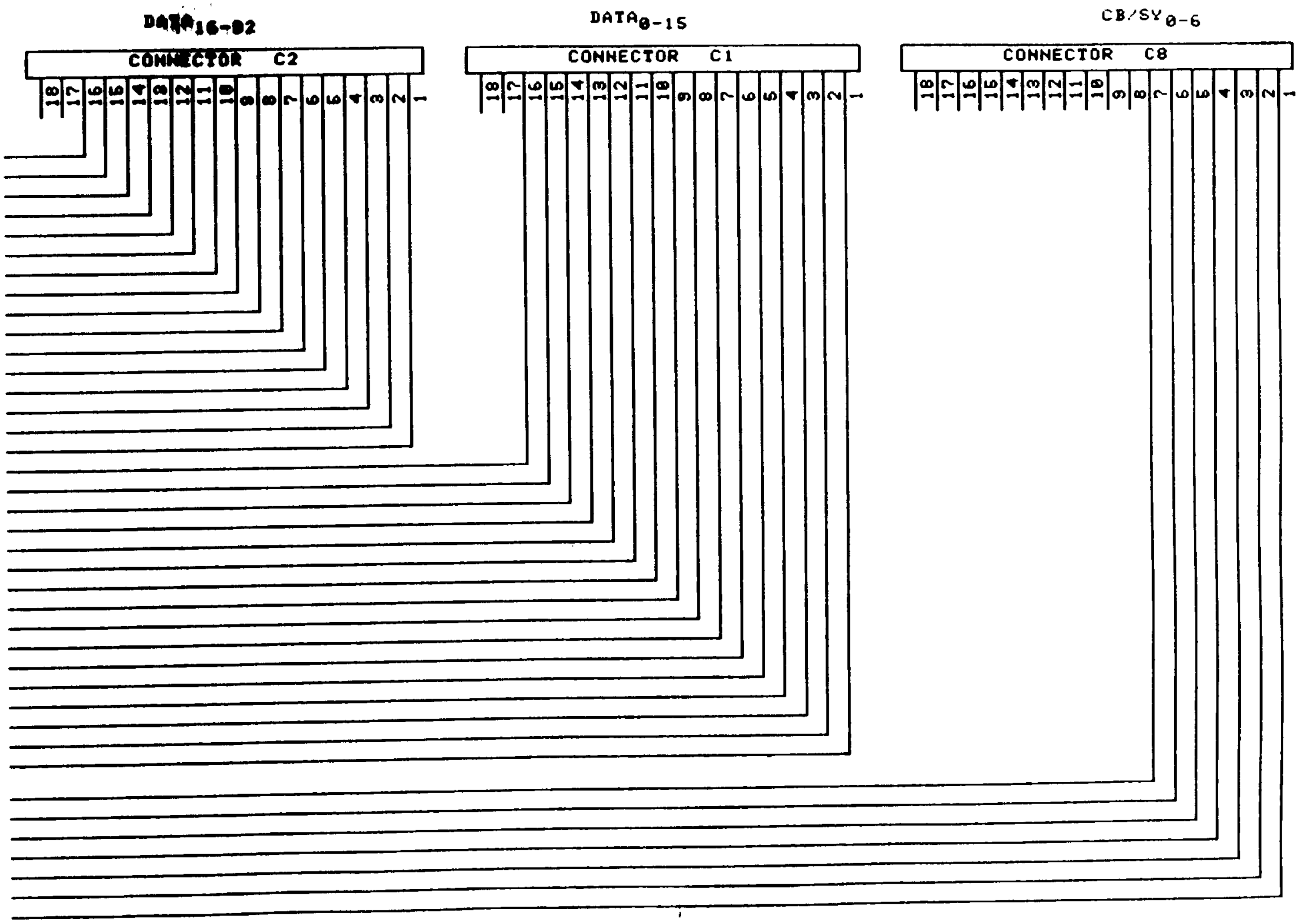
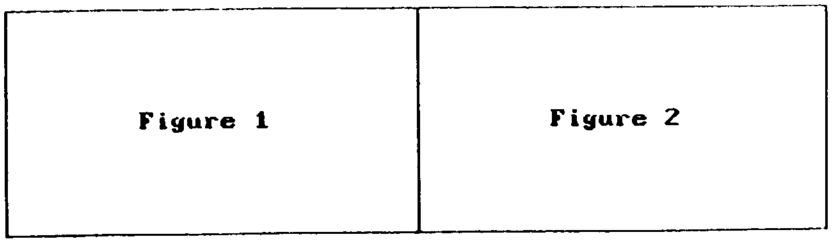

DATA16-32
CONNECTOR C2
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
DATA0-15
CONNECTOR C1
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
CB/SY0-6
CONNECTOR C8
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1

Figure 2

Map of TI 74ALS632 Adapter



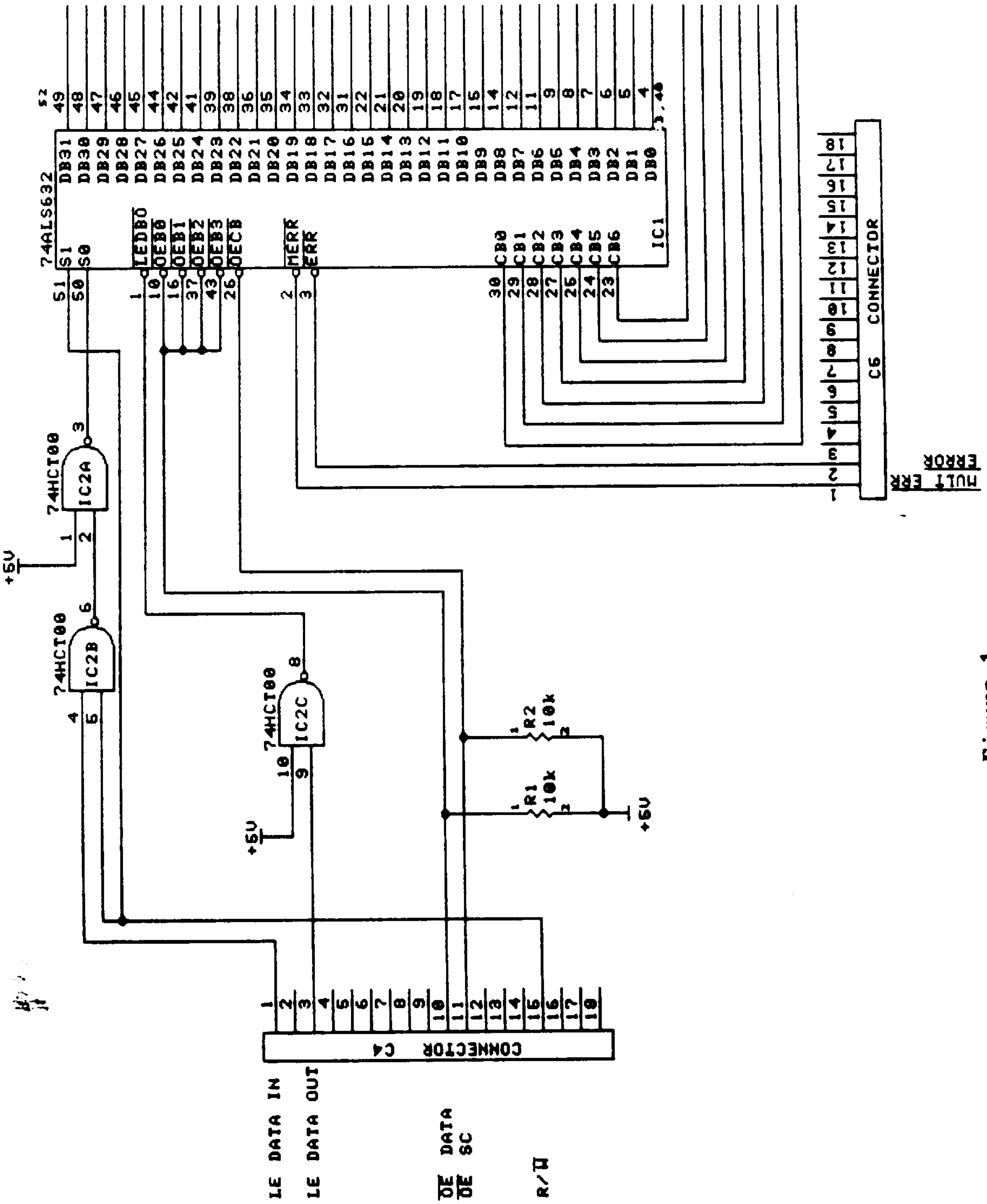

74ALS632
IC1
S1
S0
LEDBO
OEB0
OEB1
OEB2
OEB3
OECB
MERR
ERR
DB31
DB30
DB29
DB28
DB27
DB26
DB25
DB24
DB23
DB22
DB21
DB20
DB19
DB18
DB17
DB16
DB15
DB14
DB13
DB12
DB11
DB10
DB9
DB8
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0
CB0
CB1
CB2
CB3
CB4
CB5
CB6
74HCT00
IC2A
74HCT00
IC2B
74HCT00
IC2C
+5V
+5V
+5V
R1
10k
R2
10k
C6 CONNECTOR
MULT ERR
ERROR
CONNECTOR C4
LE DATA IN
LE DATA OUT
OE DATA
OE SC
R/W

Figure 1

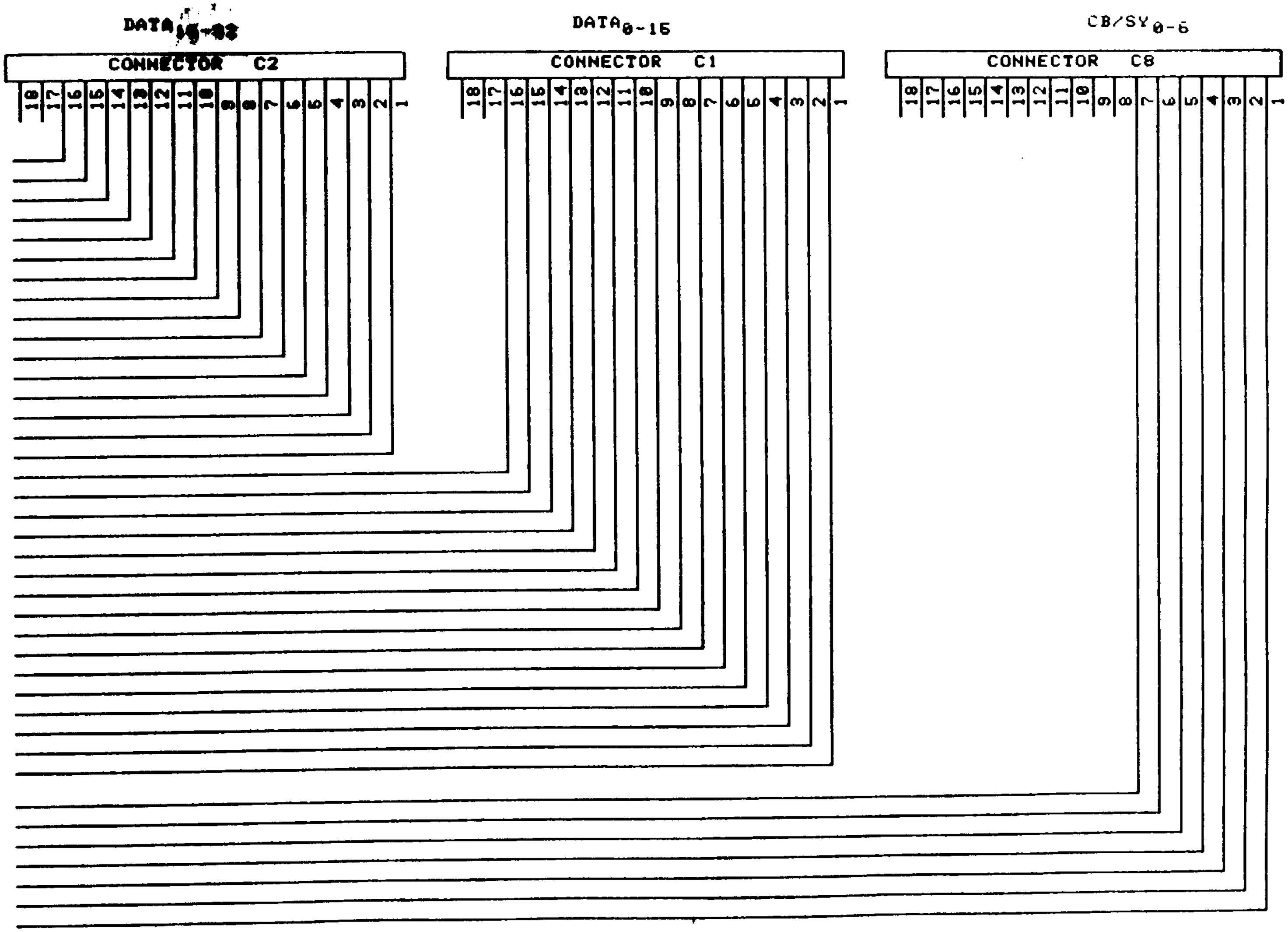
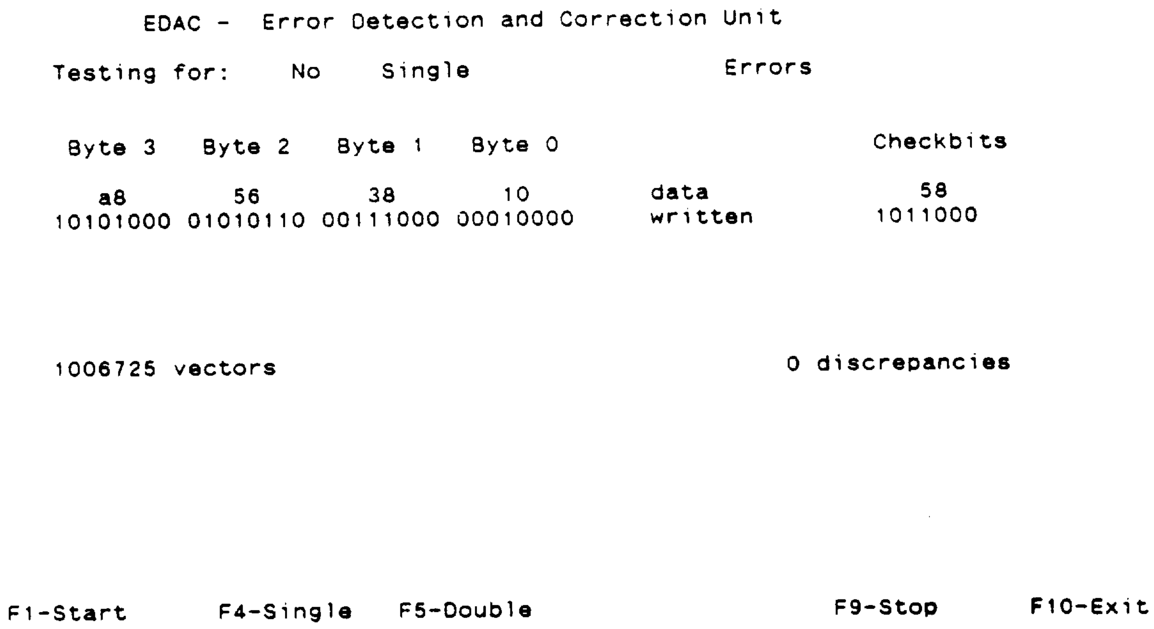


Figure 2

TI EDAC DWG

```
        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
```



```
              EDAC -  Error Detection and Correction Unit

     Testing for:     No    Single                      Errors


      Byte 3    Byte 2    Byte 1    Byte 0                              Checkbits

        a8        56        38        10        data                       58
     10101000 01010110 00111000 00010000        written                 1011000




     1006725 vectors                                           0 discrepancies





F1-Start       F4-Single   F5-Double                          F9-Stop       F10-Exit
```

APPENDIX D: FAULT SIMULATION

D.1 Fault Simulator Breadboard Layout

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

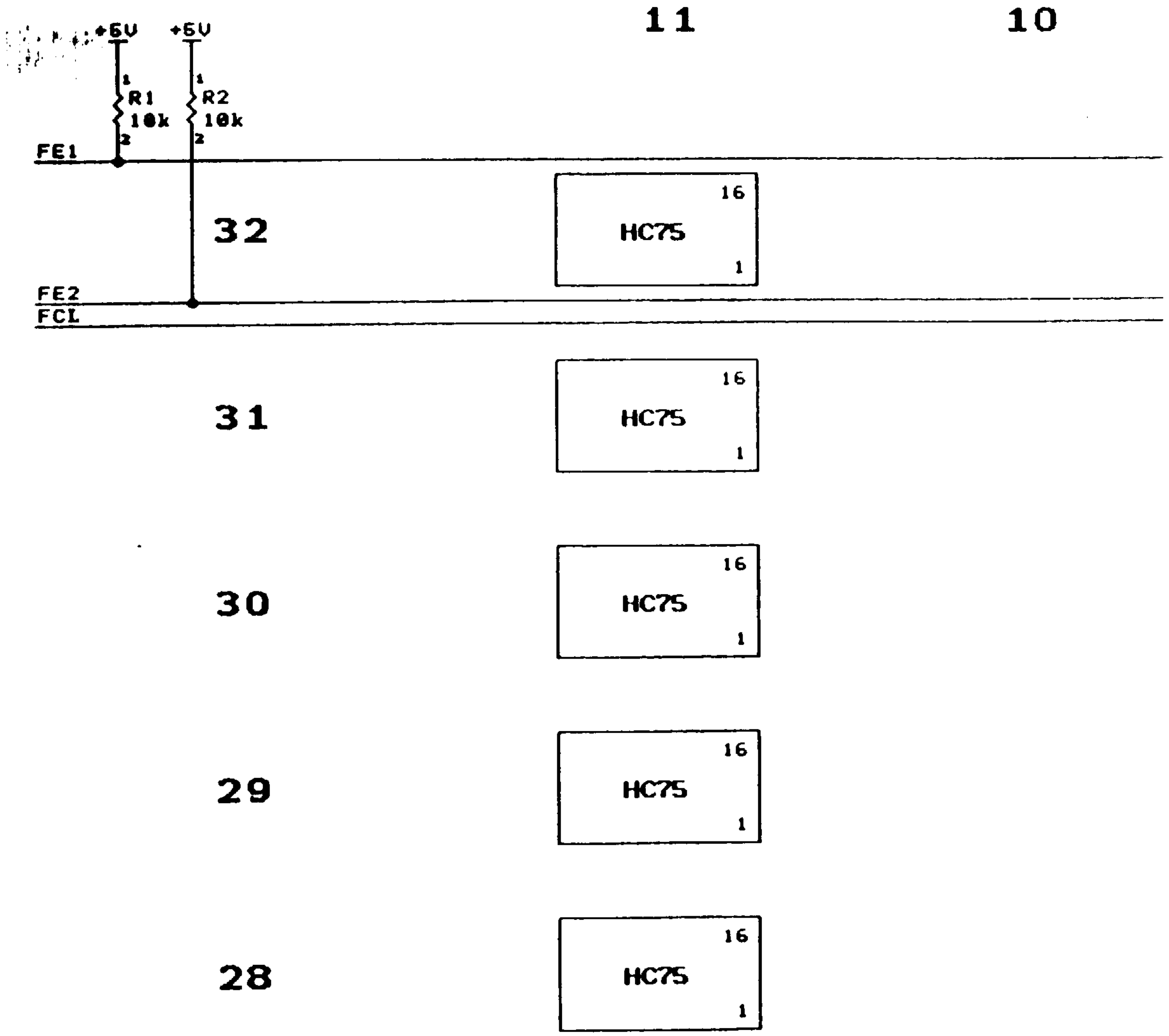

+5V
+5V
R1
10k
R2
10k
1
2
1
2
FE1
FE2
FCL
11
10
32
31
30
29
28
HC75
HC75
HC75
HC75
HC75
16
1

Figure 1

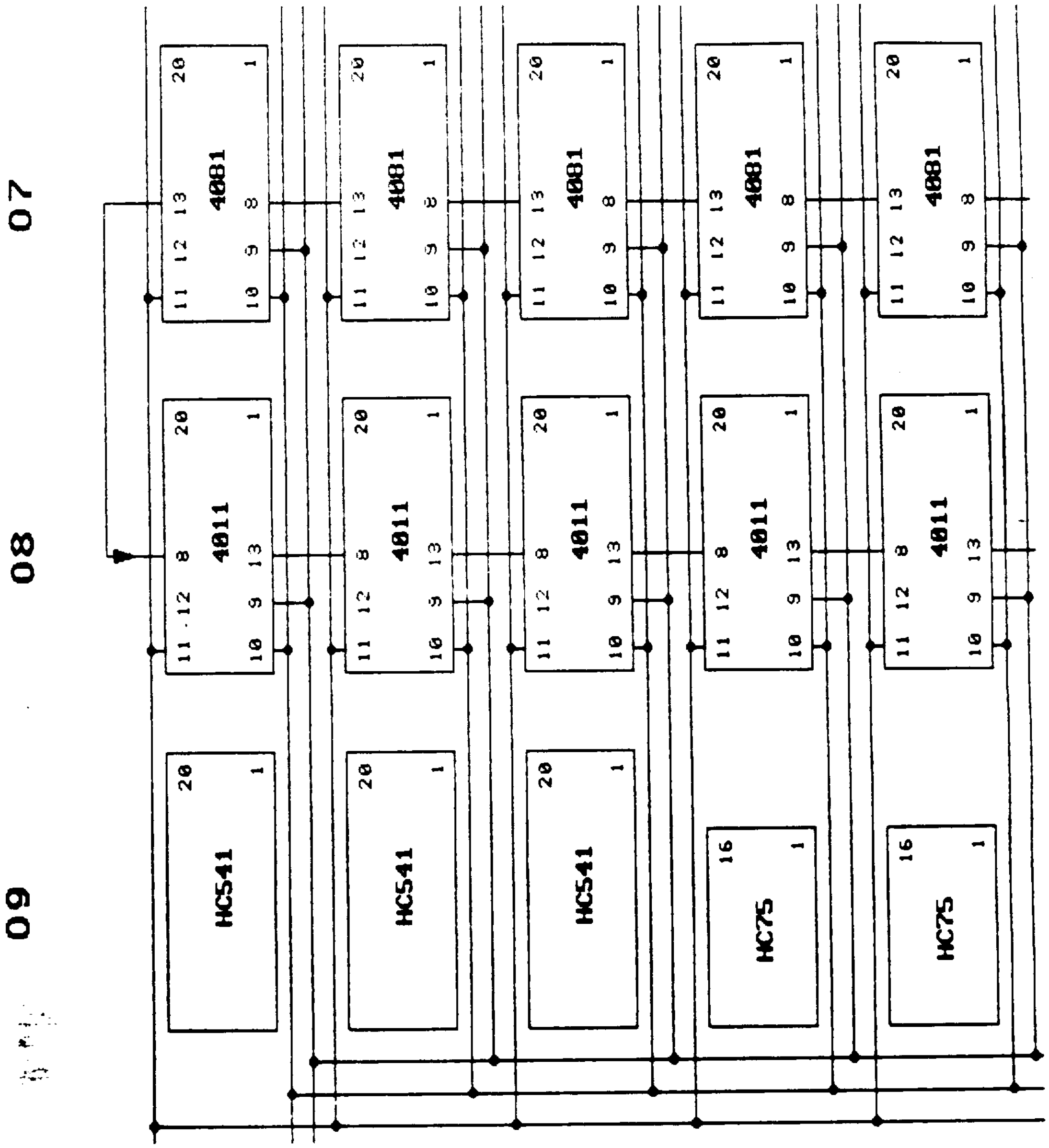


Figure 2

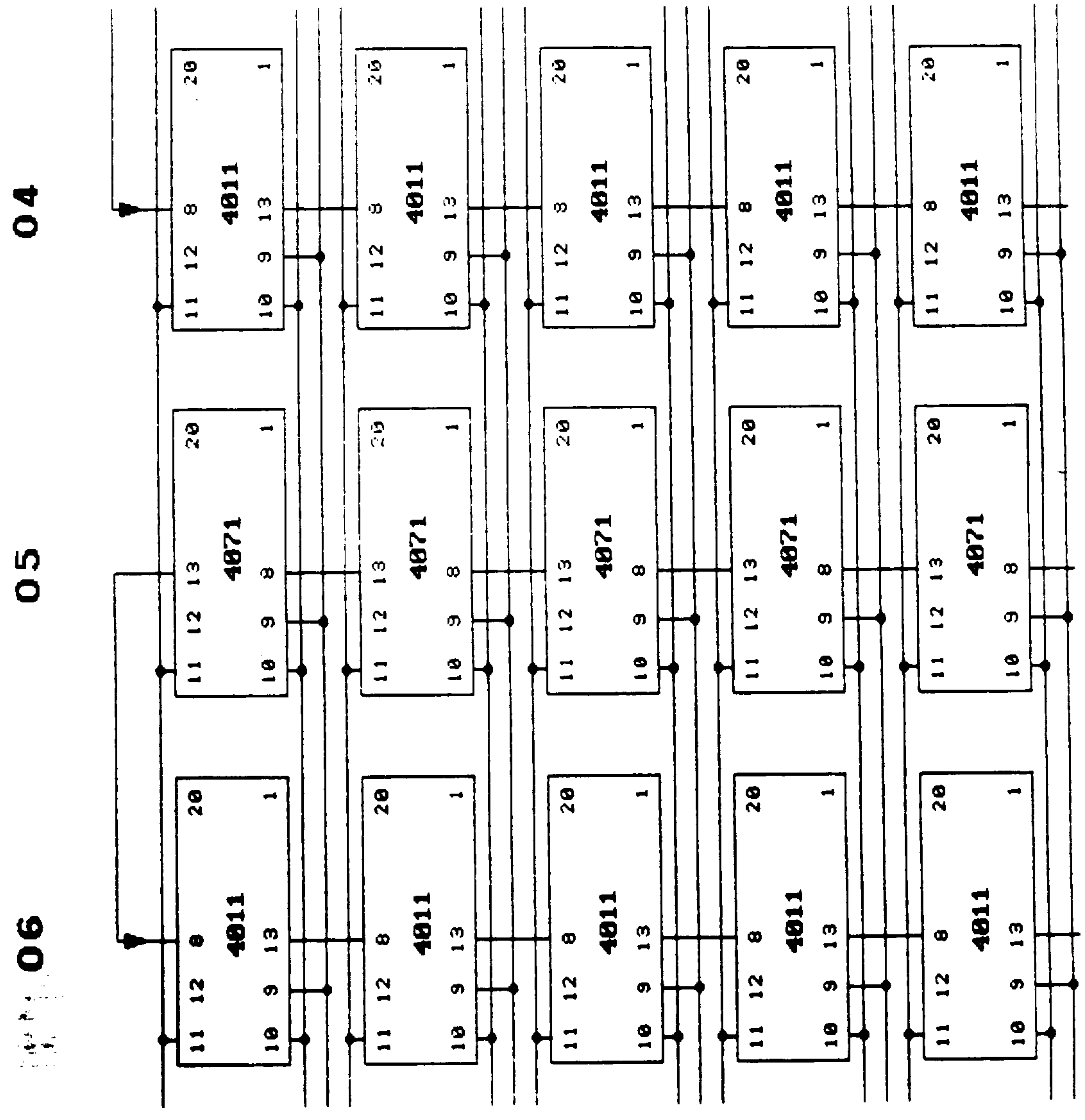


Figure 3

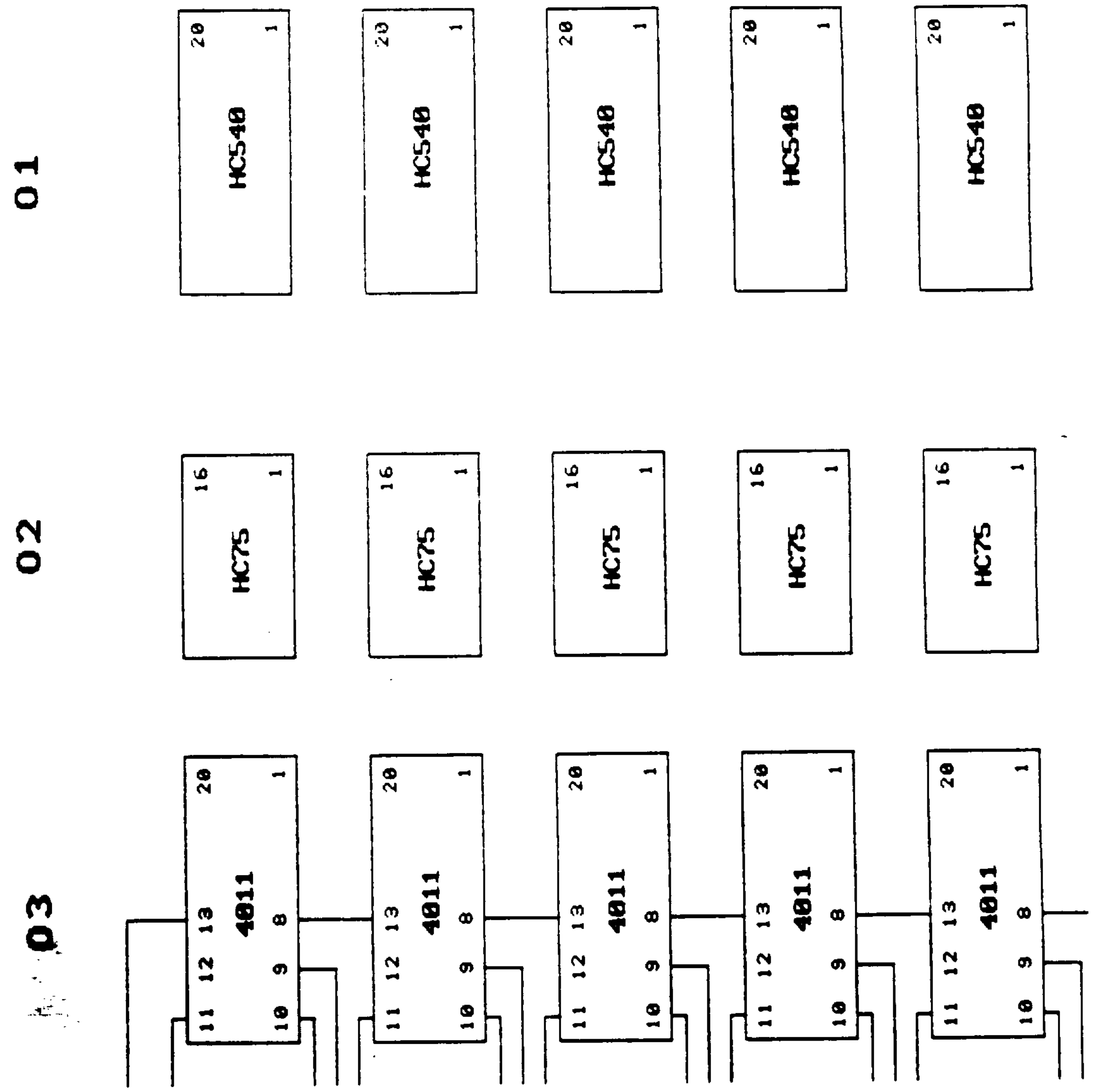

01
HC540
20
1
HC540
20
1
HC540
20
1
HC540
20
1
HC540
20
1
02
HC75
16
1
HC75
16
1
HC75
16
1
HC75
16
1
HC75
16
1
03
4011
20
1
13
12
11
8
9
10
4011
20
1
13
12
11
8
9
10
4011
20
1
13
12
11
8
9
10
4011
20
1
13
12
11
8
9
10
4011
20
1
13
12
11
8
9
10

Figure 4

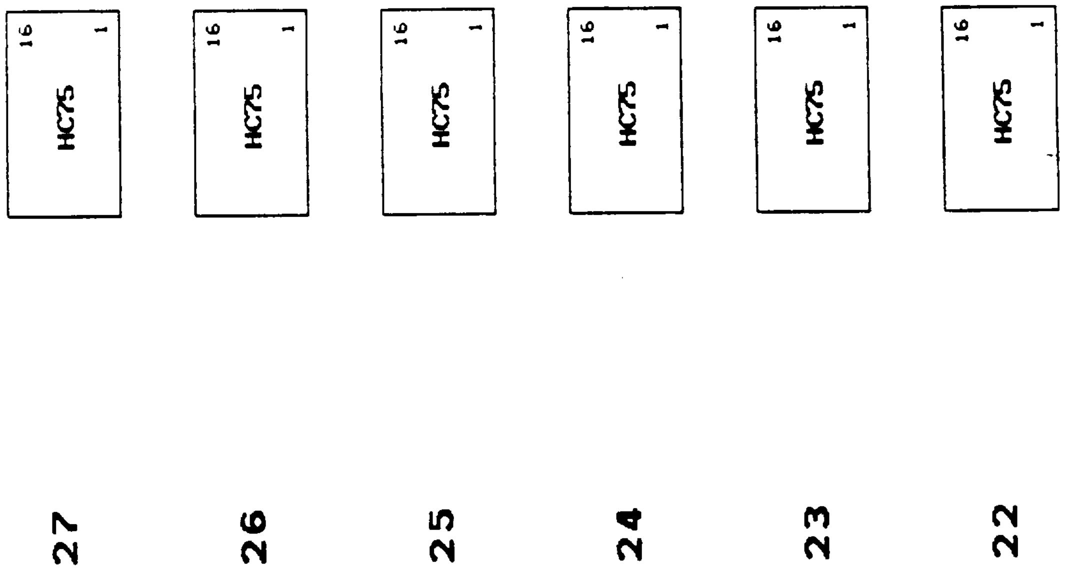


Figure 5

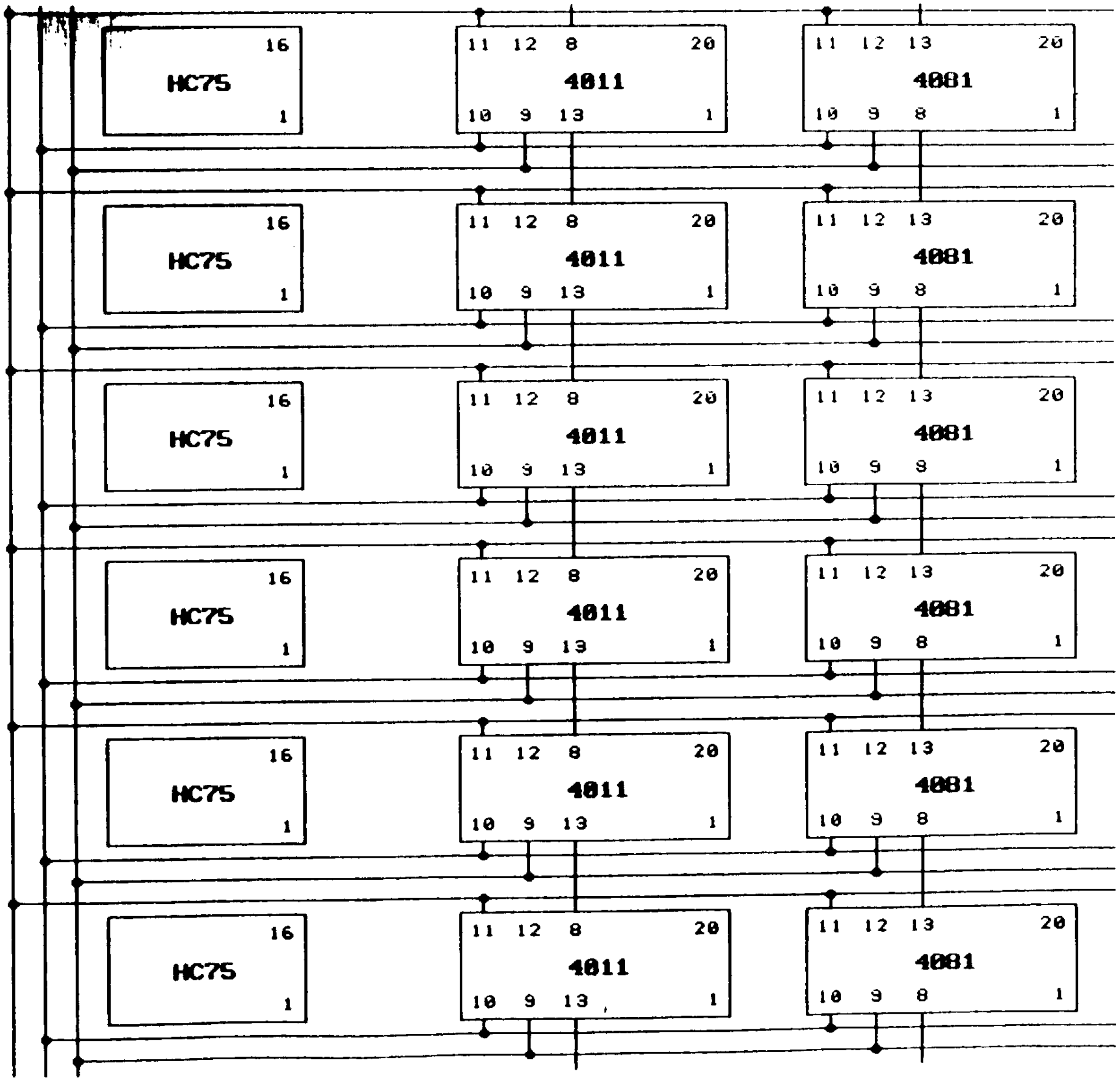


Figure 6

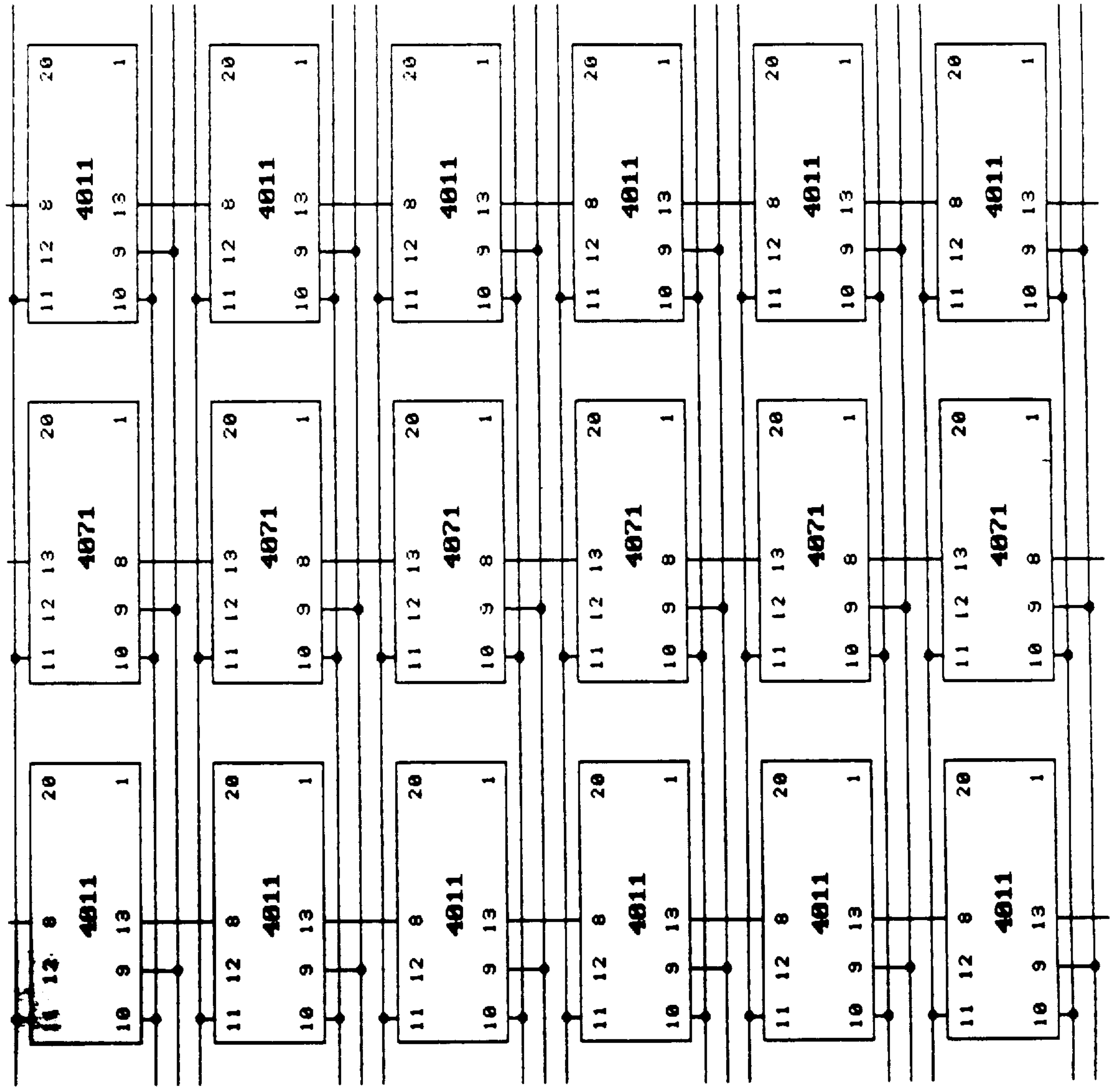


Figure 7

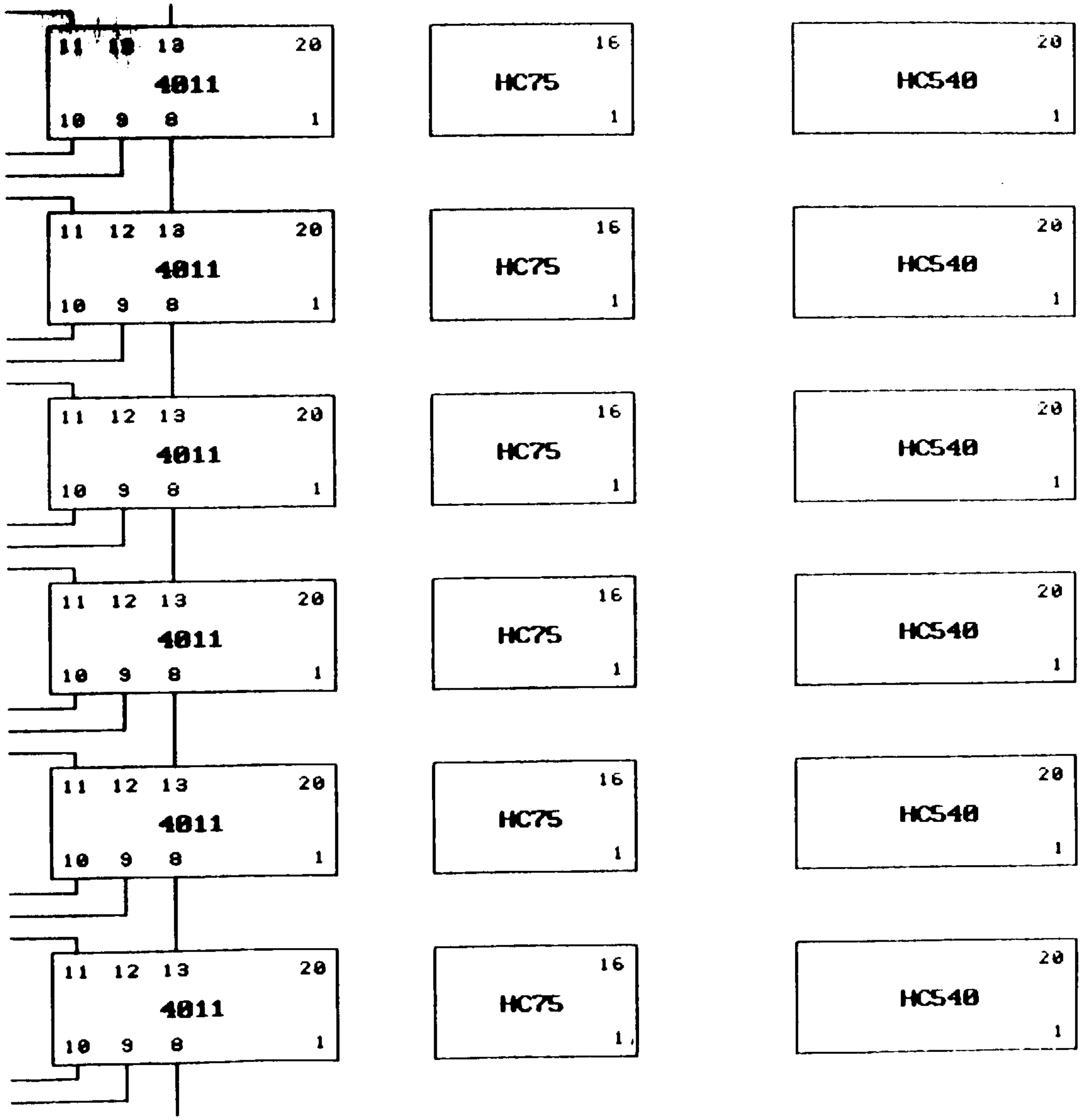


Figure 8

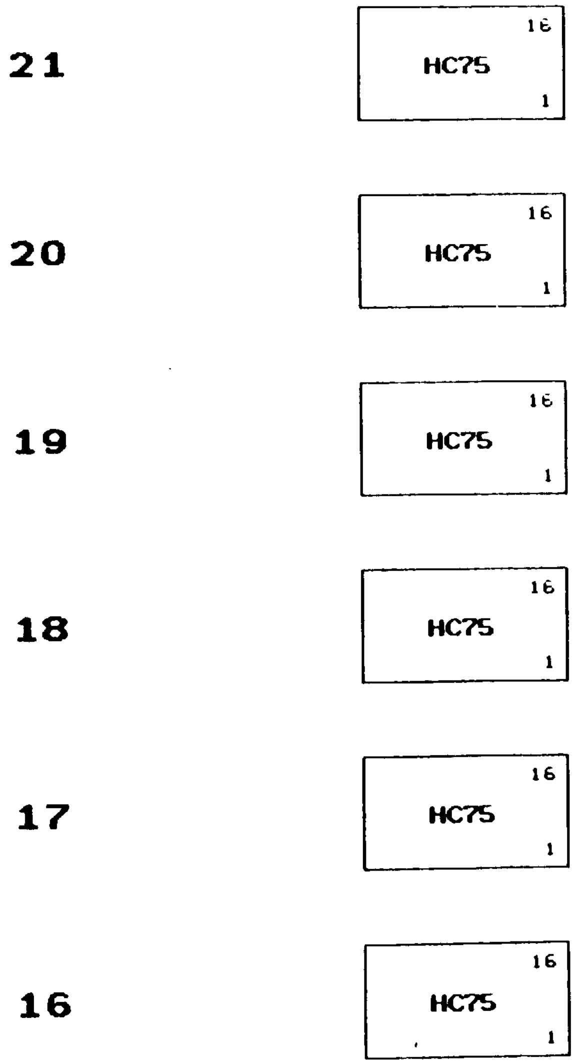


Figure 9

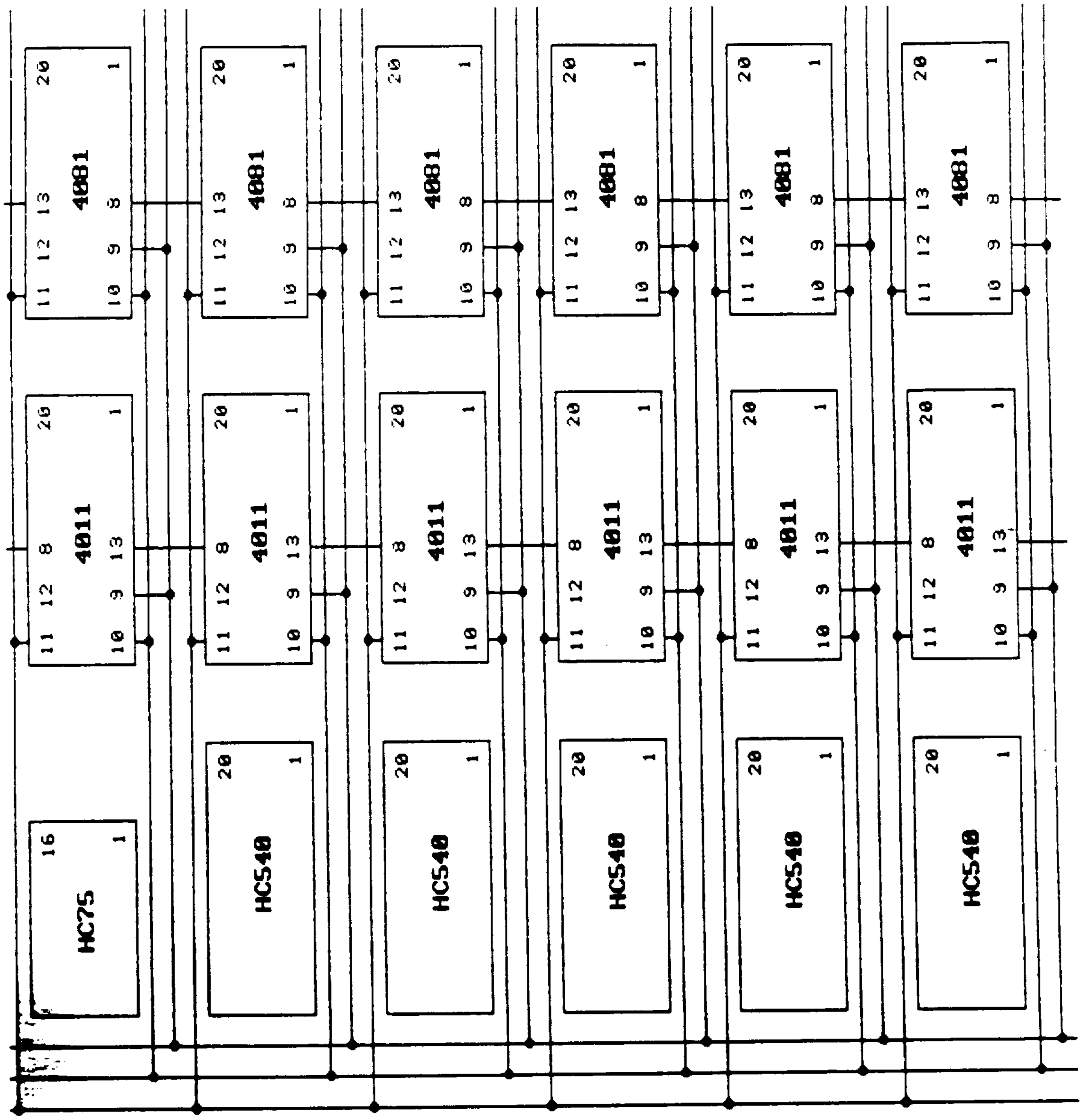

HC75
16
1
HC540
HC540
HC540
HC540
HC540
20
1
4011
4011
4011
4011
4011
4011
11 12 8
10 9 13
20
1
4081
4081
4081
4081
4081
4081
11 12 13
10 9 8
20
1

Figure 10

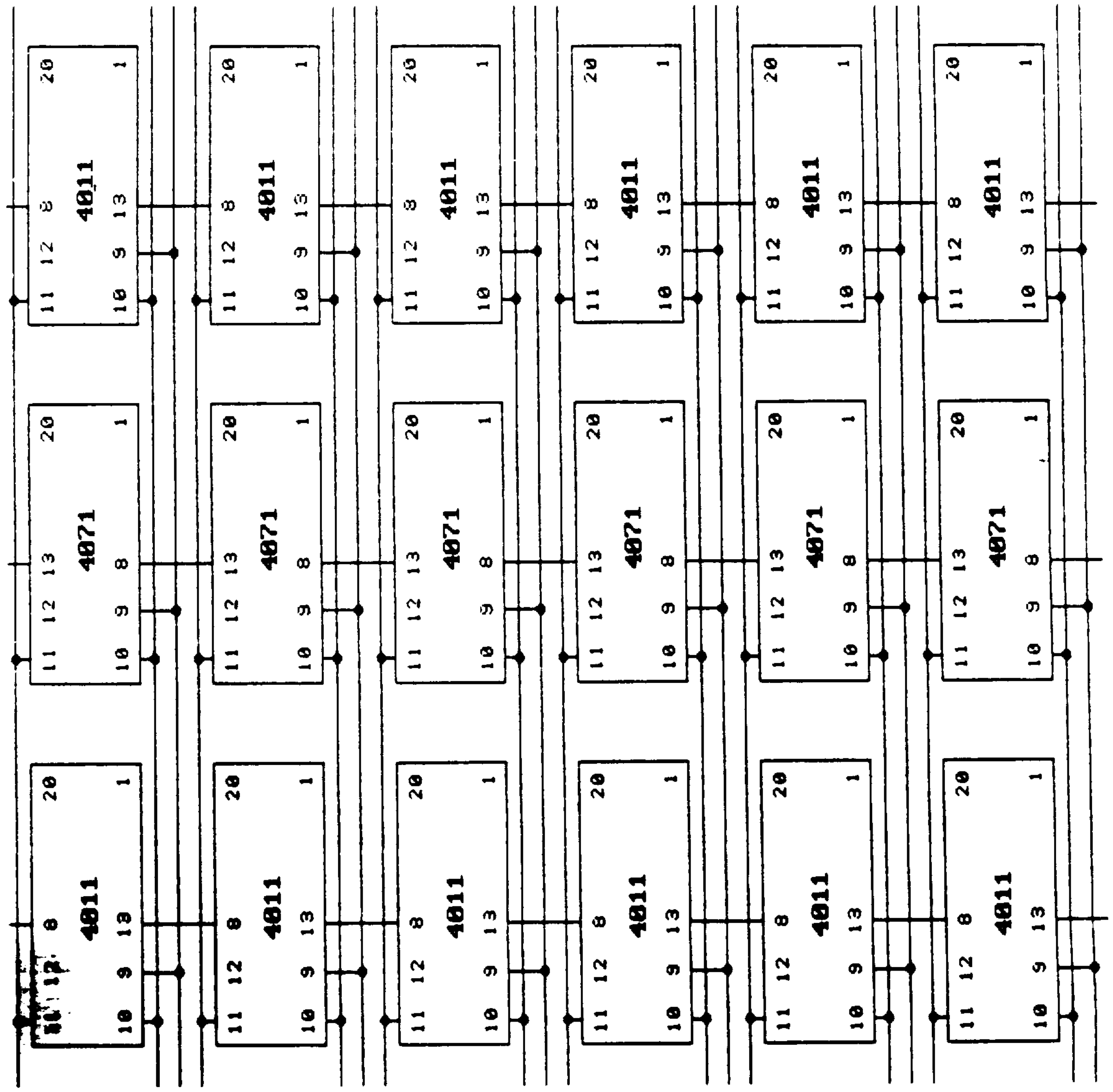


Figure 11

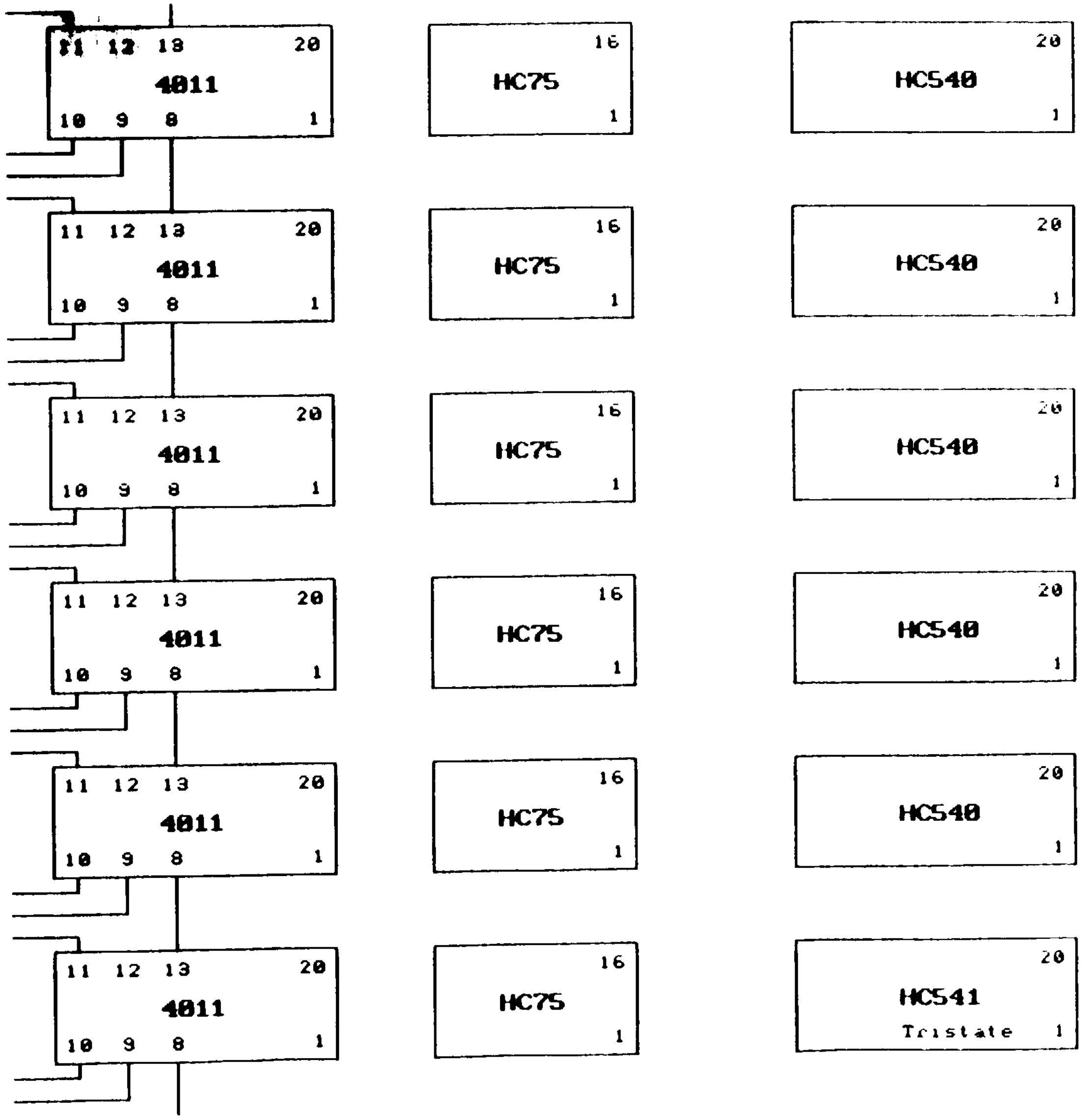


Figure 12

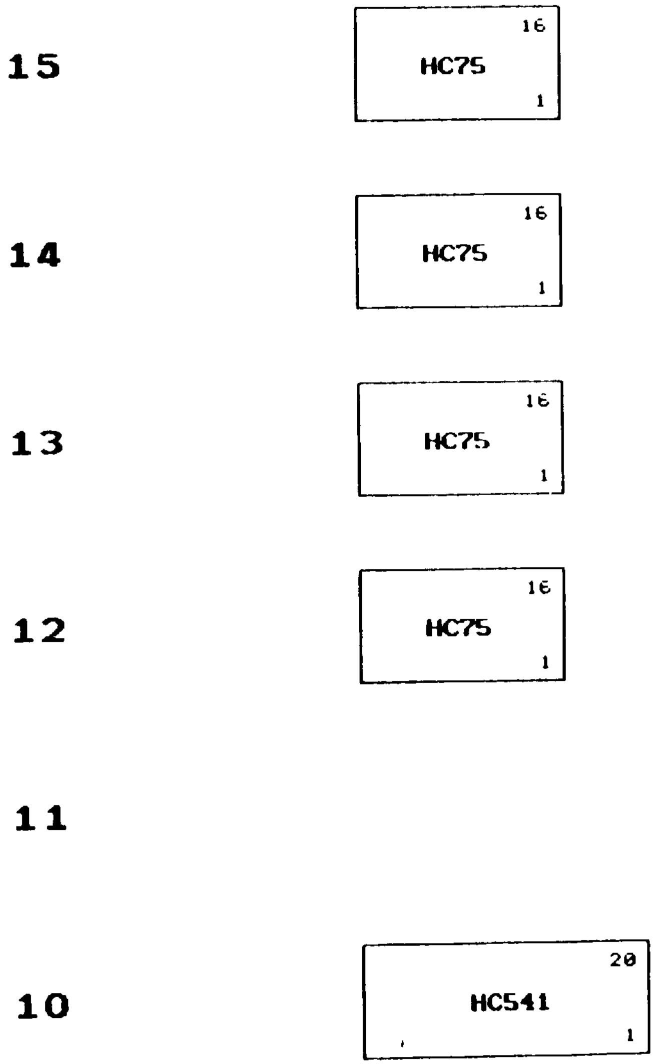


Figure 13

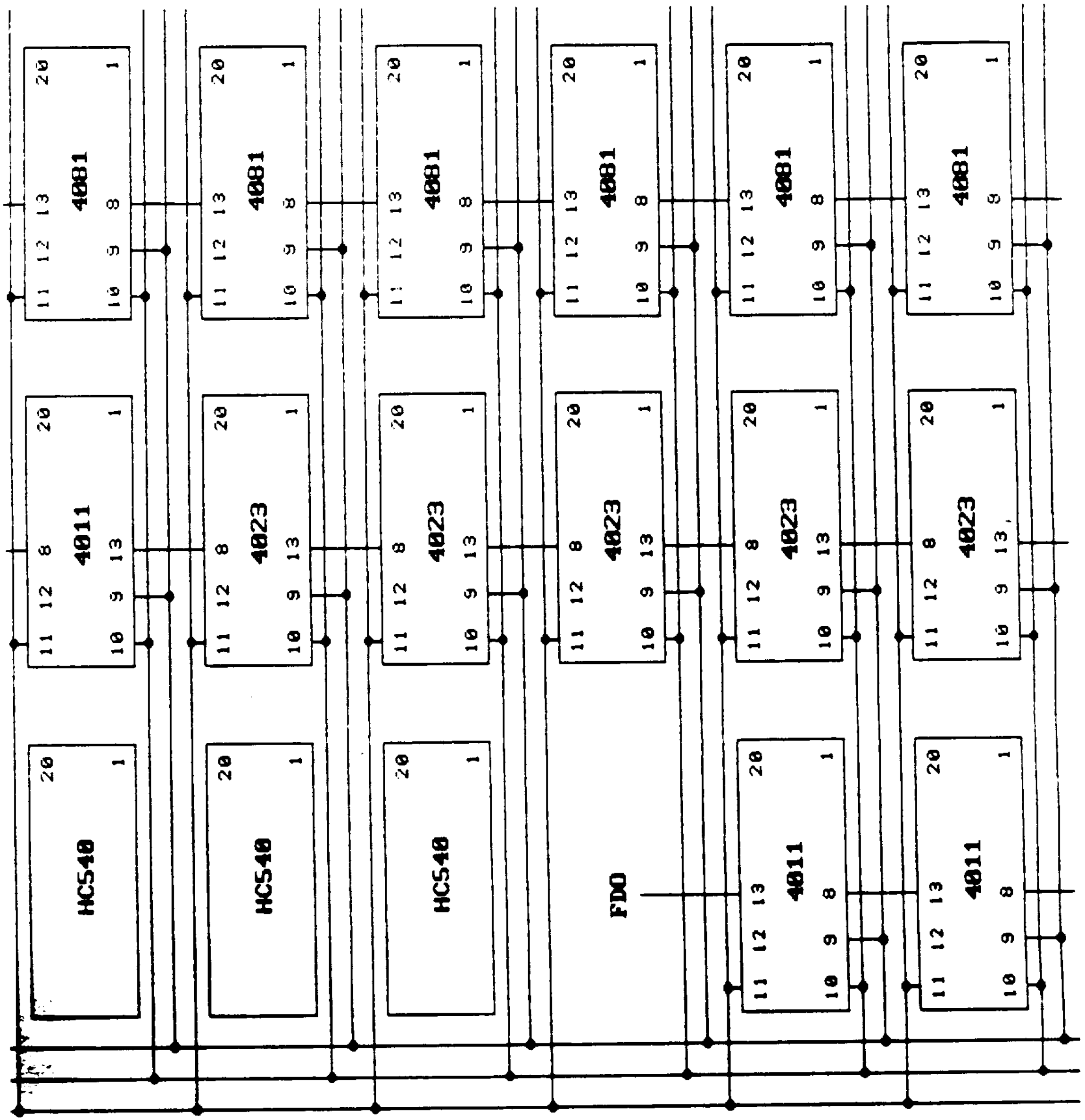

HC540
HC540
HC540
FDO
4011
4011
4011
4023
4023
4023
4023
4023
4081
4081
4081
4081
4081
4081
20
1
13
12
11
8
9
10

Figure 14

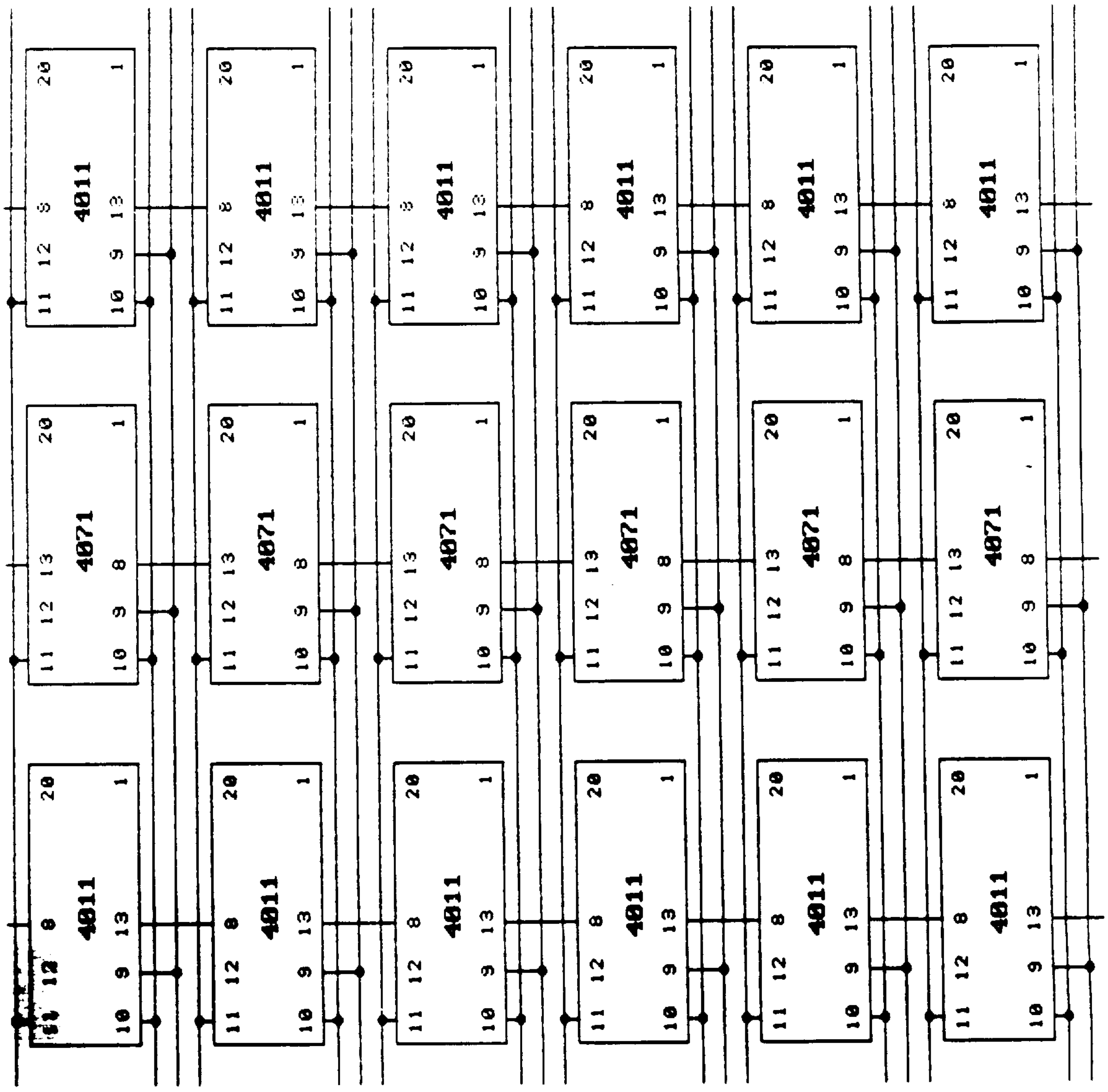


Figure 15

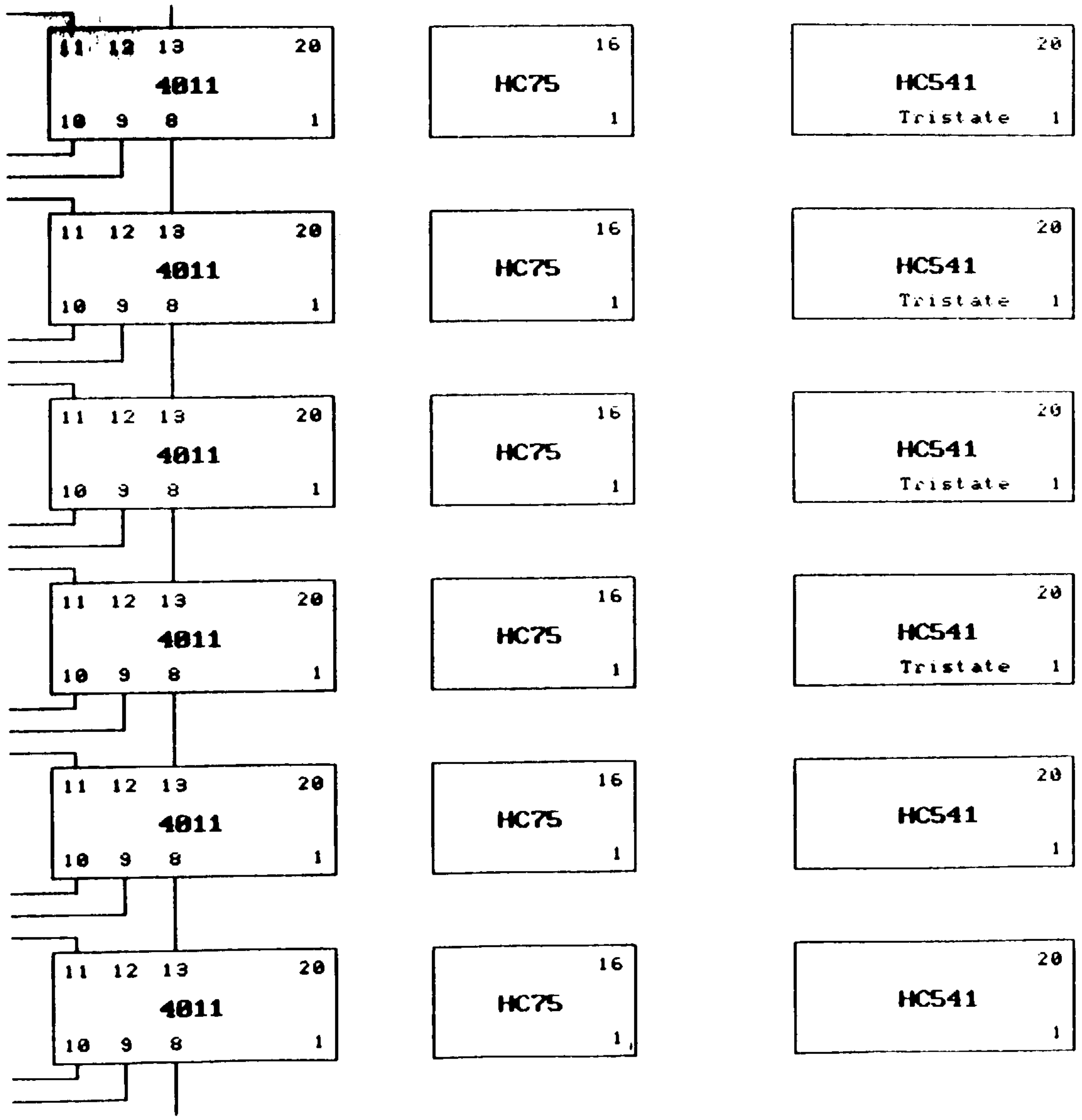


Figure 16

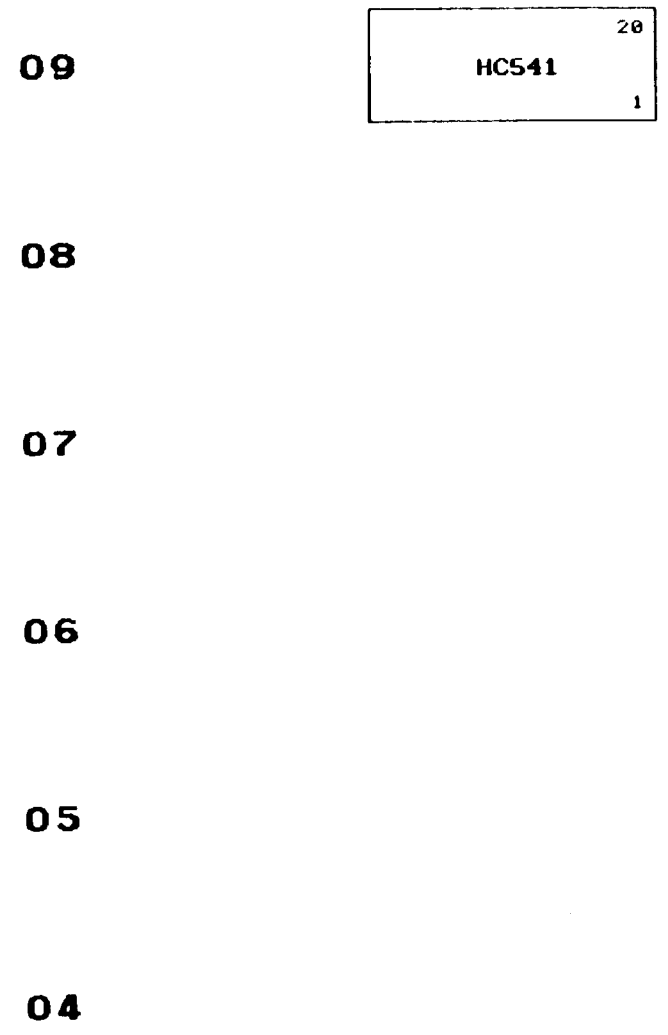


Figure 17

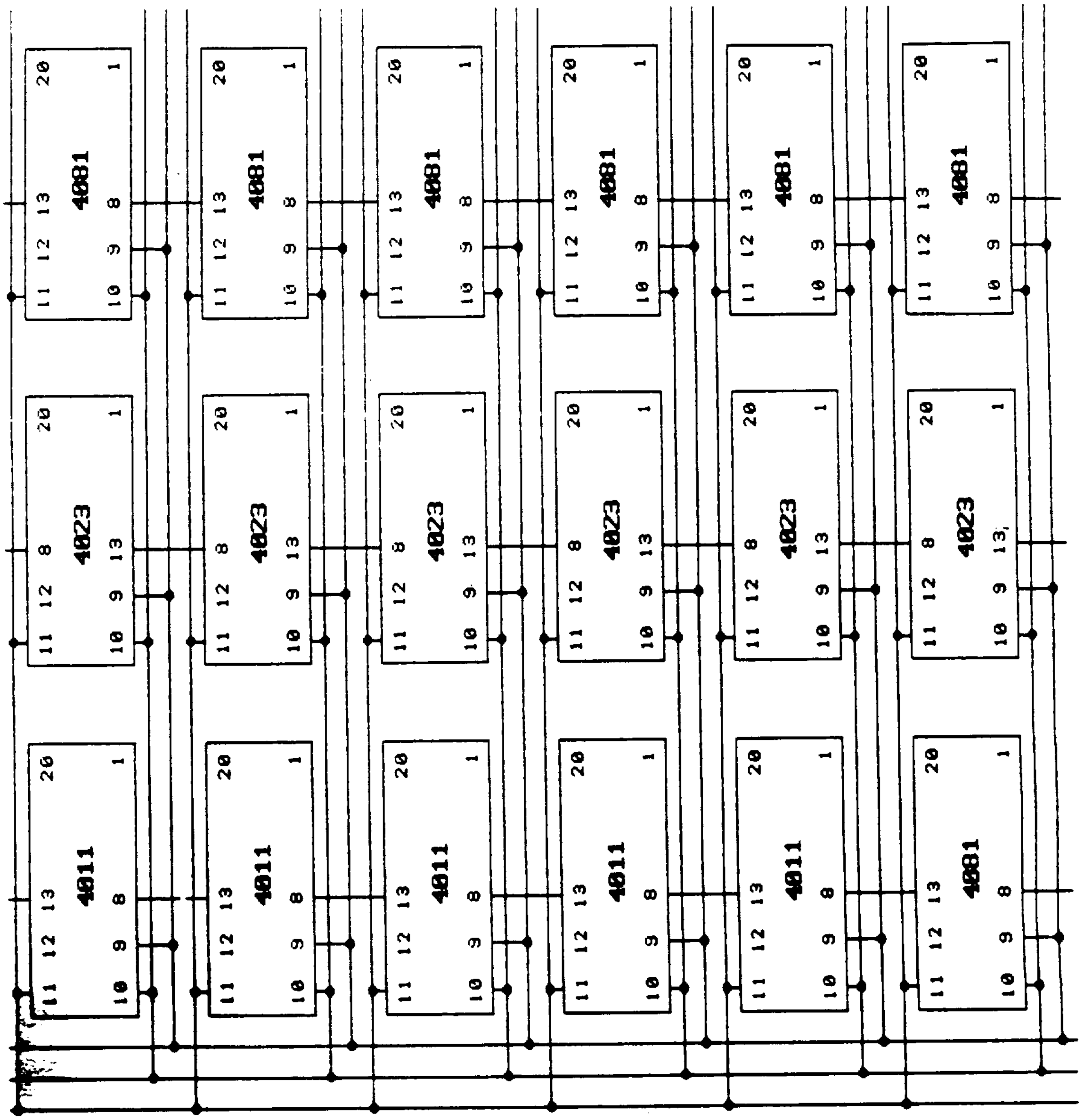

4081
4081
4081
4081
4081
4081
4023
4023
4023
4023
4023
4023
4011
4011
4011
4011
4011
4081
20
1
11
12
13
10
9
8

Figure 18

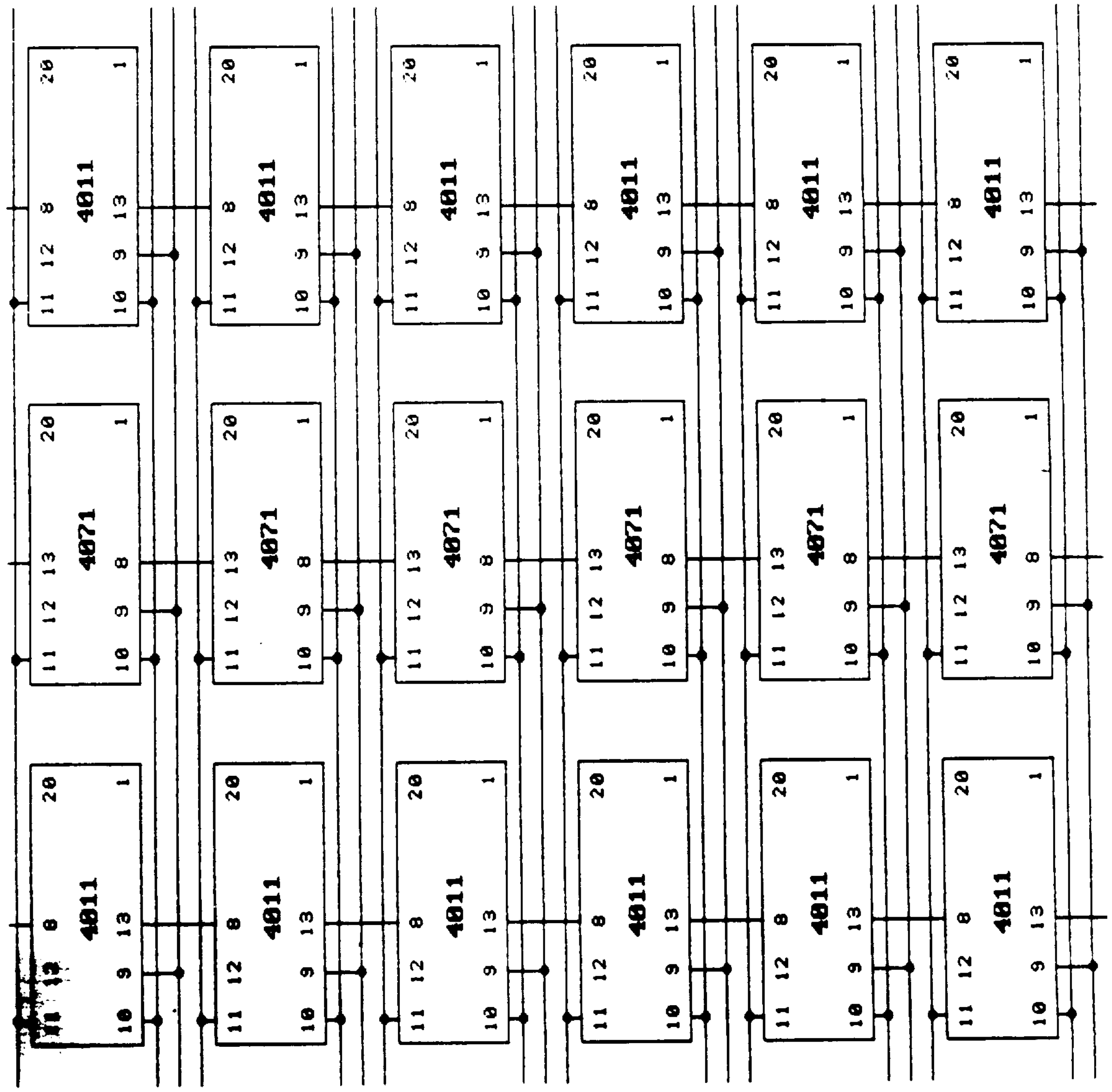


Figure 19

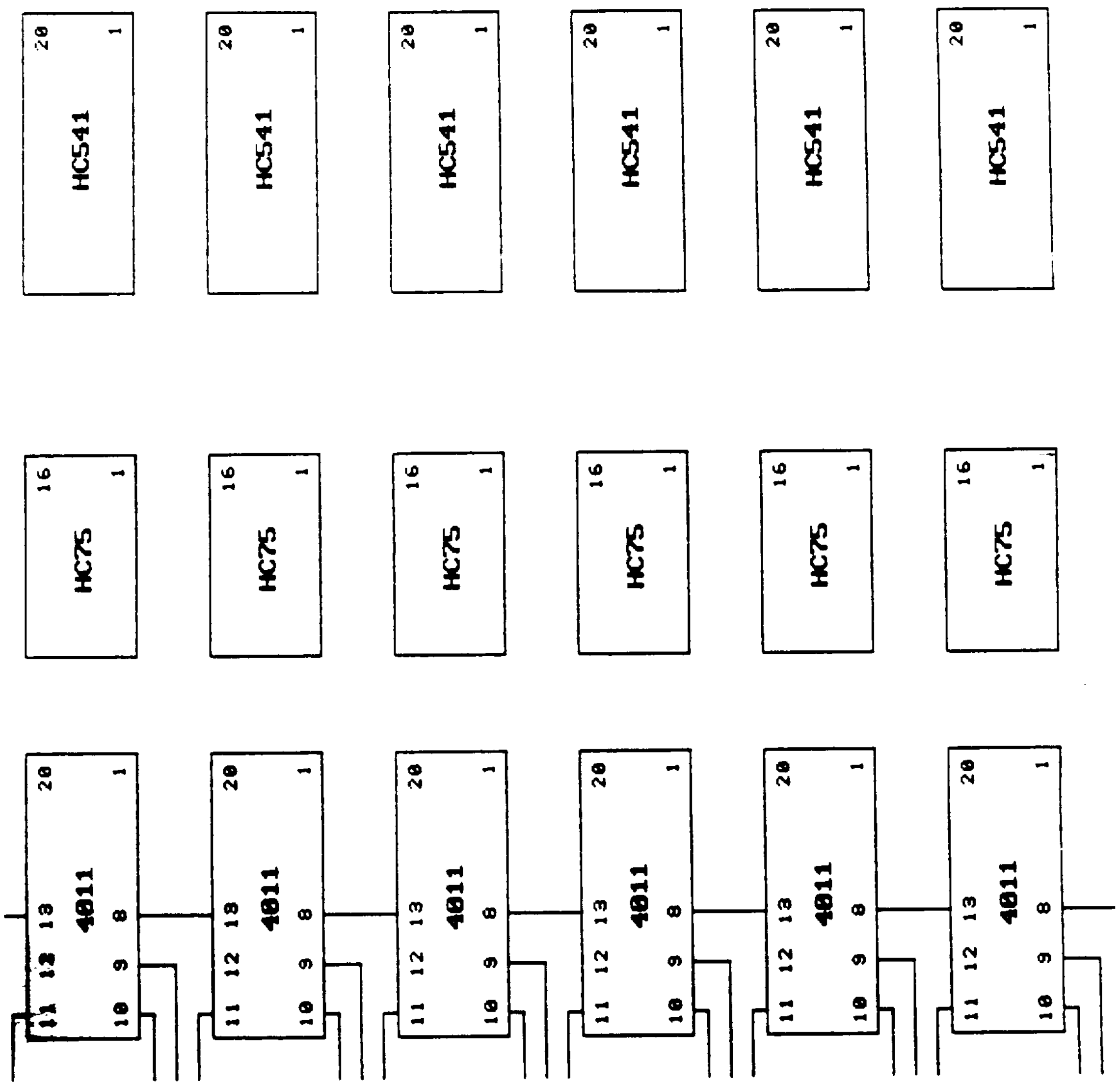


Figure 2Ø

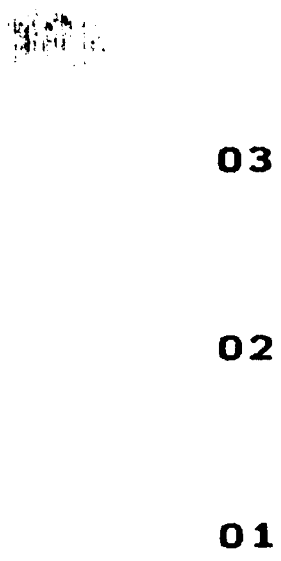


Figure 21

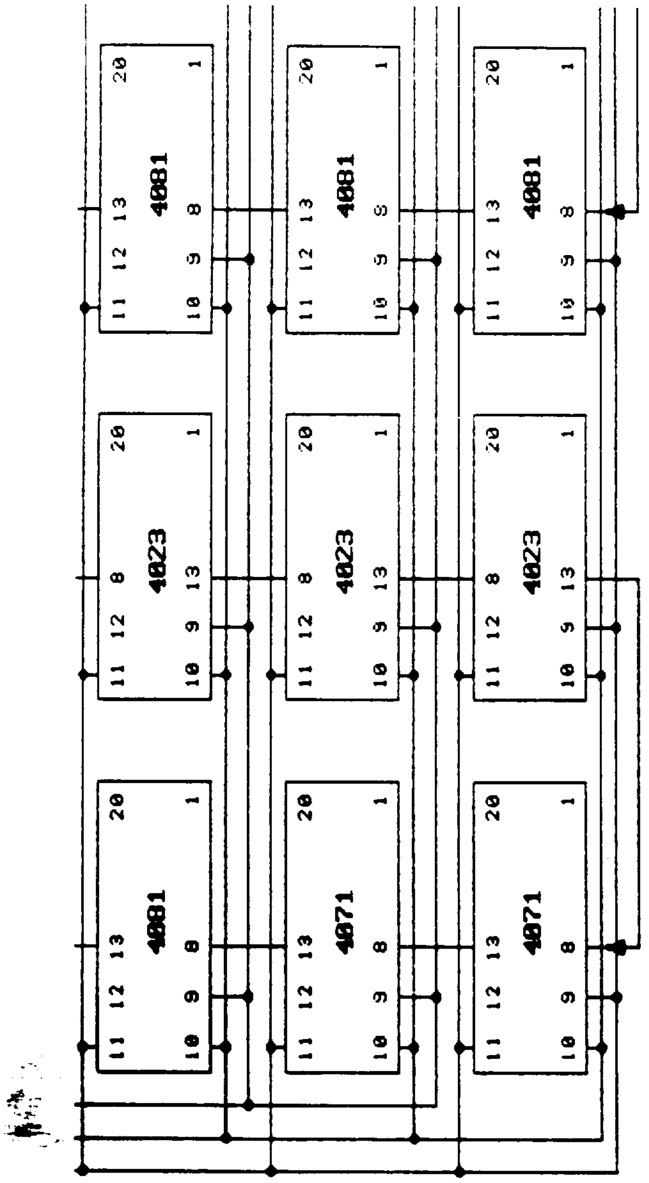
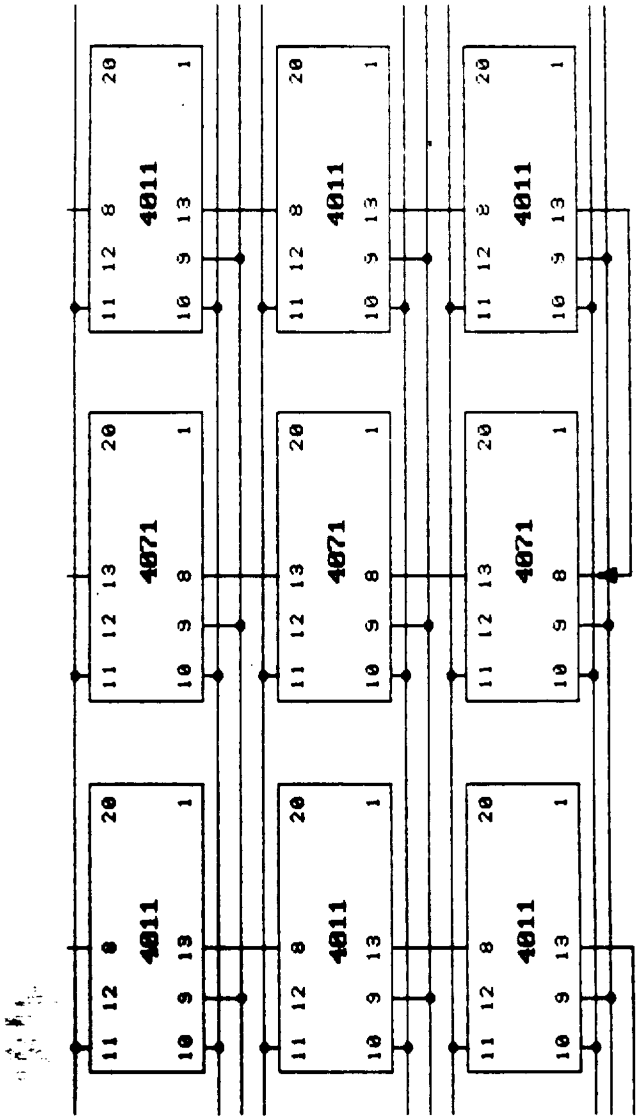

4081
11 12 13 20 10 9 8 1
4023
11 12 8 20 10 9 13 1
4081
11 12 13 20 10 9 8 1
4081
11 12 13 20 10 9 8 1
4023
11 12 8 20 10 9 13 1
4071
11 12 13 20 10 9 8 1
4081
11 12 13 20 10 9 8 1
4023
11 12 8 20 10 9 13 1
4071
11 12 13 20 10 9 8 1

Figure 22



11 12 8 20
4011
10 9 13 1
11 12 13 20
4071
10 9 8 1
11 12 8 20
4011
10 9 13 1
11 12 8 20
4011
10 9 13 1
11 12 13 20
4071
10 9 8 1
11 12 8 20
4011
10 9 13 1
11 12 8 20
4011
10 9 13 1
11 12 13 20
4071
10 9 8 1
11 12 8 20
4011
10 9 13 1

Figure 23

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

NAS7-1028

PART 3

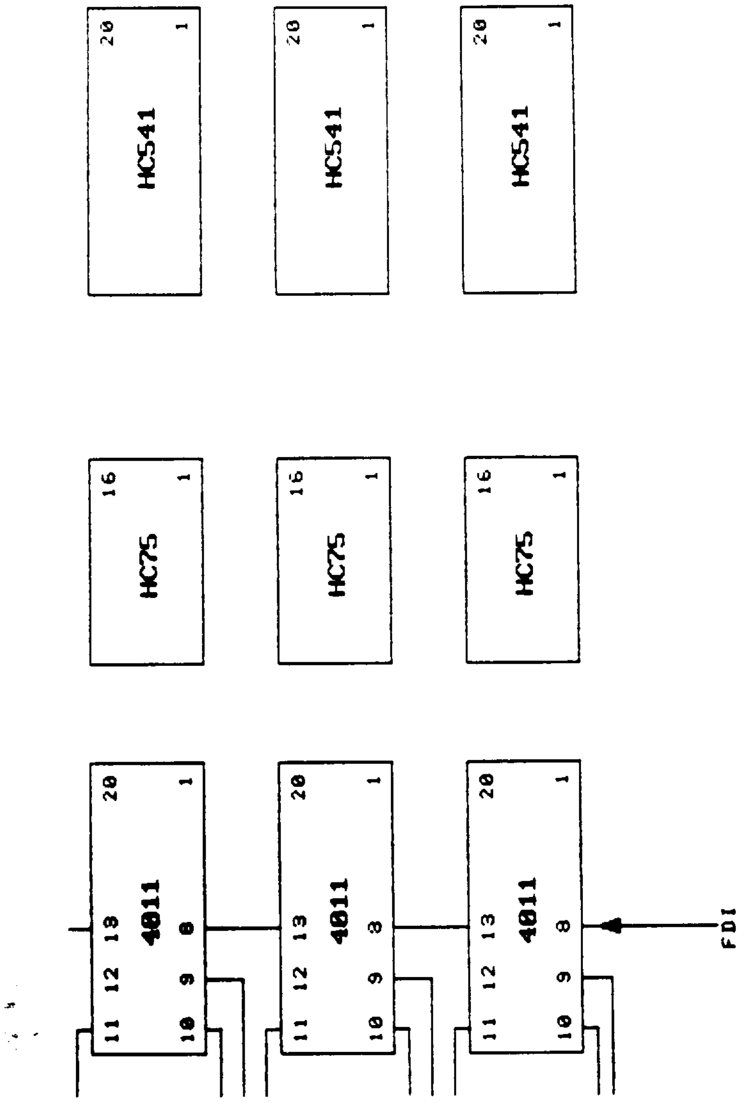

HC541
20
1
HC541
20
1
HC541
20
1
HC75
16
1
HC75
16
1
HC75
16
1
4011
20
1
11 12 13
10 9 8
4011
20
1
11 12 13
10 9 8
4011
20
1
11 12 13
10 9 8
FDI

Figure 24

Quad 2-Input NAND Gate
Microsimulator
Part Number 4HFS11

1 March 1981

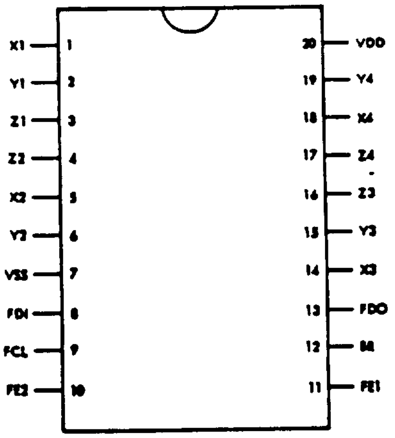
Pin Definitions

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.

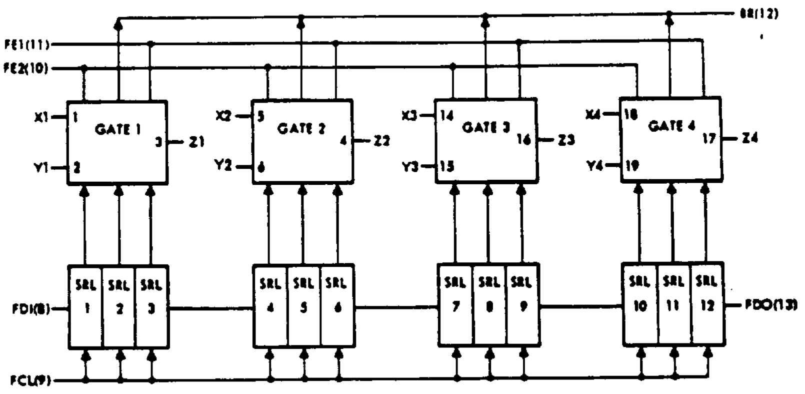
Each nand gate with inputs Xi, Yi and output Zi is connected to three shift register latches (SRL's) from the 12-bit shift register as follows:

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 Z1 can be made observable at BR if an "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 Zl 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 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 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-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 (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

Triple 3-Input NAND Gate
Microsimulator
Part Number 4HFS023

1 March 1981

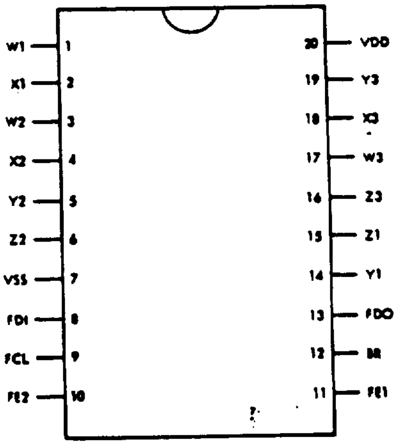
Pin Definitions

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

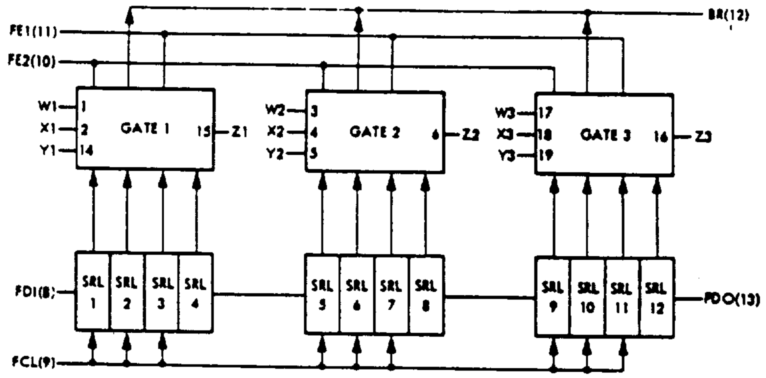
Each OR gate with inputs Wi, Xi, Yi and output Zi is connected to four shift register latches (SRL's) from the 12-bit shift register as follows:

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 Wi, 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 Wi is being injected. A "1" in the second SRL of a gate injects a stuck-at-one at Xi and a "1" in the third SRL of a gate injects a stuck-at-one at Yi. Moreover, a "1" in the fourth SRL 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 OR function from VDD or VSS. For example, Wi stuck-open disconnects VDD from the p-transistor whose gate is connected to Wi; 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, Zi 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.

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	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 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 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 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 6 Switching characteristics

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

Quad 2-Input OR Gate
Microsimulator
Part Number 4HFS071

1 March 1981

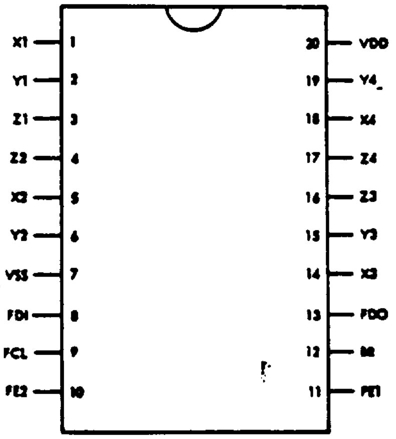
Pin Definitions

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

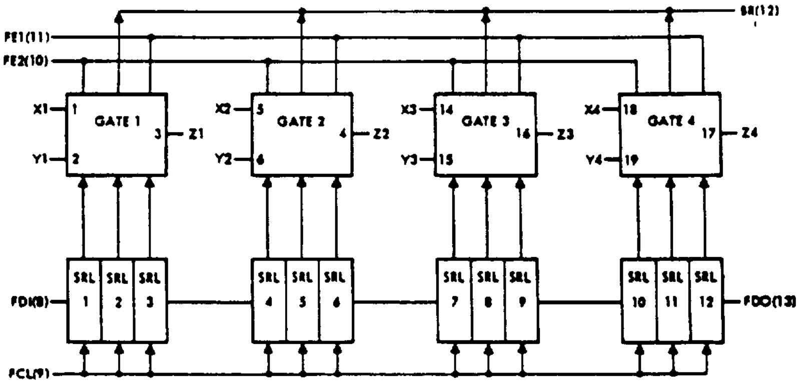
Each OR gate with inputs Xi, Yi and output Zi is connected to three shift register latches (SRL's) from the 12-bit shift register as follows:

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 4071 quad 2-input OR. Additionally, the logic value appearing at Z1 can be made observable at BR if an "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 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. Xi stuck-open disconnects the VSS from the n 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 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 Xi or Yi to Zi	50 pF	400 nsec
Propagation delay from Zi 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

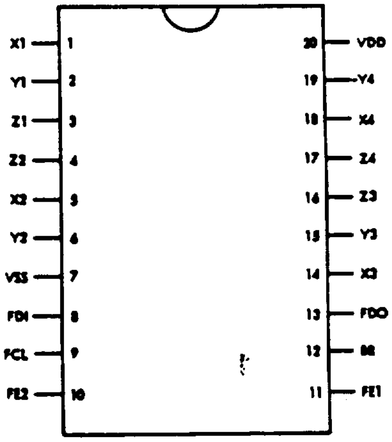
1 March 1981

	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 Xi, Yi and Output Zi is connected to three shift register latches (SRL's) from the 12-bit shift register as follows:

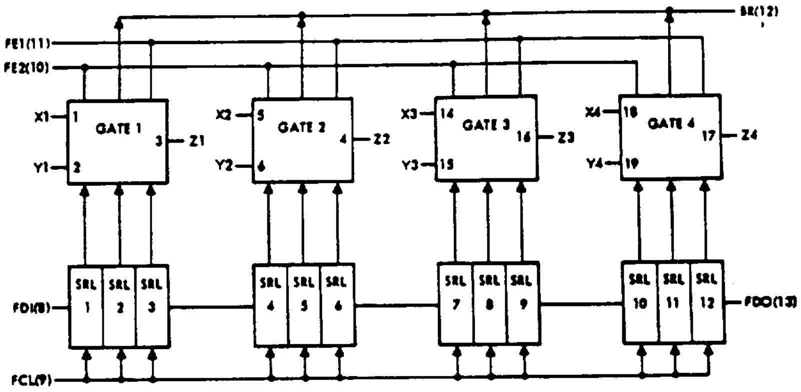
SRL1, SRL2, SRL3	are connected to Gate 1
SRL4, SRL5, SRL6	are connected to Gate 2
SRL7, SRL8, SRL9	are connected to Gate 3
SRL10, SRL11, SRL 12	are connected to Gate 4

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



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 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 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-zero 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-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. 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 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 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 Xi or Yi to Zi	50 pF	400 nsec
Propagation delay from Zi 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
```

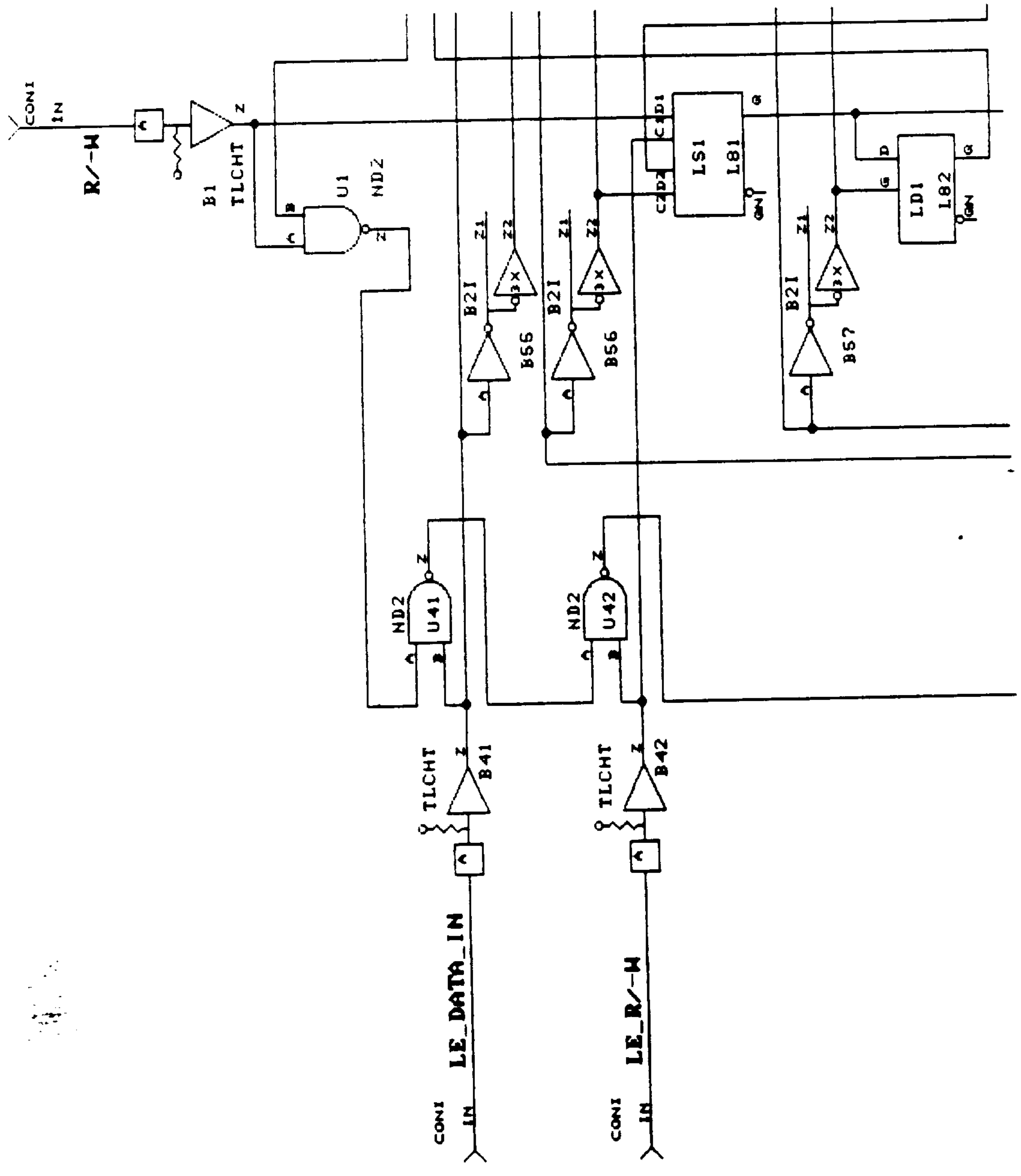
```
     EDAC Fault Simulation (EDAC logic)                Stuck-open


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

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



R/-W
CONI
IN
B1
TLCHT
Z
U1
ND2
B56
B57
B21
Z1
Z2
3X
LS1
L81
LD1
L82
ND2
U41
U42
TLCHT
B41
B42
LE_DATA_IN
LE_R/-W
CONI
IN

Figure 1

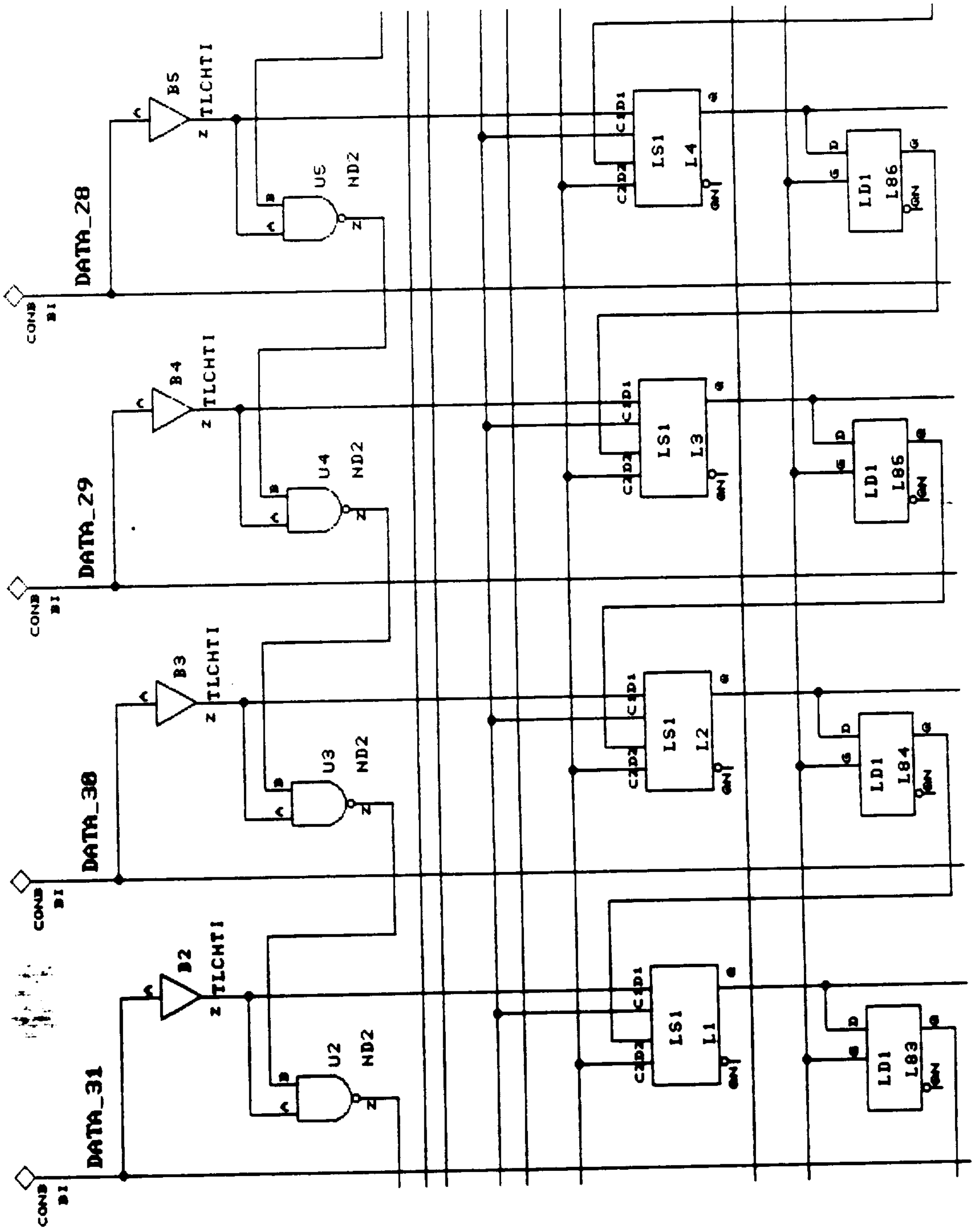

DATA_28
DATA_29
DATA_30
DATA_31
CONB BI
B5 TLCHTI
B4 TLCHTI
B3 TLCHTI
B2 TLCHTI
U5 ND2
U4 ND2
U3 ND2
U2 ND2
LS1 L4
LS1 L3
LS1 L2
LS1 L1
LD1 L86
LD1 L86
LD1 L84
LD1 L83
C1D1
C2D2
Q
GN
D
G
A
B
Z

Figure 2

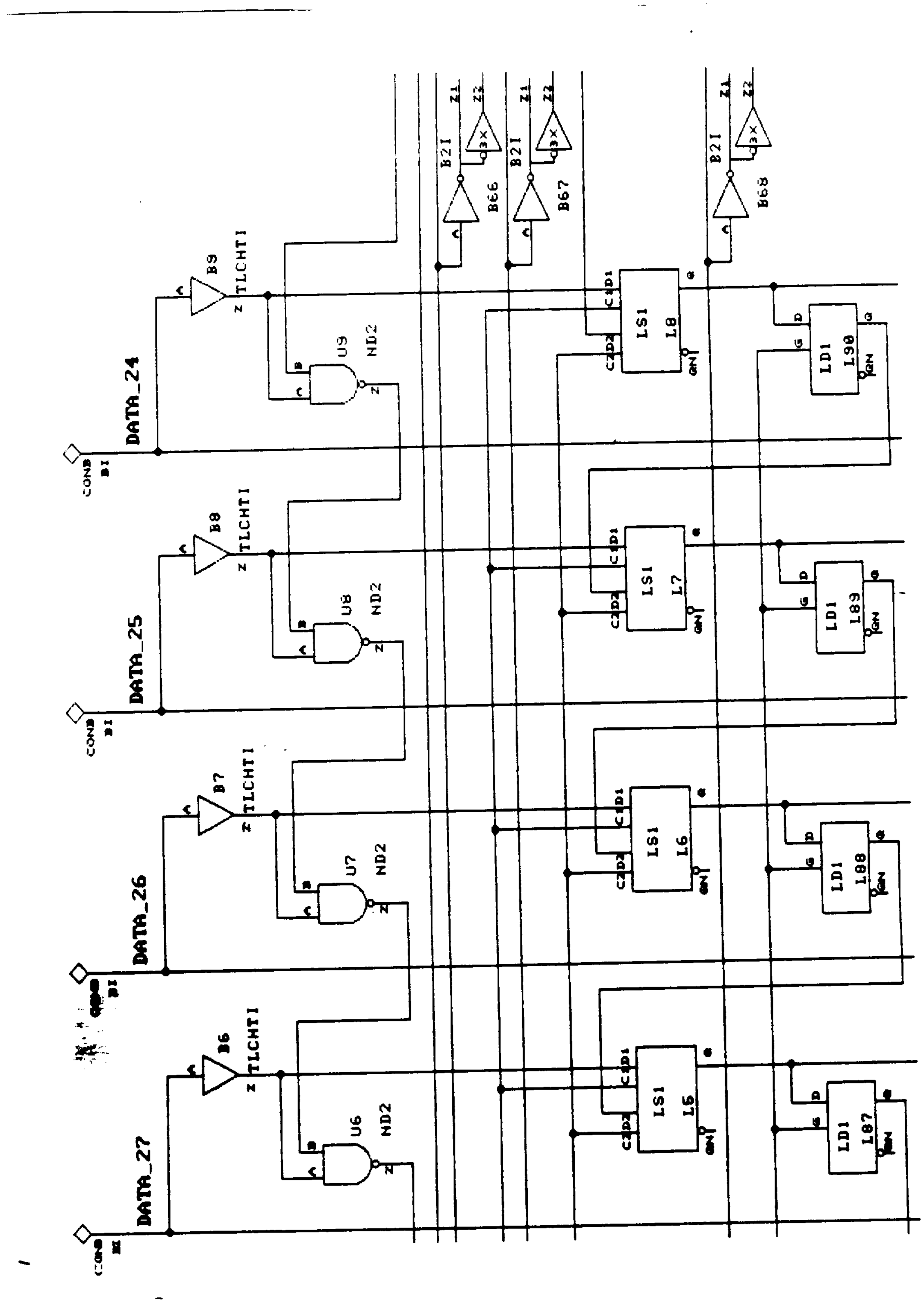

DATA_24
DATA_25
DATA_26
DATA_27
CONB
BI
B9
B8
B7
B6
TLCHTI
A
Z
U9
U8
U7
U6
ND2
B
LS1
L8
L7
L6
L5
CSD1
CZD2
Q
GN
LD1
L90
L89
L88
L87
D
G
B66
B67
B68
B2I
Z1
Z2
3X

Figure 3

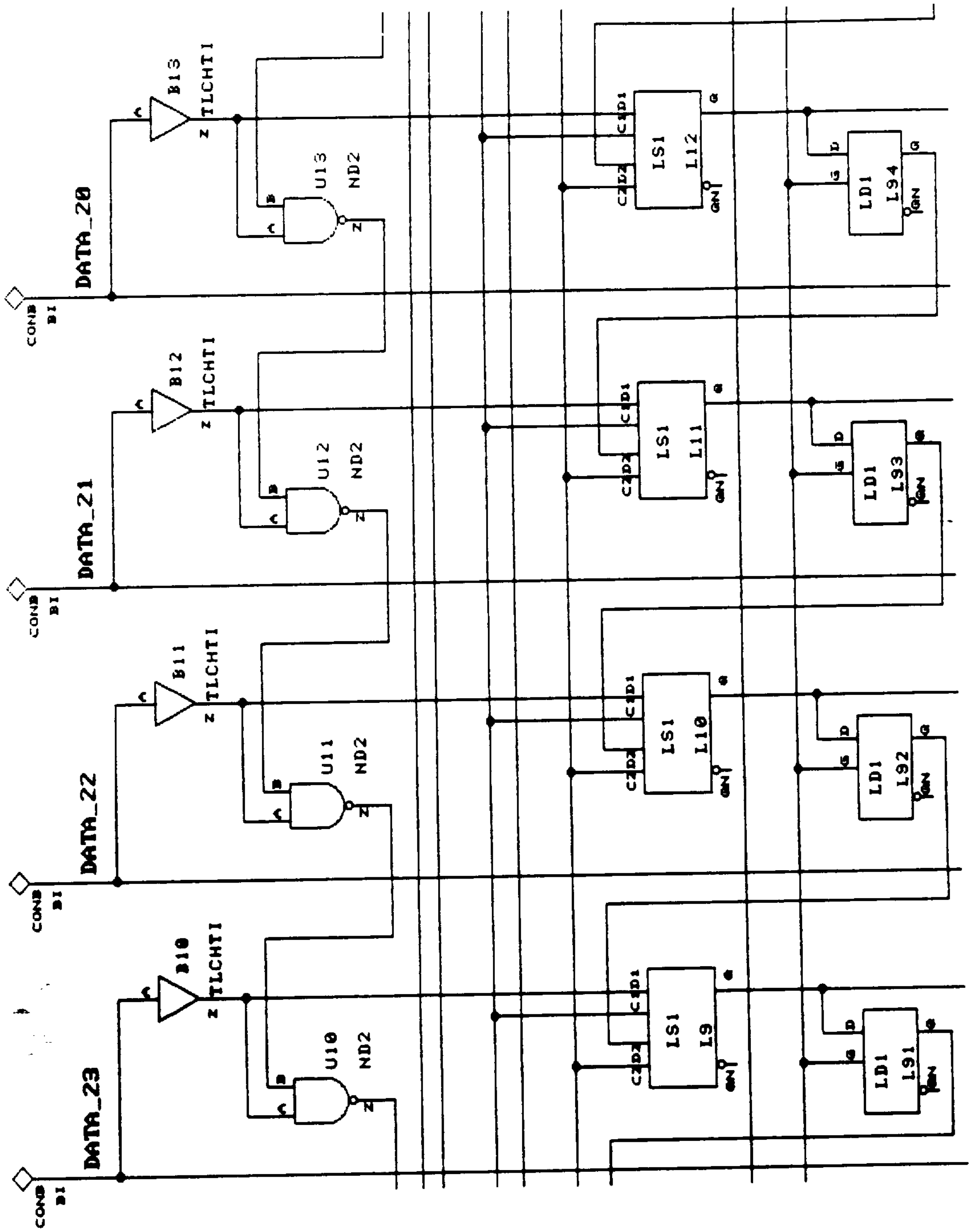

DATA_20
DATA_21
DATA_22
DATA_23
CONB
BI
B13
B12
B11
B10
TLCHTI
U13
U12
U11
U10
ND2
LS1
L12
L11
L10
L9
LD1
L94
L93
L92
L91
C1D1
C2D2
GN
D
G
Q
A
B
Z

Figure 4

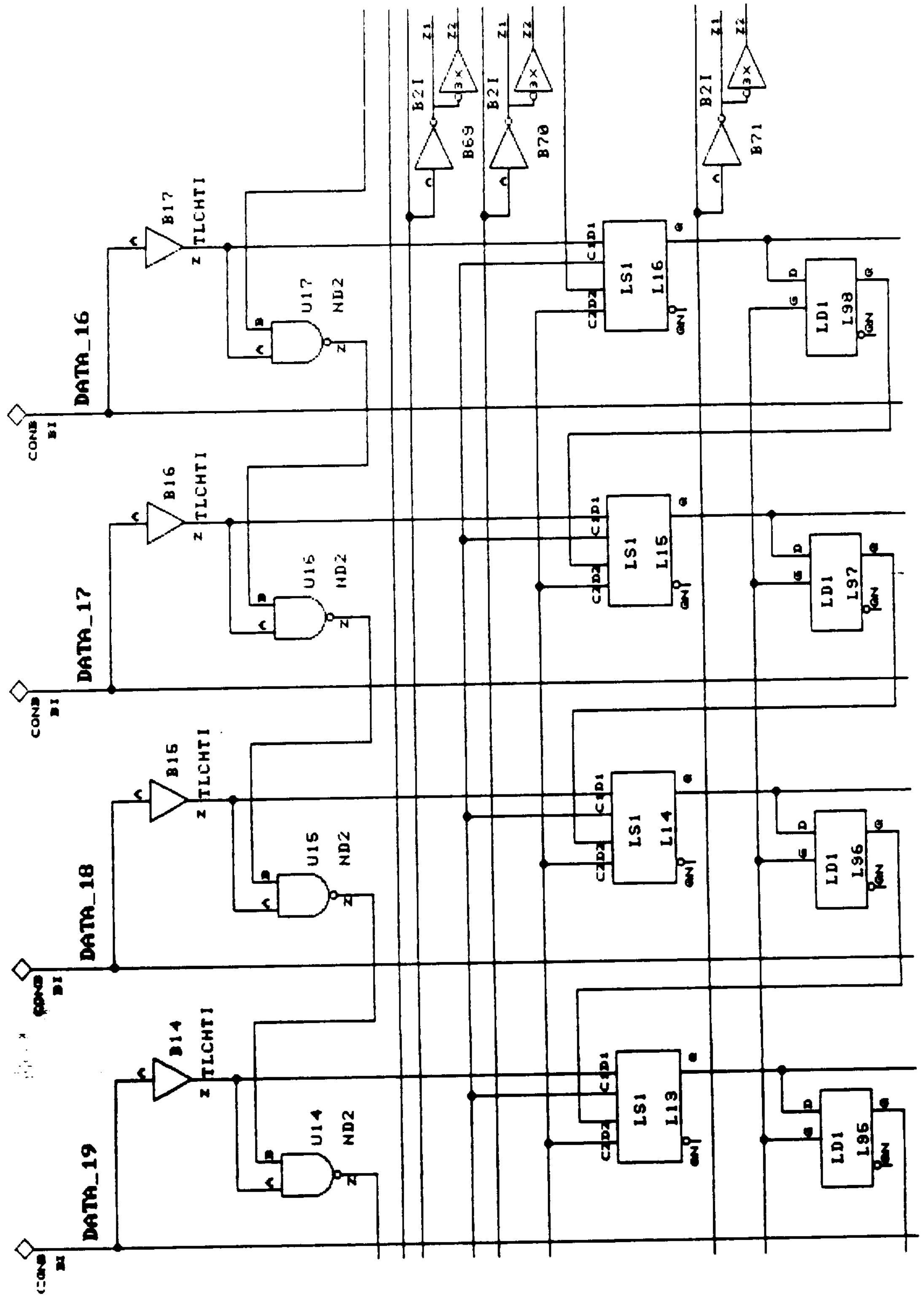

DATA_16
DATA_17
DATA_18
DATA_19
CONB
BI
B17
B16
B15
B14
TLCHTI
U17
U16
U15
U14
ND2
LS1
L16
L15
L14
L13
LD1
L98
L97
L96
L95
B69
B70
B71
B2I
Z1
Z2
3X
A
B
Z
C1D1
C2D2
G
GN
D

Figure 5

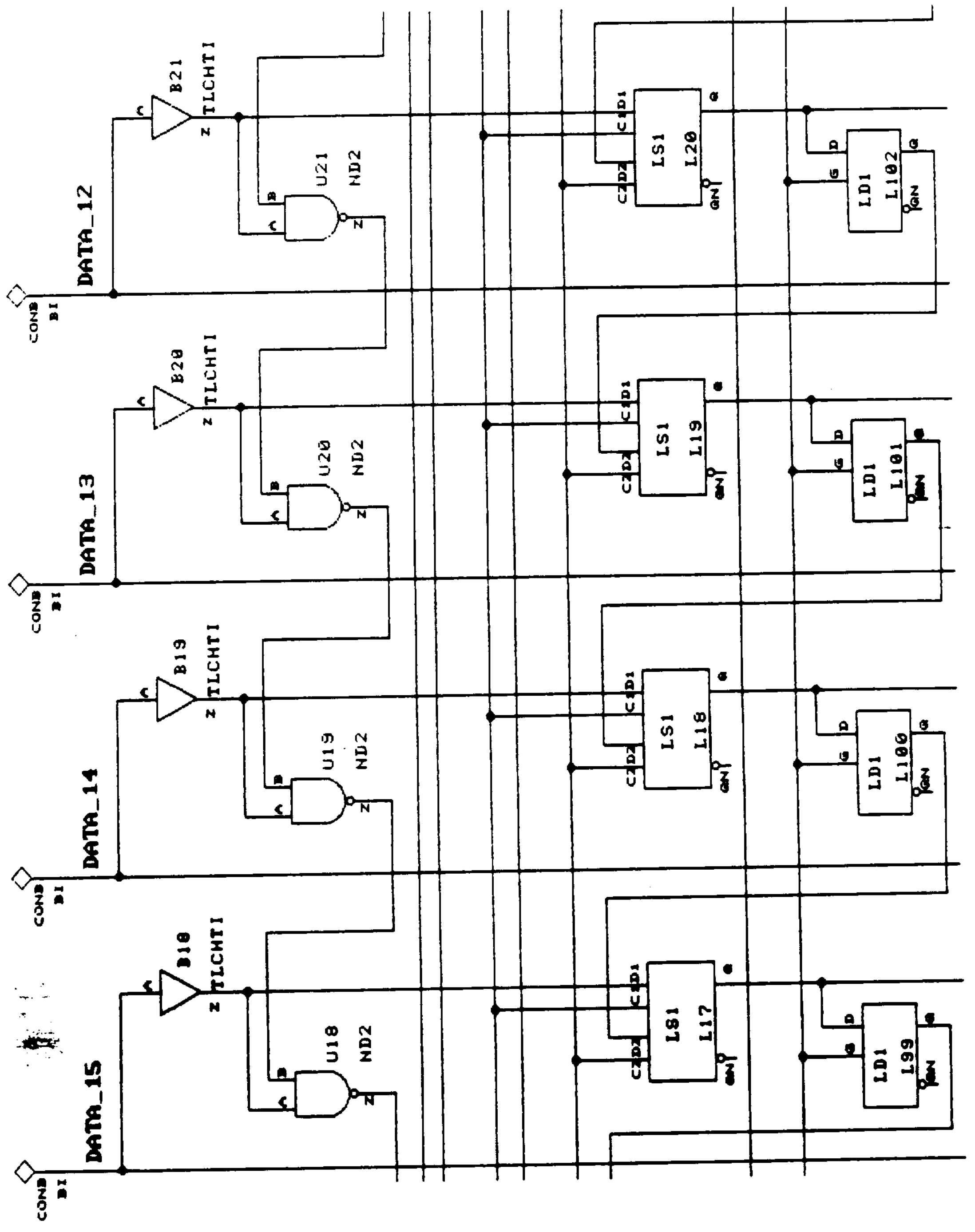

DATA_12
DATA_13
DATA_14
DATA_15
CONB
BI
B21
B20
B19
B18
TLCHTI
A
Z
U21
U20
U19
U18
ND2
B
LS1
L20
L19
L18
L17
C1D1
C2D2
Q
GN
LD1
L102
L101
L100
L99
D
G

Figure 6

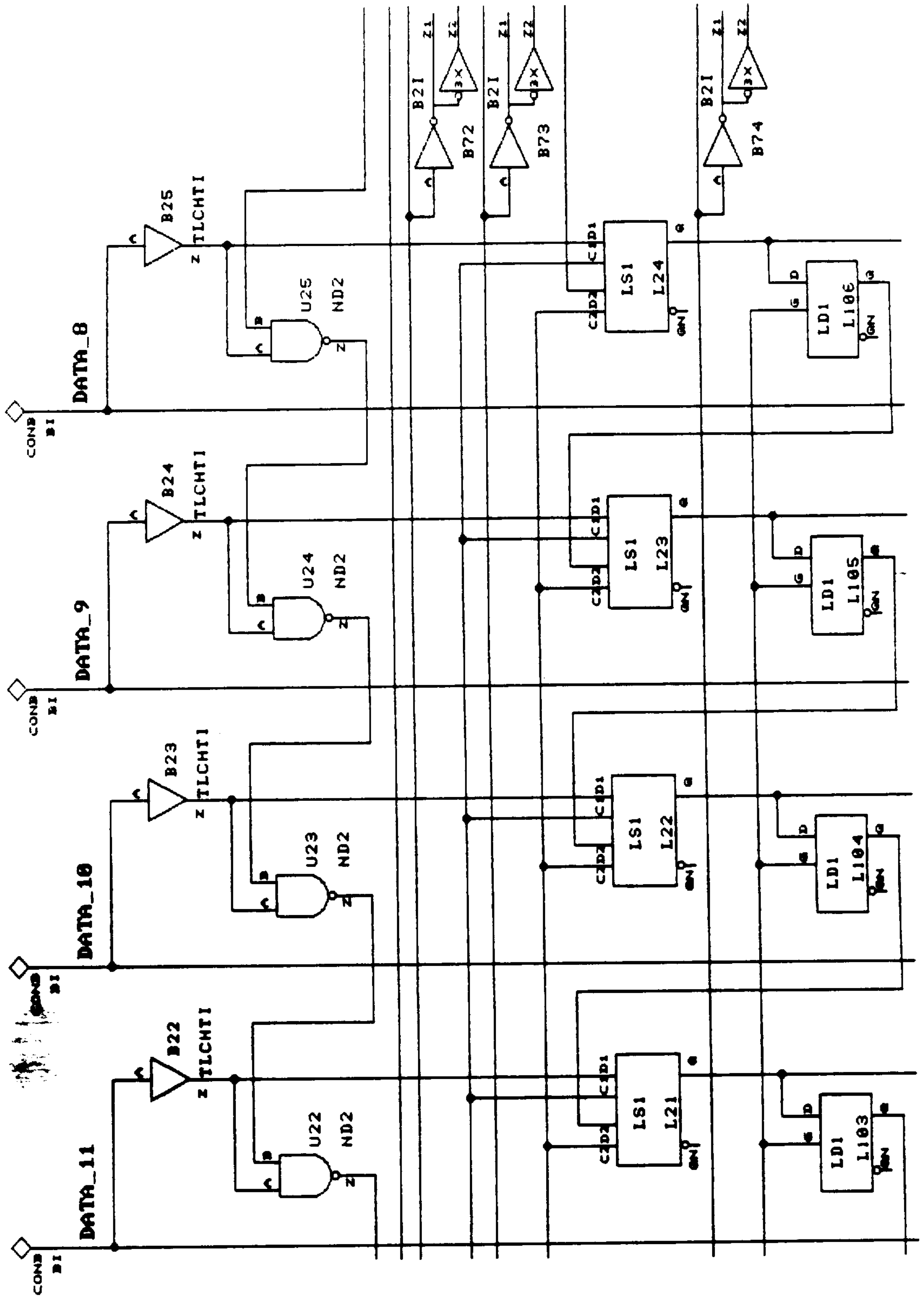

DATA_8
DATA_9
DATA_10
DATA_11
CONB
BI
B22
B23
B24
B25
TLCHTI
U22
U23
U24
U25
ND2
LS1
L21
L22
L23
L24
C1D1
C2D2
GN
LD1
L103
L104
L105
L106
B2I
B72
B73
B74
Z1
Z2
3X

Figure 7

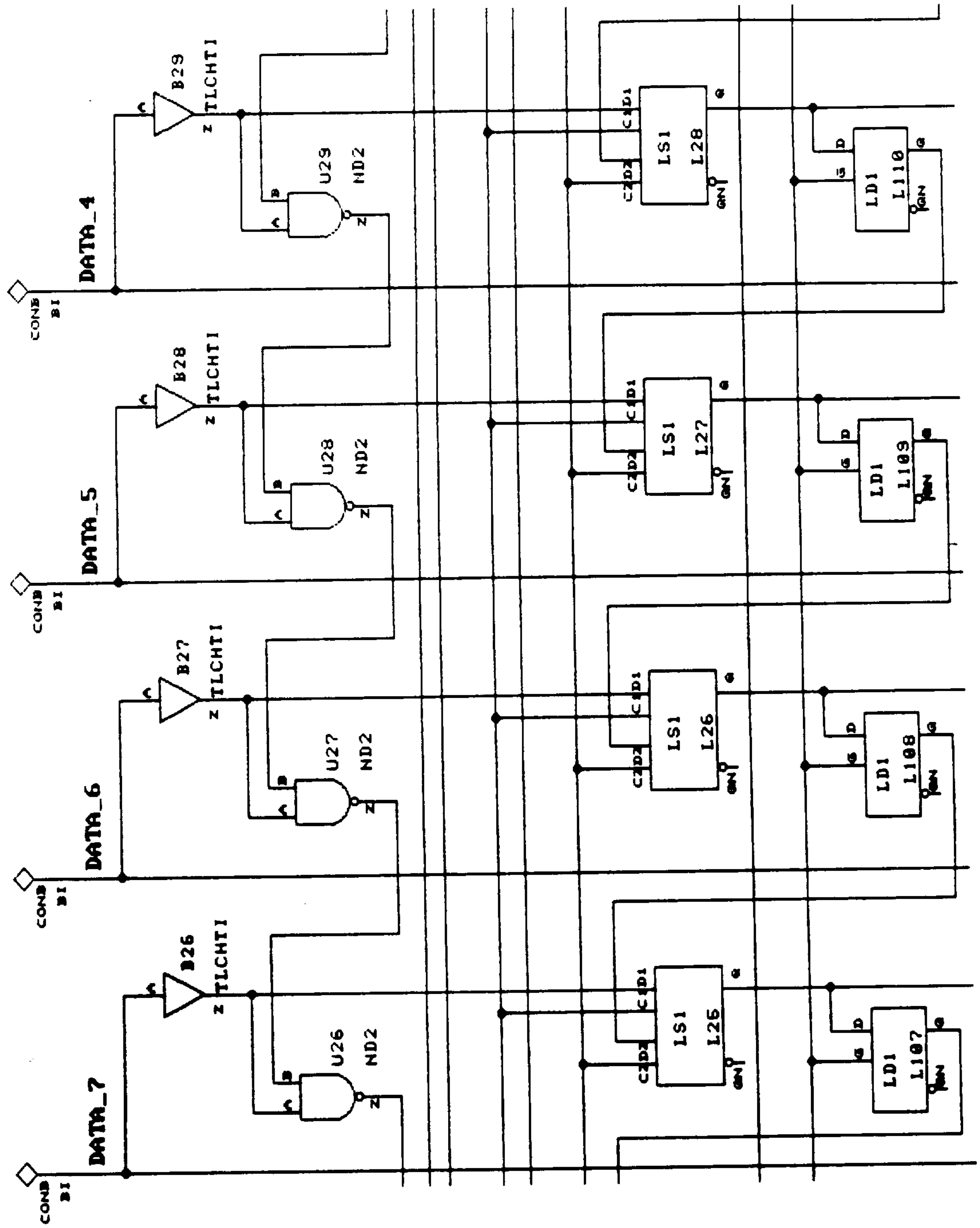

CONB BI
DATA_4
B29 TLCHTI
U29 ND2
LS1 L28
LD1 L110
CONB BI
DATA_5
B28 TLCHTI
U28 ND2
LS1 L27
LD1 L109
CONB BI
DATA_6
B27 TLCHTI
U27 ND2
LS1 L26
LD1 L108
CONB BI
DATA_7
B26 TLCHTI
U26 ND2
LS1 L25
LD1 L107
A
Z
B
C1D1
C2D2
Q
GN
D
G

Figure 8

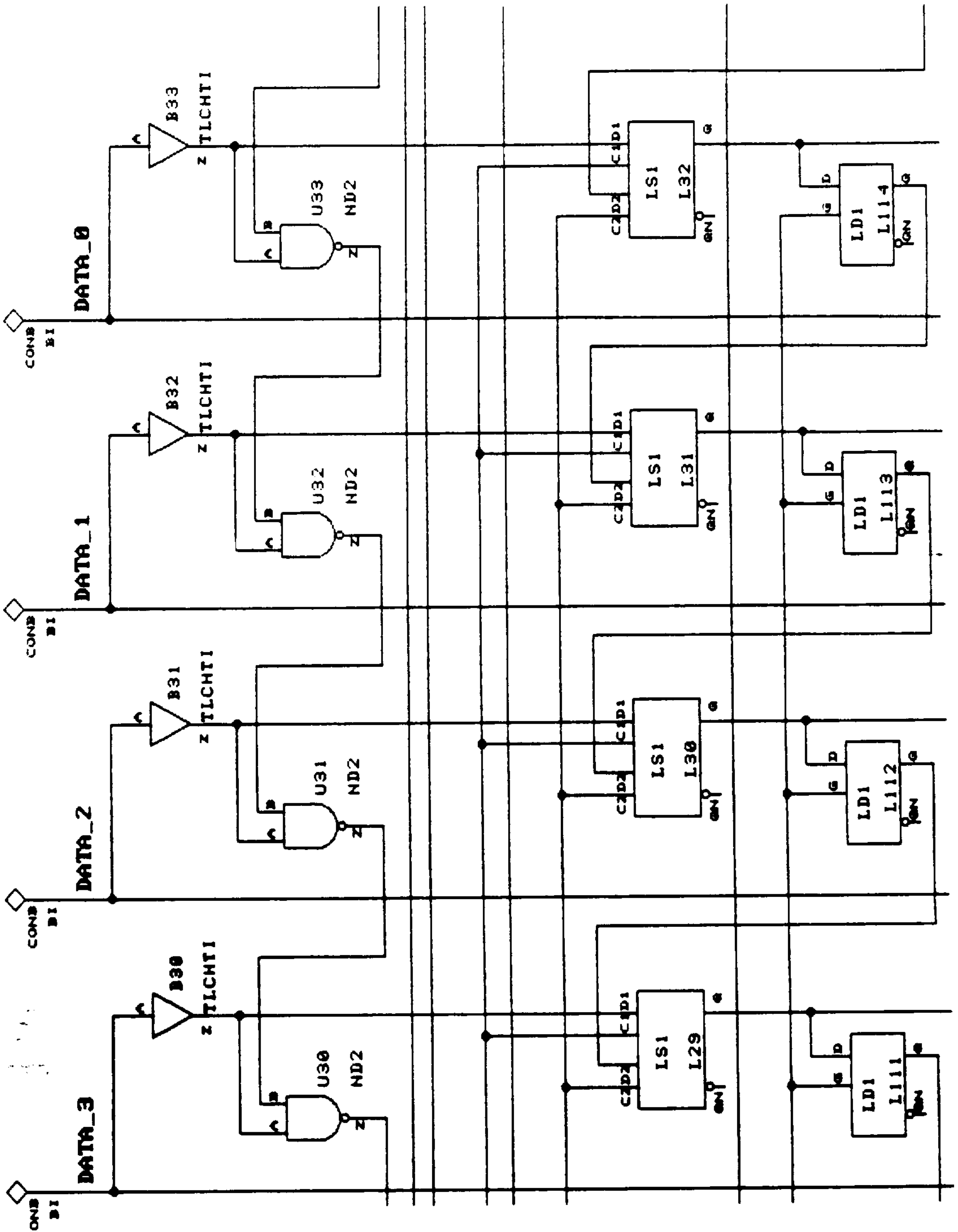

DATA_0
DATA_1
DATA_2
DATA_3
CONB
BI
B33
B32
B31
B30
TLCHTI
U33
U32
U31
U30
ND2
LS1
L32
L31
L30
L29
LD1
L114
L113
L112
L111
C1D1
C2D2
GN
D
G
Q
A
B
Z

Figure 9

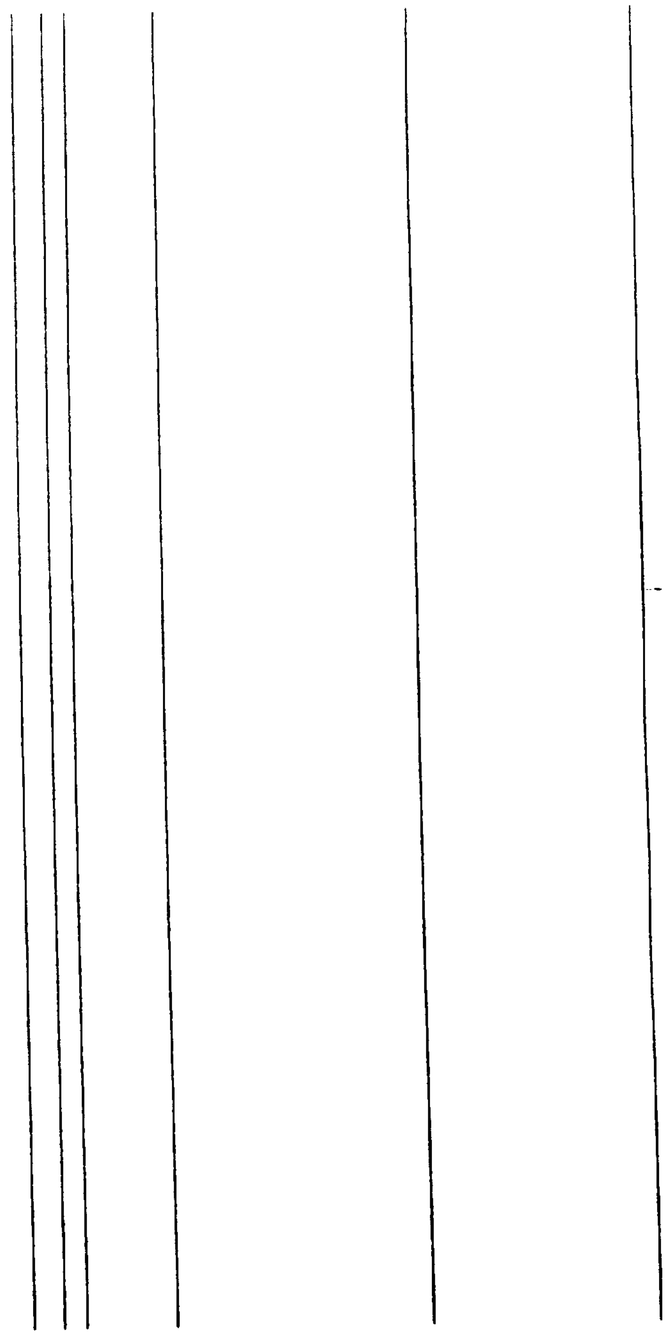


Figure 10

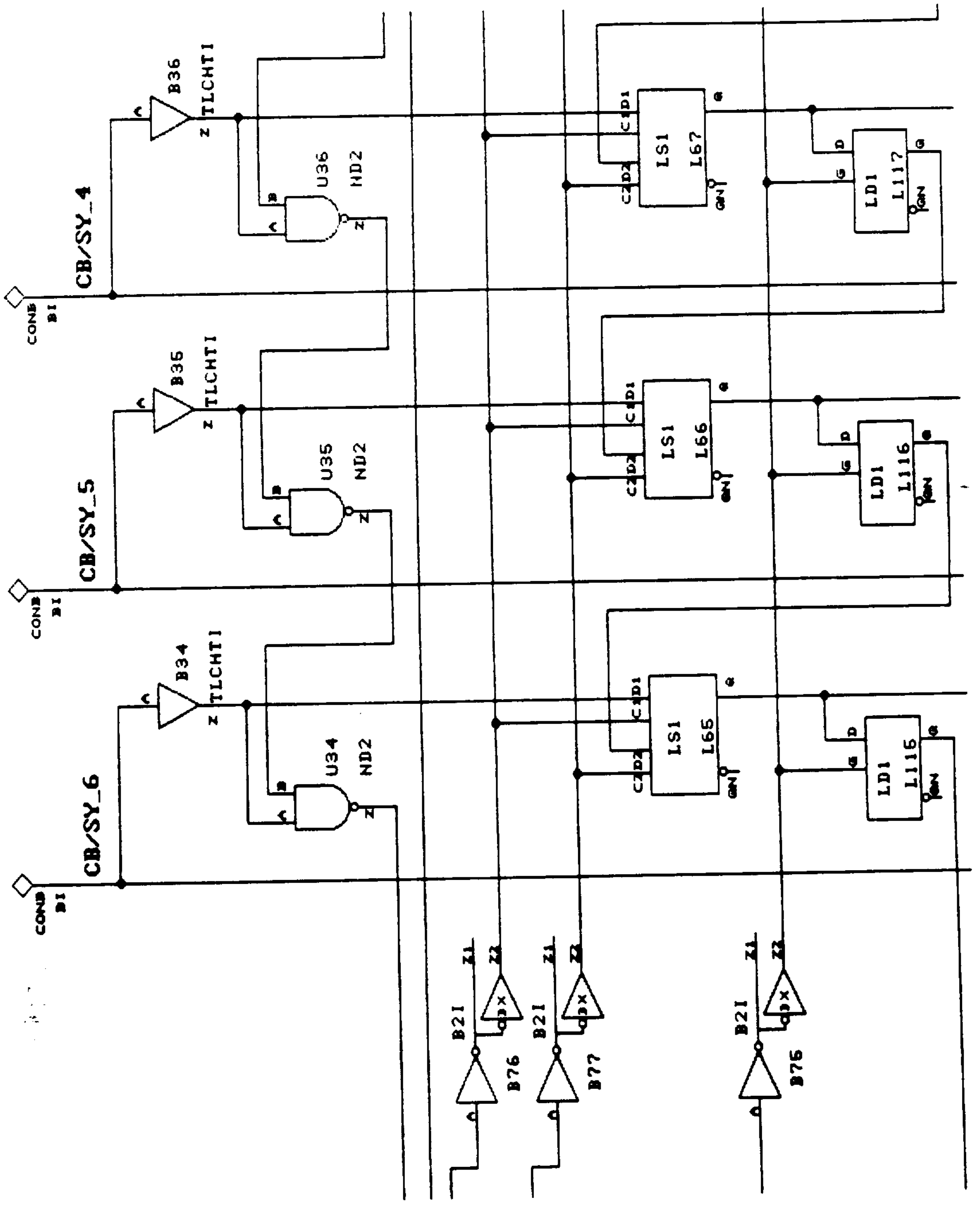

CB/SY_4
CB/SY_5
CB/SY_6
CONB BI
CONB BI
CONB BI
B36 TLCHTI
B35 TLCHTI
B34 TLCHTI
U36 ND2
U35 ND2
U34 ND2
LS1 L67
LS1 L66
LS1 L65
C1D1
C2D2
GN
LD1 L117
LD1 L116
LD1 L115
B21
Z1
Z2
B76
B77
B75

Figure 11

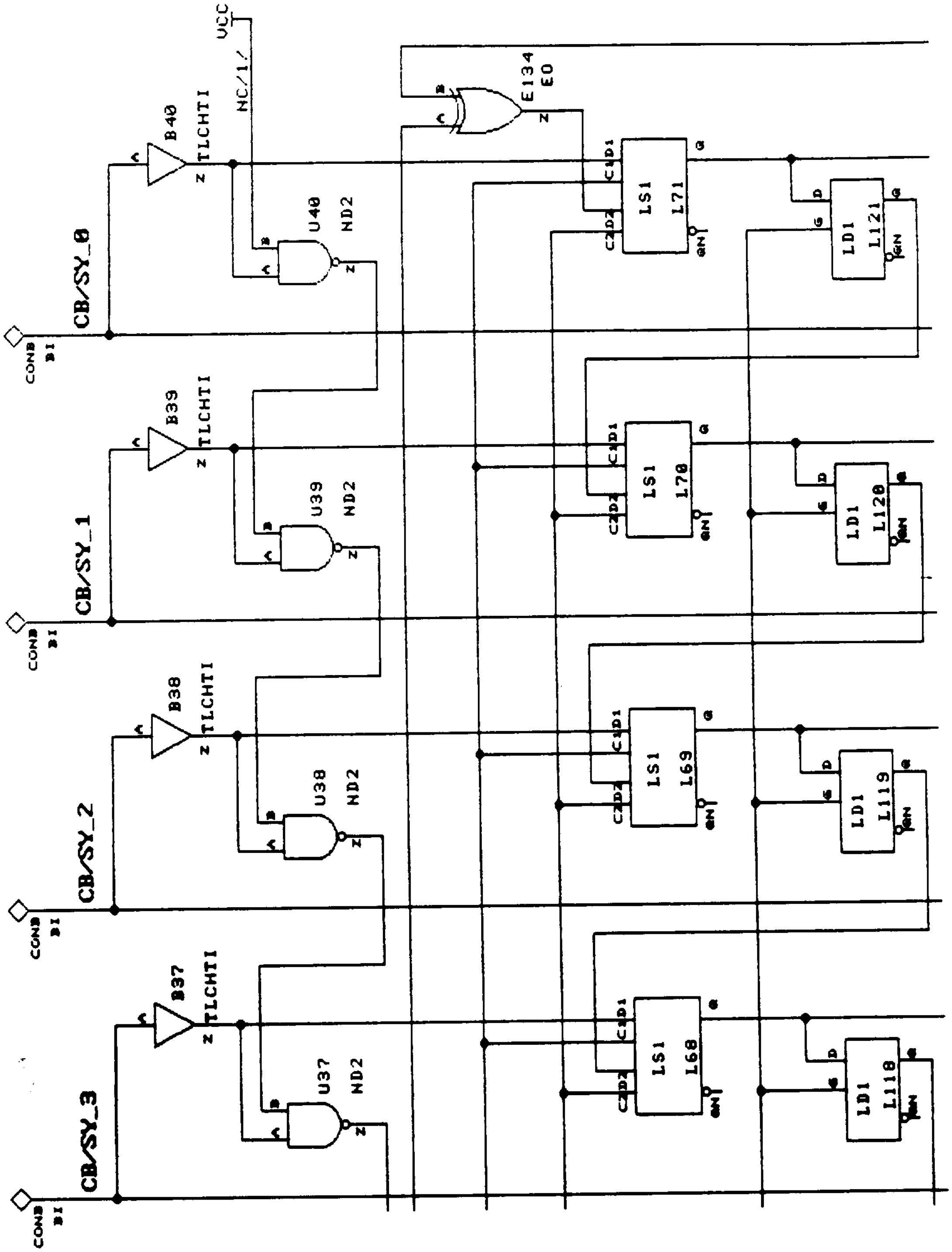

VCC
NC/1/
B40
TLCHTI
B39
TLCHTI
B38
TLCHTI
B37
TLCHTI
U40
ND2
U39
ND2
U38
ND2
U37
ND2
E134
EO
CB/SY_0
CB/SY_1
CB/SY_2
CB/SY_3
CONB
BI
LS1
L71
L70
L69
L68
LD1
L121
L120
L119
L118

Figure 12

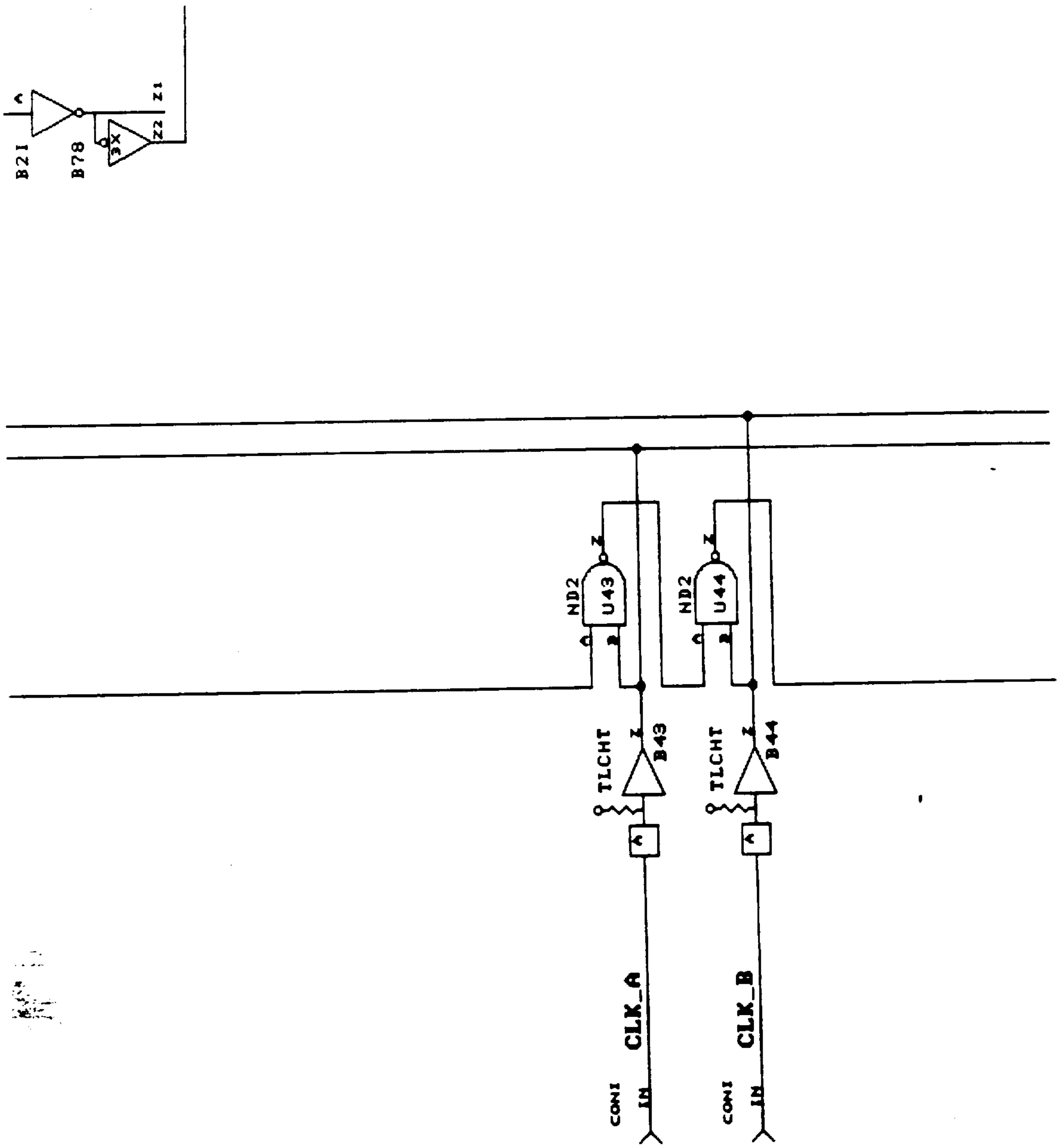

B21
B78
A
Z1
Z2
3X
ND2
U43
ND2
U44
A
B
Z
TLCHT
B43
TLCHT
B44
A
CLK_A
CLK_B
CONI
IN
CONI
IN

Figure 13

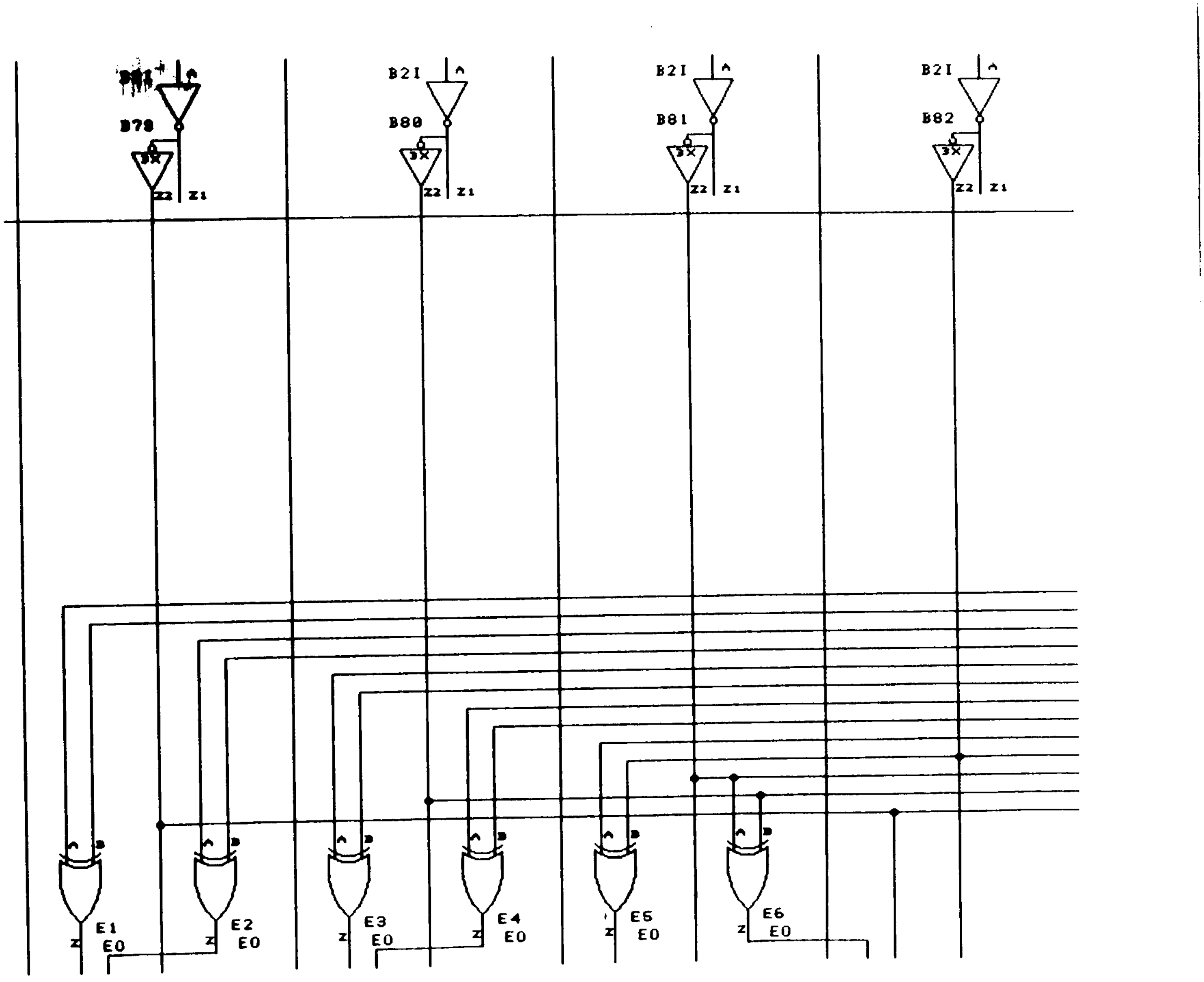

B21
A
B79
3X
Z2
Z1
B21
A
B80
3X
Z2
Z1
B21
A
B81
3X
Z2
Z1
B21
A
B82
3X
Z2
Z1
A
B
E1
Z
E0
A
B
E2
Z
E0
A
B
E3
Z
E0
A
B
E4
Z
E0
A
B
E5
Z
E0
A
B
E6
Z
E0

Figure 14

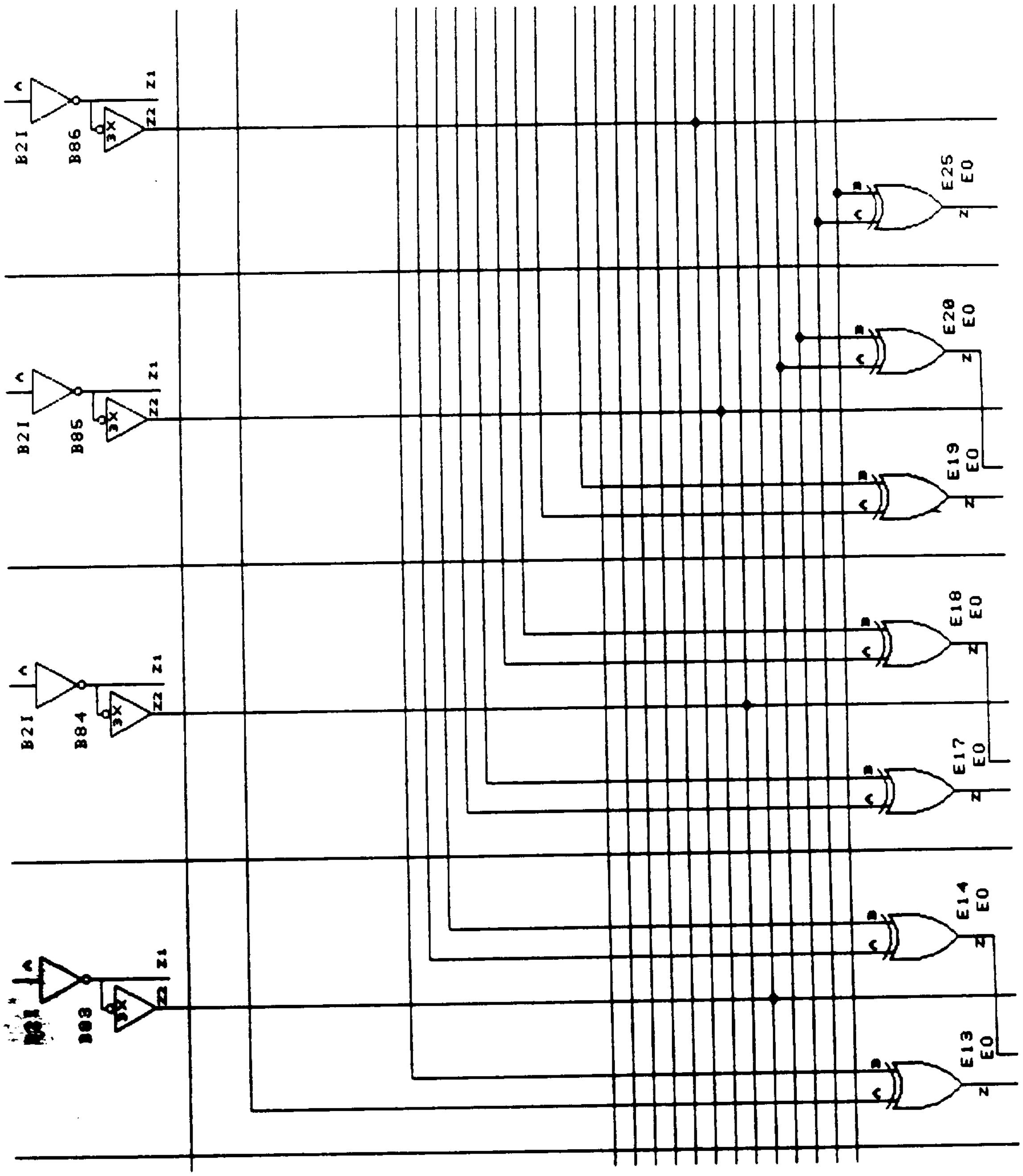

B2I
B86
A
Z1
Z2
3X
B2I
B85
A
Z1
Z2
3X
B2I
B84
A
Z1
Z2
3X
B2I
B83
A
Z1
Z2
3X
E25
EO
E20
EO
E19
EO
E18
EO
E17
EO
E14
EO
E13
EO
A
B
Z

Figure 15

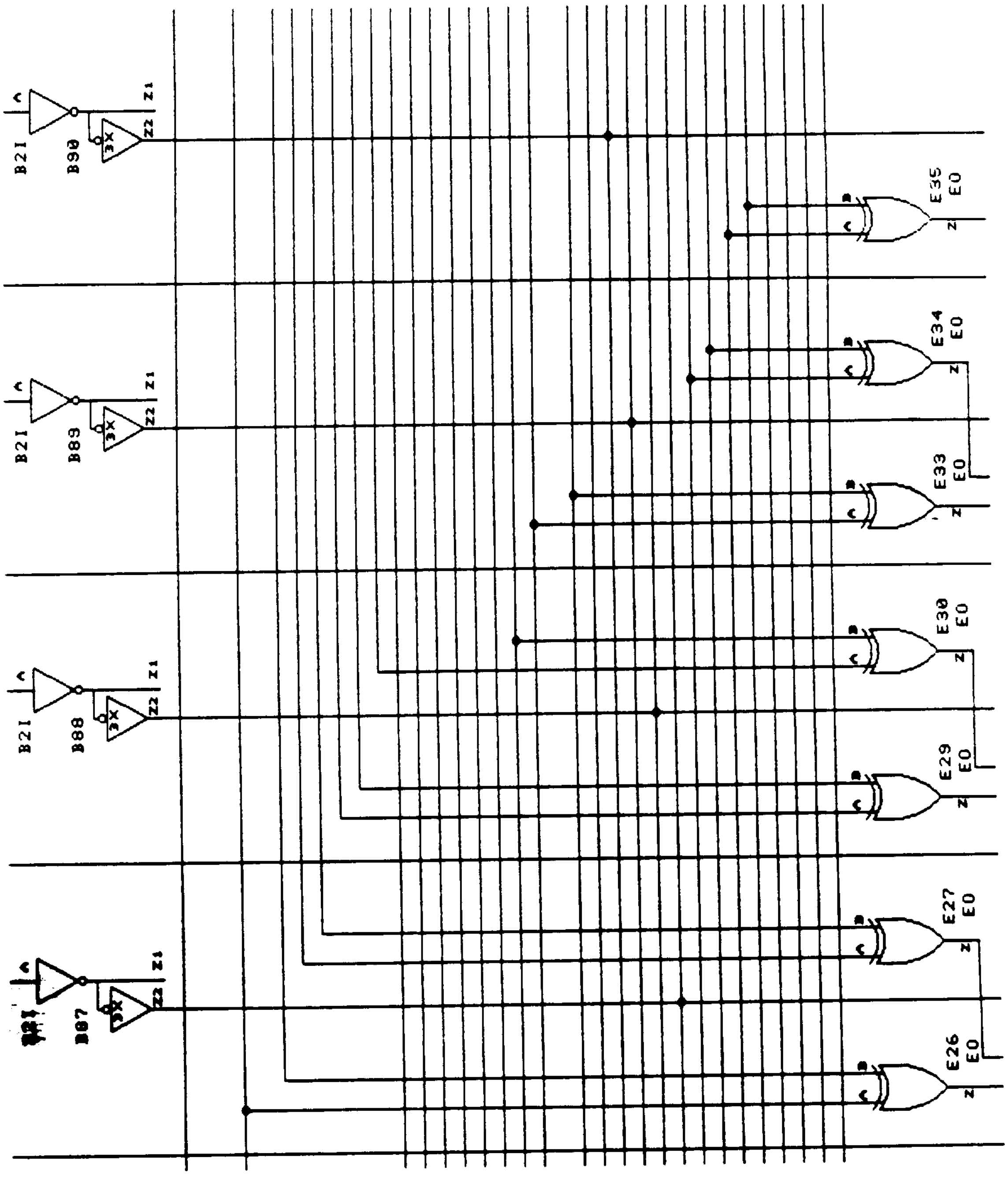

B2I
B90
B89
B88
B87
A
Z1
Z2
3X
A
B
Z
E35
EO
E34
EO
E33
EO
E30
EO
E29
EO
E27
EO
E26
EO

Figure 16

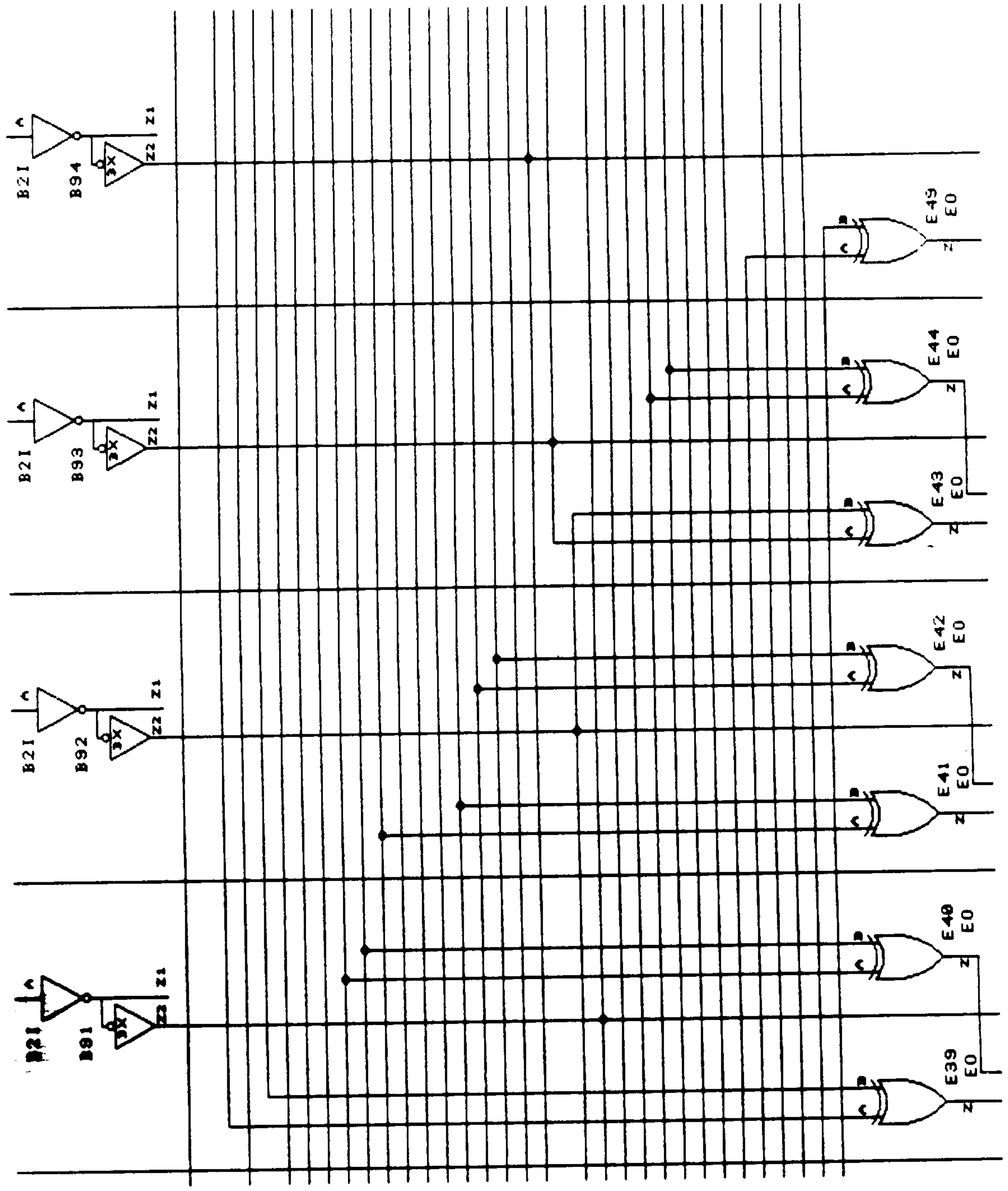

B2I
B94
A
Z1
Z2
3X
B2I
B93
B2I
B92
B2I
B91
E49
E0
E44
E0
E43
E0
E42
E0
E41
E0
E40
E0
E39
E0
A
B
Z

Figure 17

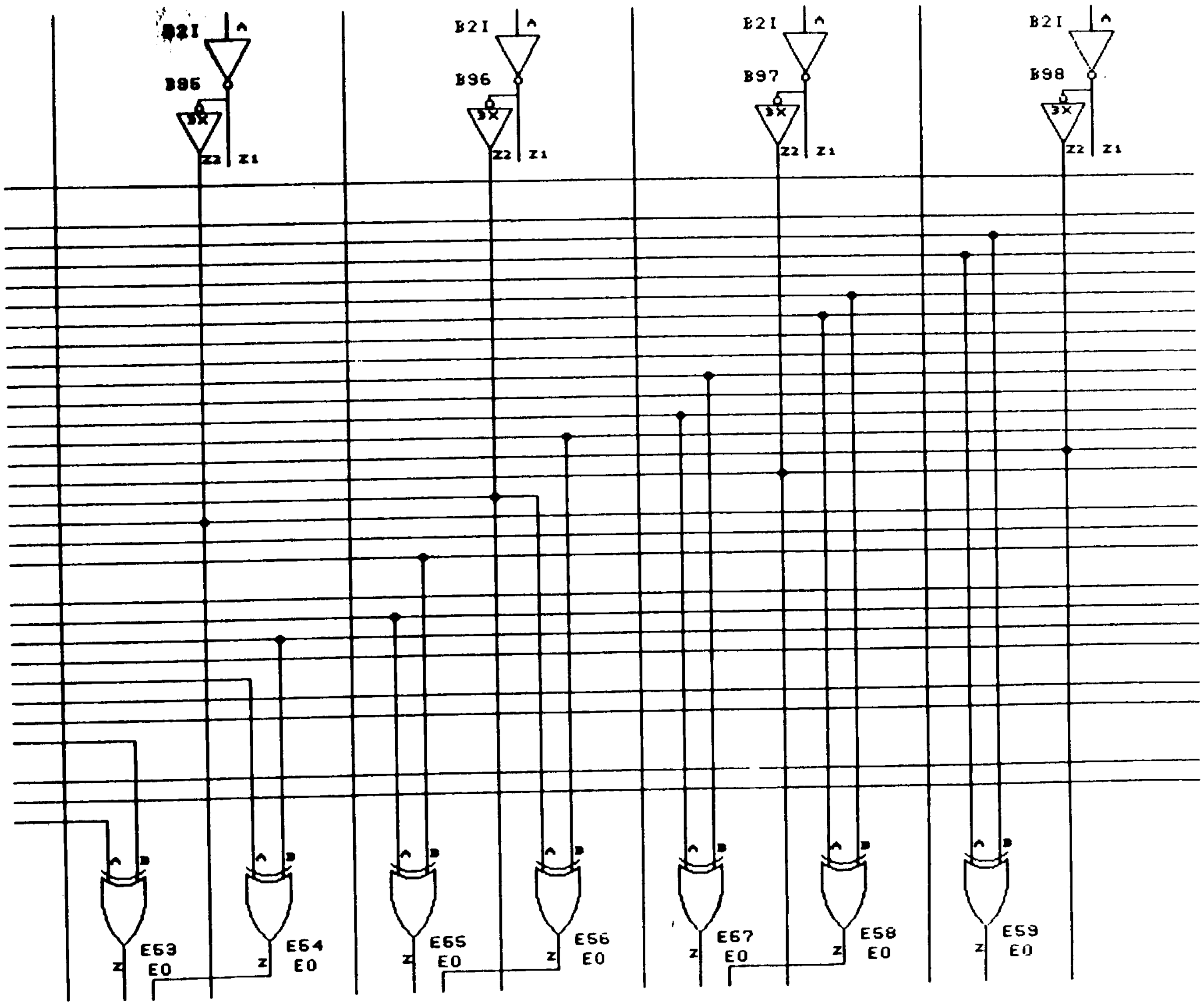

B2I
A
B95
3X
Z2
Z1
B2I
A
B96
3X
Z2
Z1
B2I
A
B97
3X
Z2
Z1
B2I
A
B98
3X
Z2
Z1
A
B
Z
E63
E0
A
B
Z
E64
E0
A
B
Z
E65
E0
A
B
Z
E56
E0
A
B
Z
E67
E0
A
B
Z
E58
E0
A
B
Z
E59
E0

Figure 18

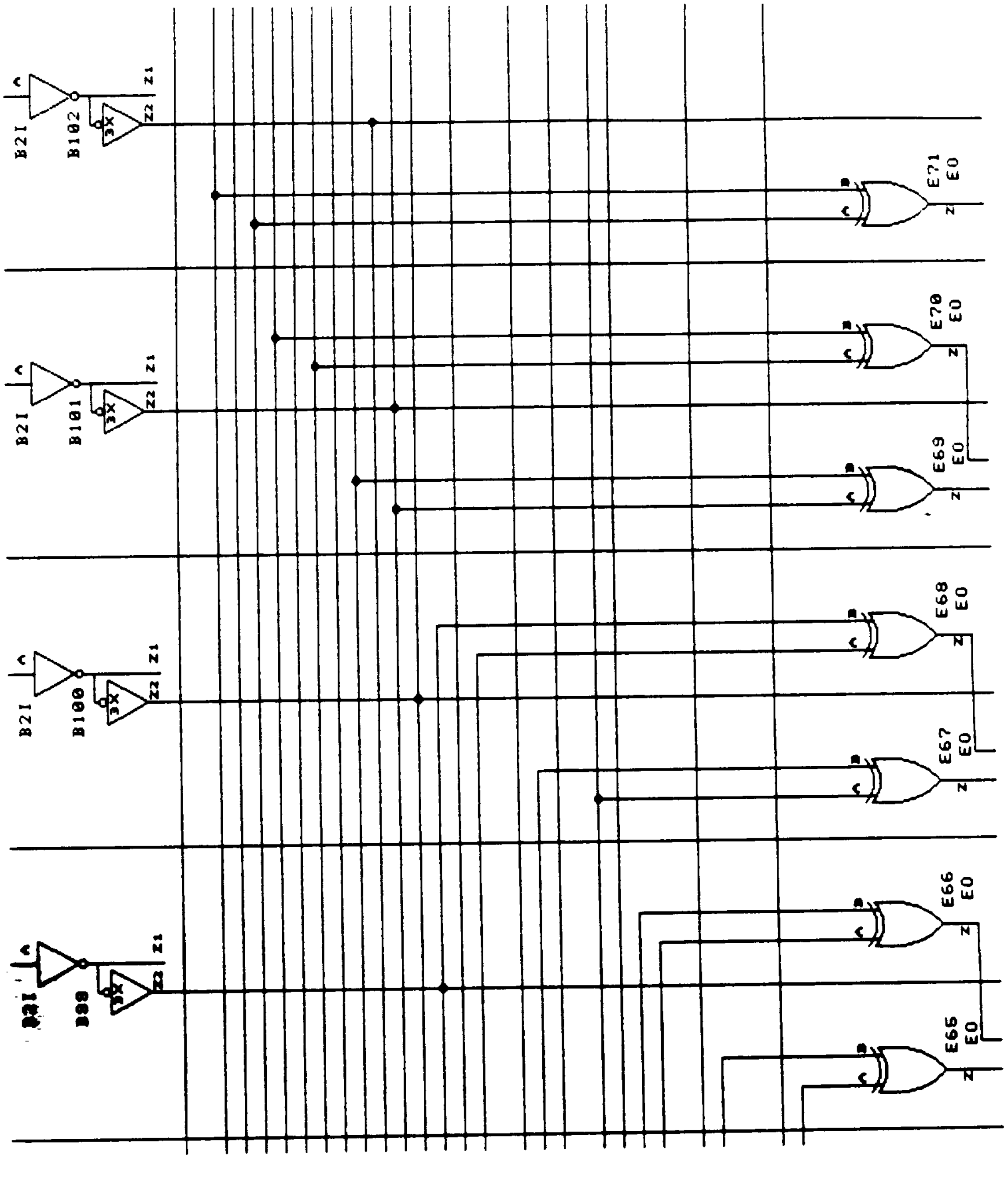

B2I
B102
A
Z1
Z2
3X
B2I
B101
A
Z1
Z2
3X
B2I
B100
A
Z1
Z2
3X
B2I
B98
A
Z1
Z2
3X
E71
E0
E70
E0
E69
E0
E68
E0
E67
E0
E66
E0
E65
E0
B
C
Z

Figure 19

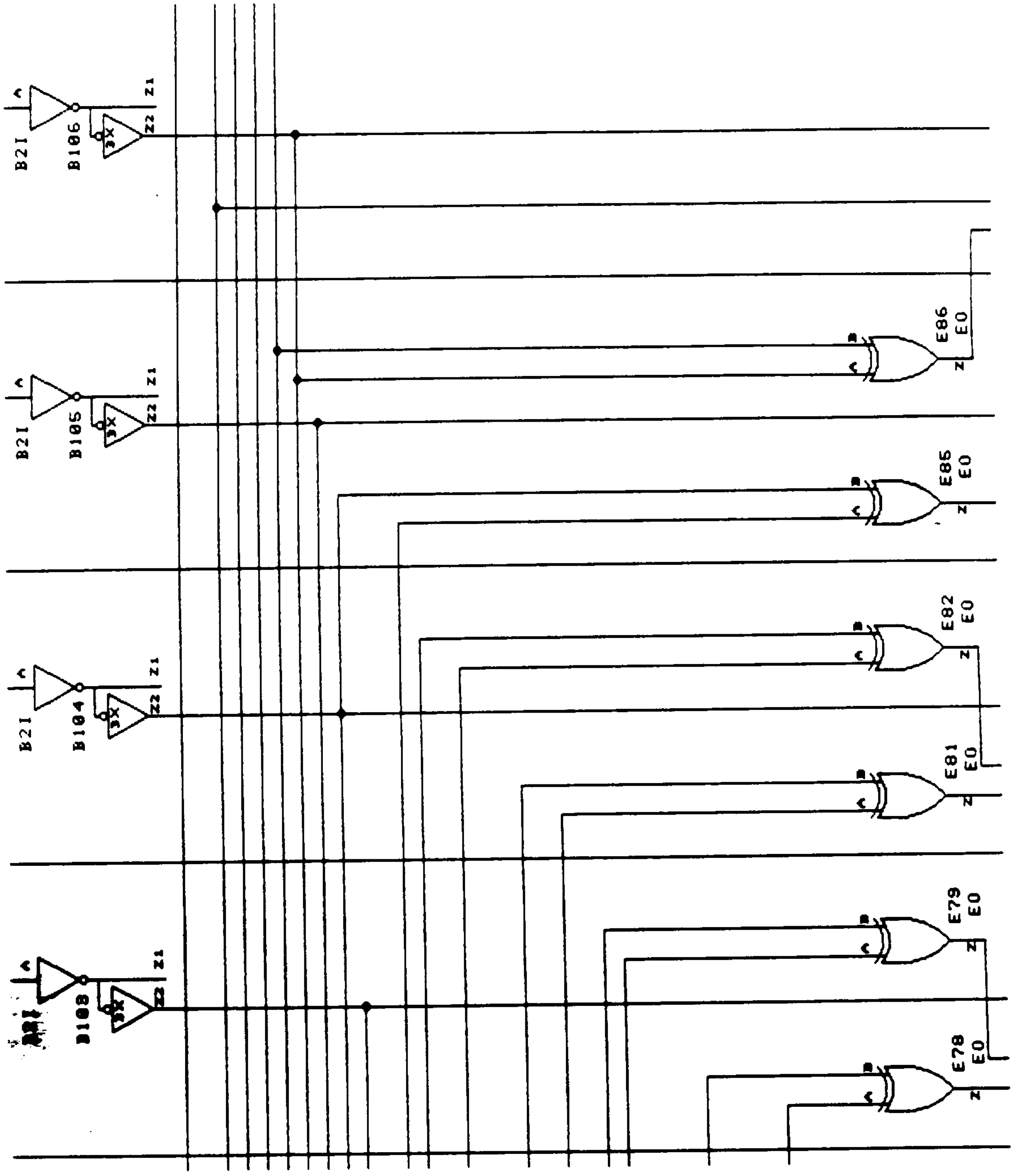

B21
B106
A
Z1
Z2
3X
B21
B105
A
Z1
Z2
3X
B21
B104
A
Z1
Z2
3X
B21
B108
A
Z1
Z2
3X
E86
E0
E85
E0
E82
E0
E81
E0
E79
E0
E78
E0
A
B
Z

Figure 20

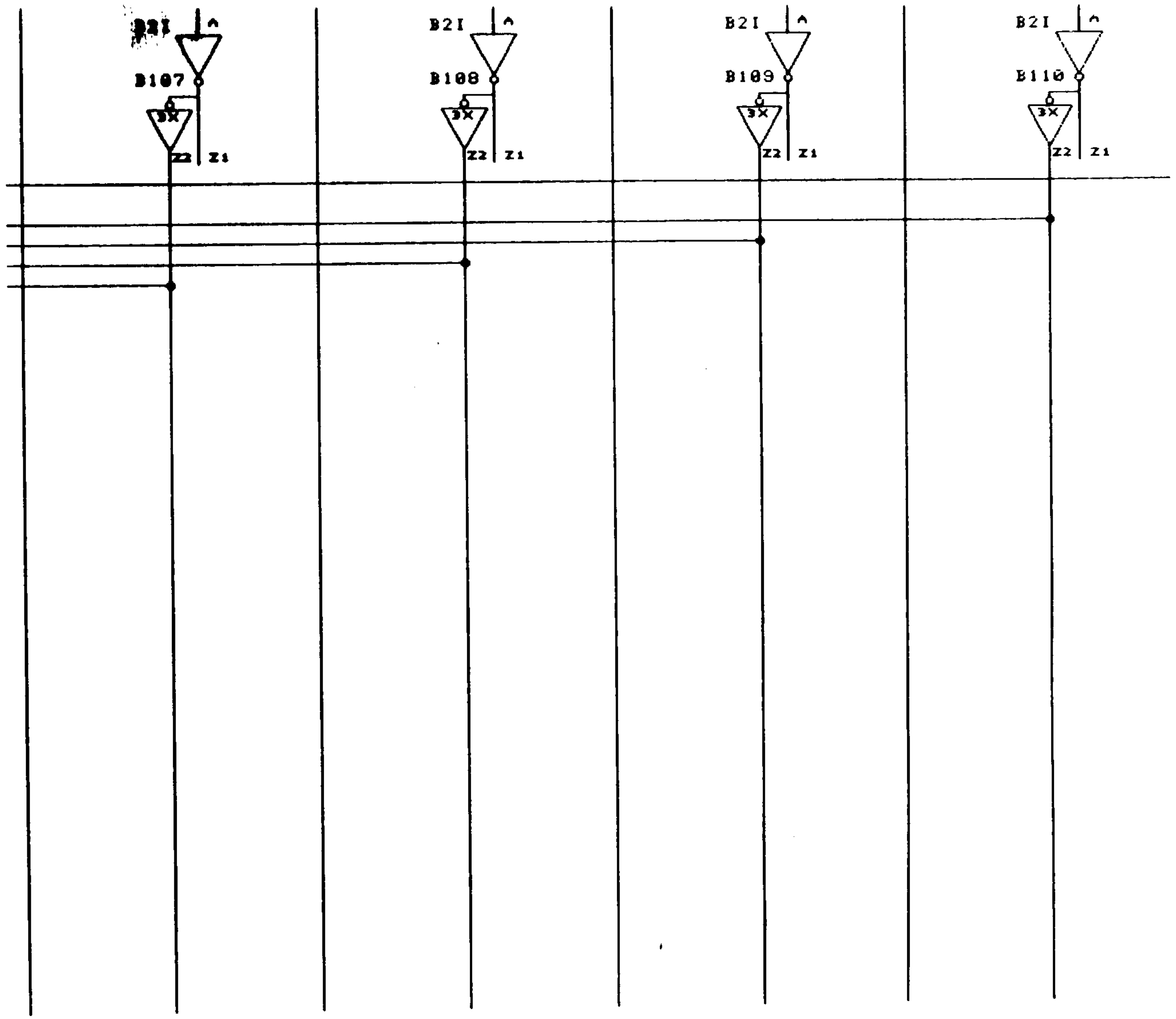

B21
B107
3X
Z2
Z1
B21
B108
3X
Z2
Z1
B21
B109
3X
Z2
Z1
B21
B110
3X
Z2
Z1

Figure 21

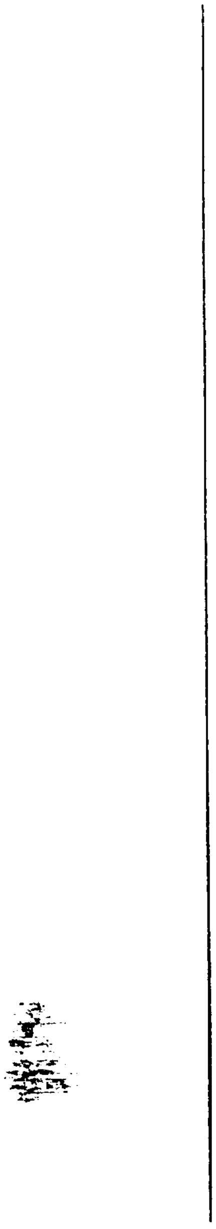


Figure 22

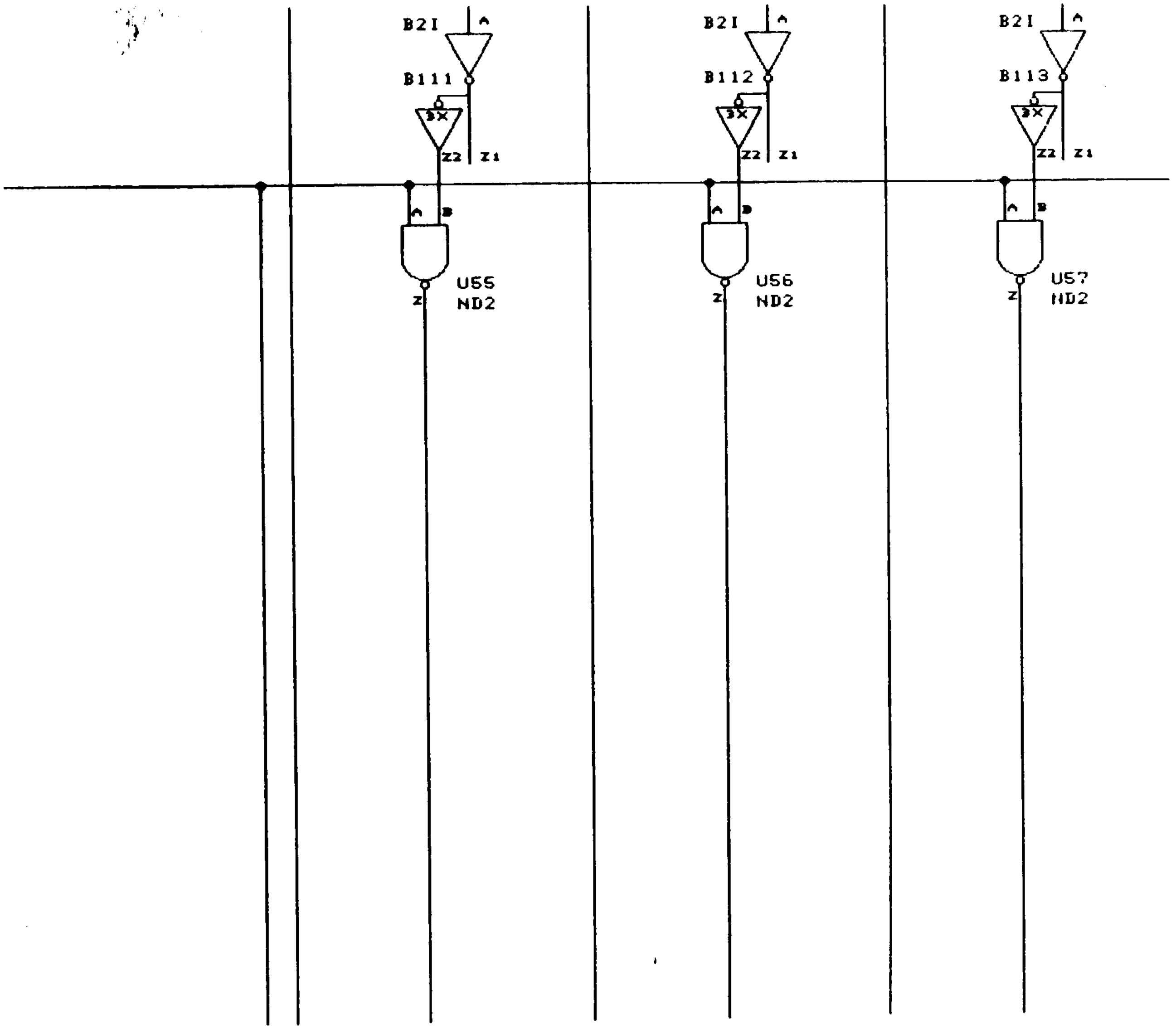
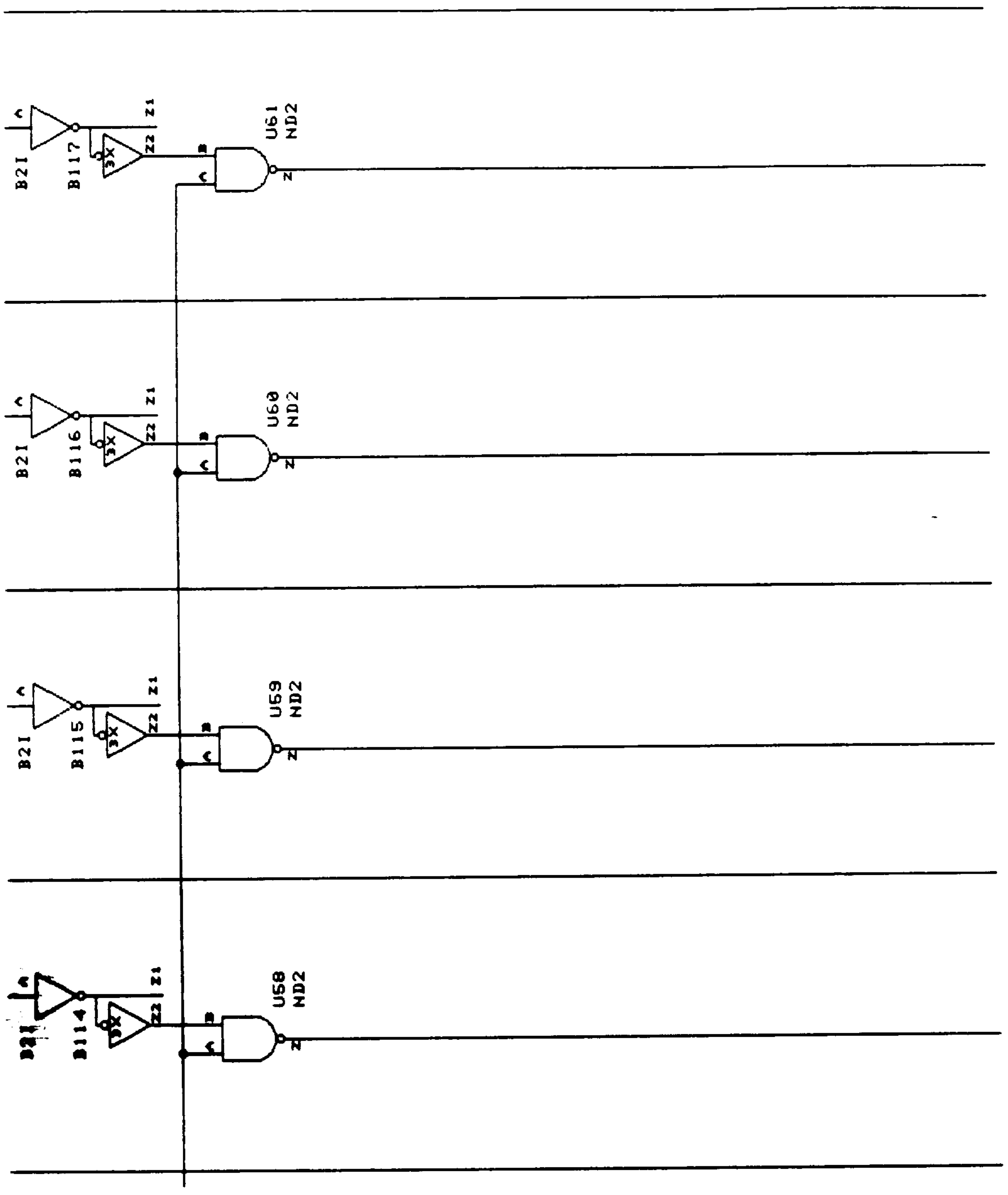

B2I
A
B111
3X
Z2
Z1
A
B
U55
ND2
Z
B2I
A
B112
3X
Z2
Z1
A
B
U56
ND2
Z
B2I
A
B113
3X
Z2
Z1
A
B
U57
ND2
Z

Figure 23



B2I
B117
U61
ND2
B2I
B116
U60
ND2
B2I
B115
U59
ND2
B2I
B114
U58
ND2
A
B
Z
Z1
Z2
3X

Figure 74

Figure 25

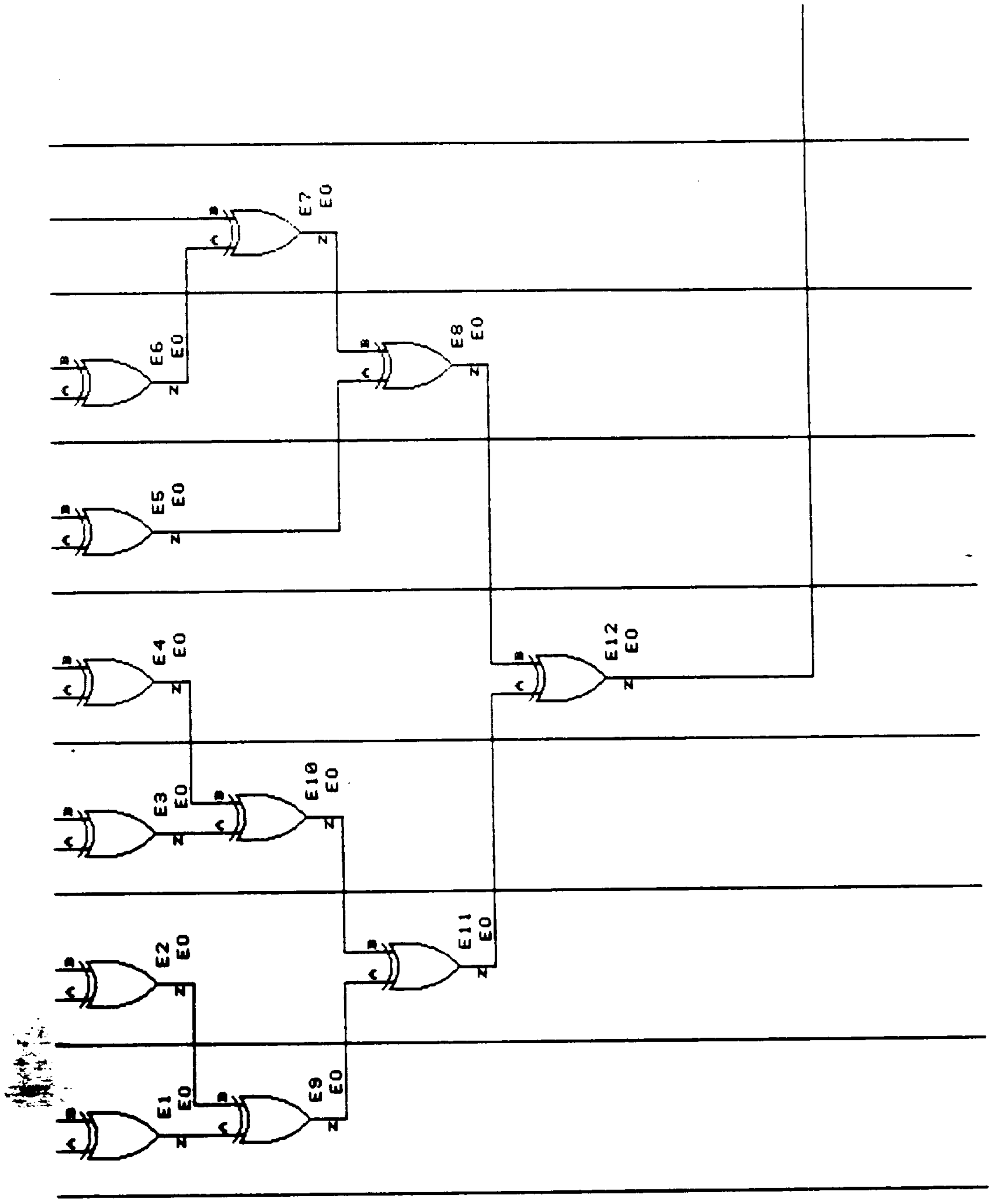

E1
E0
E2
E0
E3
E0
E4
E0
E5
E0
E6
E0
E7
E0
E8
E0
E9
E0
E10
E0
E11
E0
E12
E0
B
C
Z

Figure 26

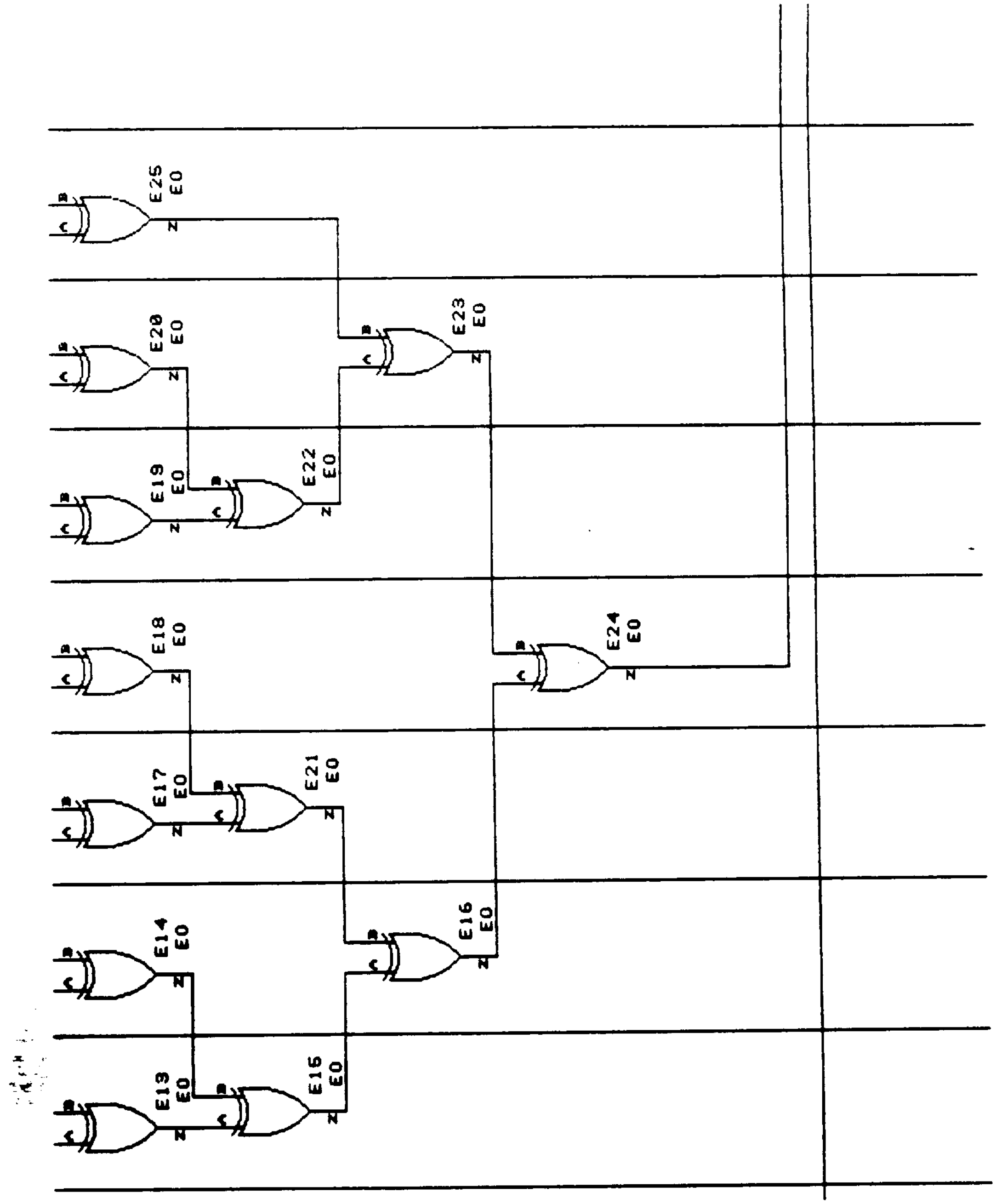

E25
E0
E20
E0
E19
E0
E22
E0
E23
E0
E18
E0
E24
E0
E17
E0
E21
E0
E14
E0
E13
E0
E15
E0
E16
E0
B
C
Z

Figure 27

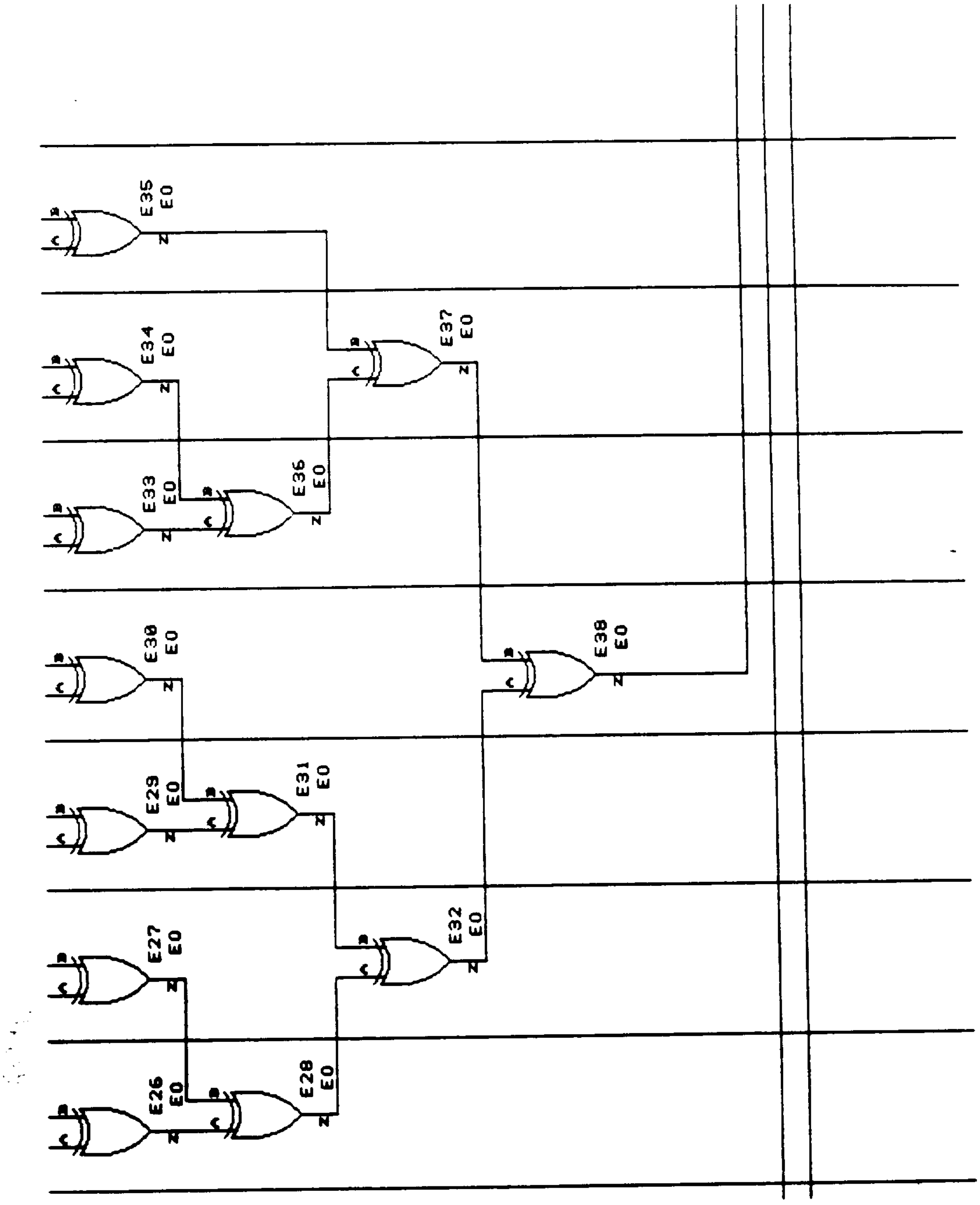
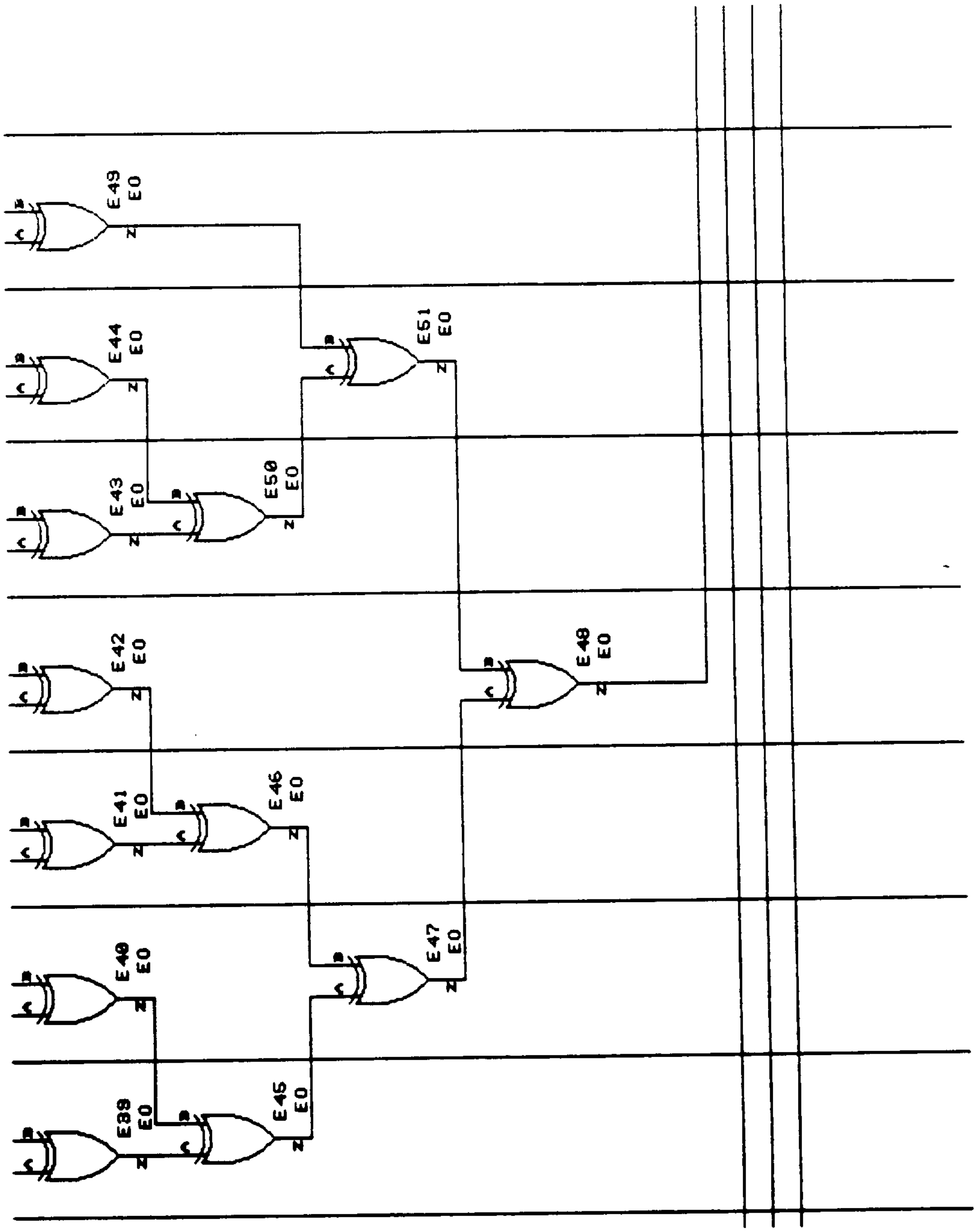

E35
E0
E34
E0
E37
E0
E33
E0
E36
E0
E30
E0
E38
E0
E29
E0
E31
E0
E27
E0
E32
E0
E26
E0
E28
E0
A
C
Z

Figure 28



E49
E0
E44
E0
E43
E0
E42
E0
E41
E0
E40
E0
E39
E0
E51
E0
E50
E0
E46
E0
E45
E0
E47
E0
E48
E0
A
B
Z

Figure 29

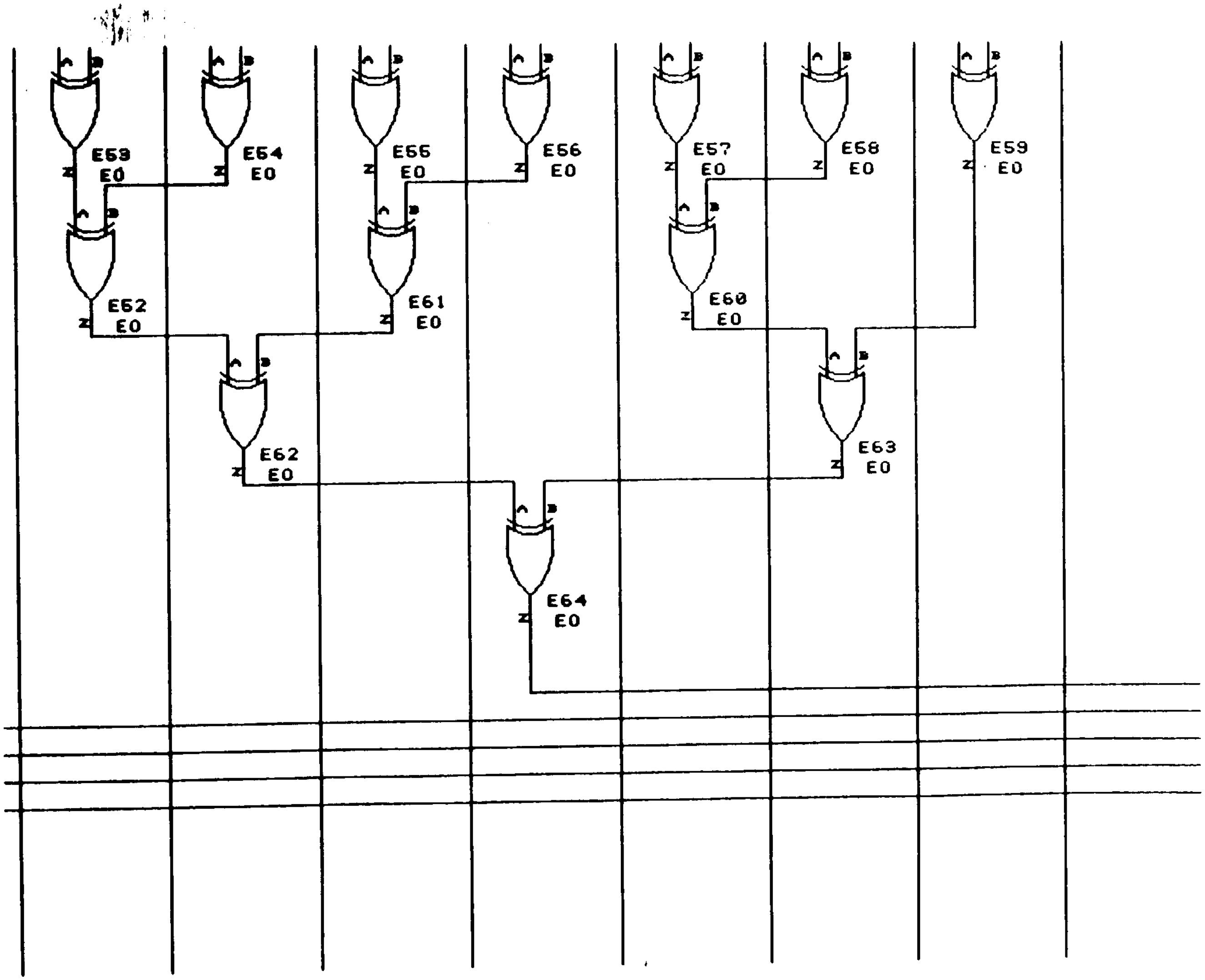

E53
E0
E54
E0
E55
E0
E56
E0
E57
E0
E58
E0
E59
E0
E52
E0
E61
E0
E60
E0
E62
E0
E63
E0
E64
E0
A
B
Z

Figure 30

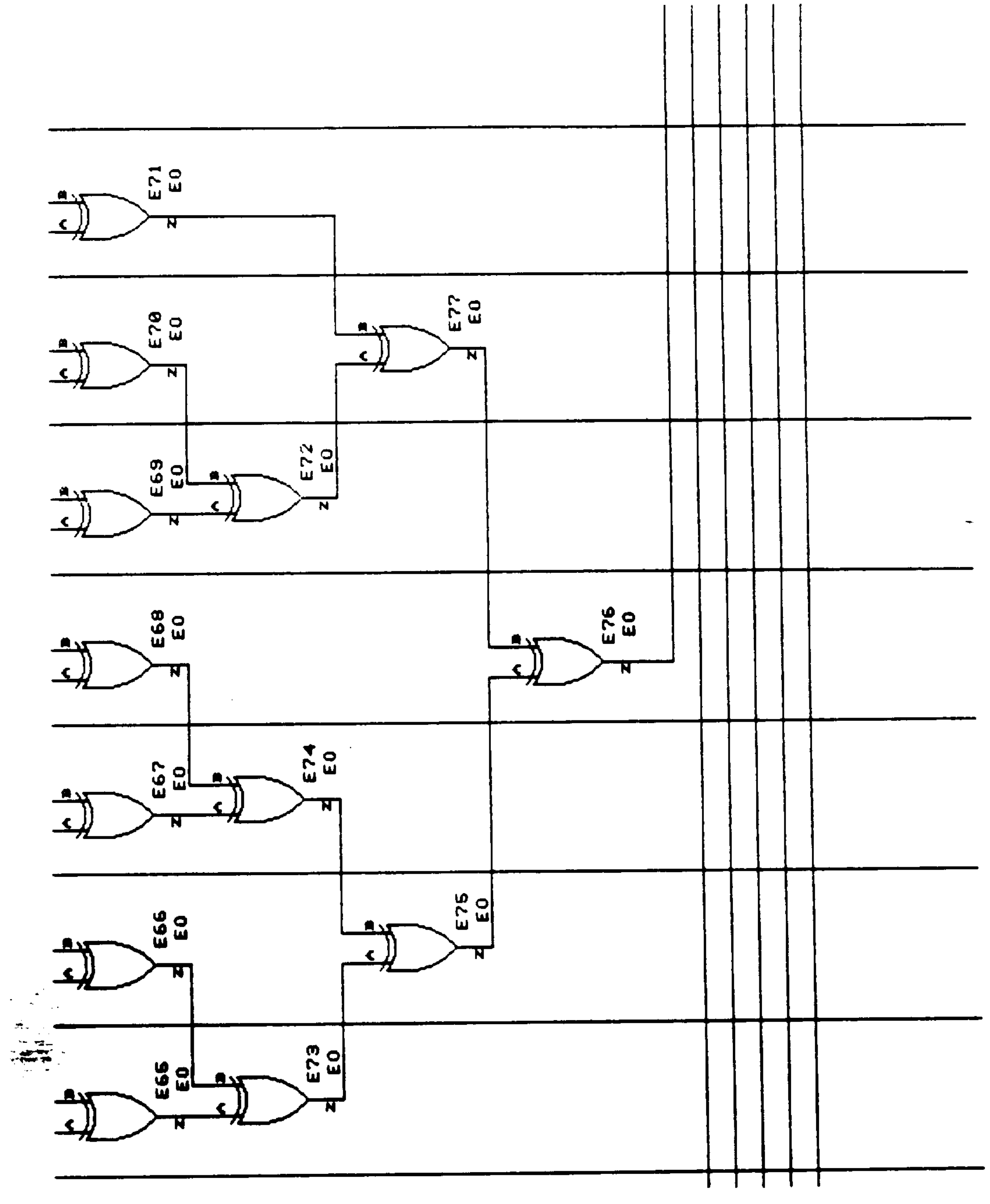

E71
E0
E70
E0
E69
E0
E72
E0
E77
E0
E68
E0
E76
E0
E67
E0
E74
E0
E66
E0
E75
E0
E66
E0
E73
E0
B
C
Z

Figure 31

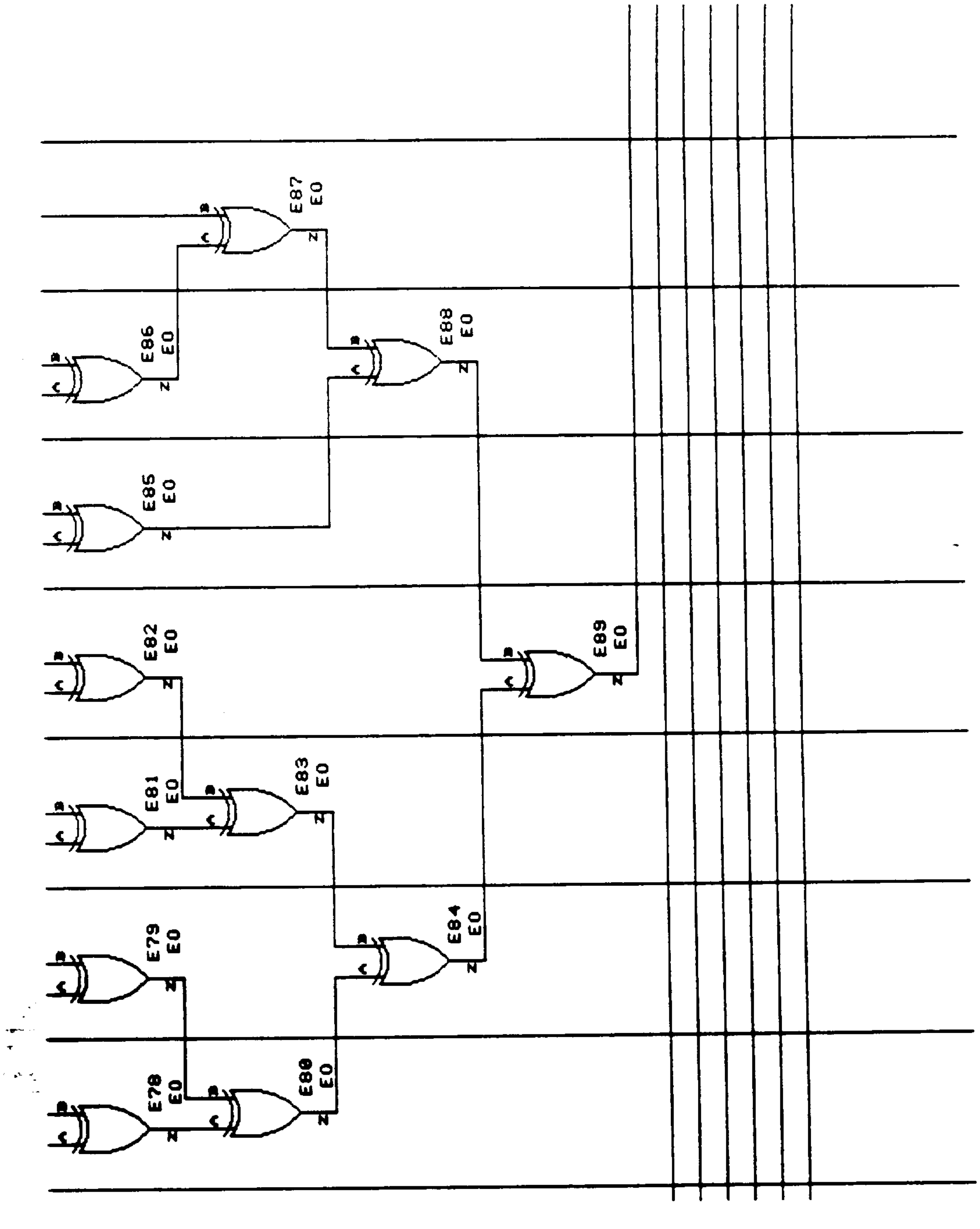

E87
E0
E86
E0
E88
E0
E85
E0
E82
E0
E89
E0
E81
E0
E83
E0
E79
E0
E84
E0
E78
E0
E80
E0
B
C
Z

Figure 32

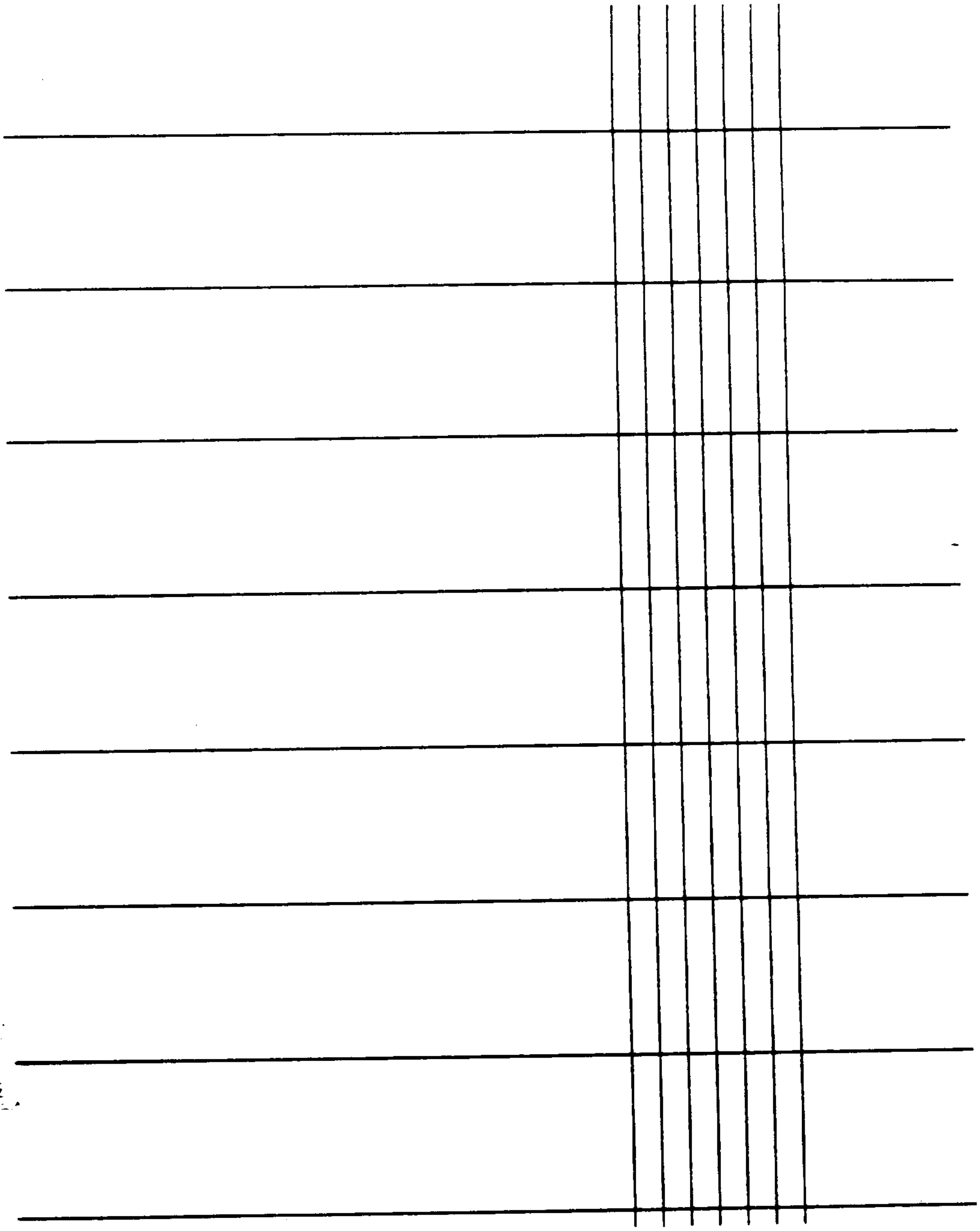


Figure 33

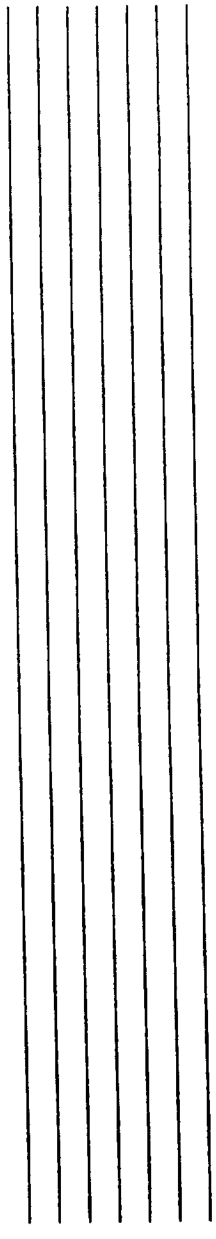


Figure 34

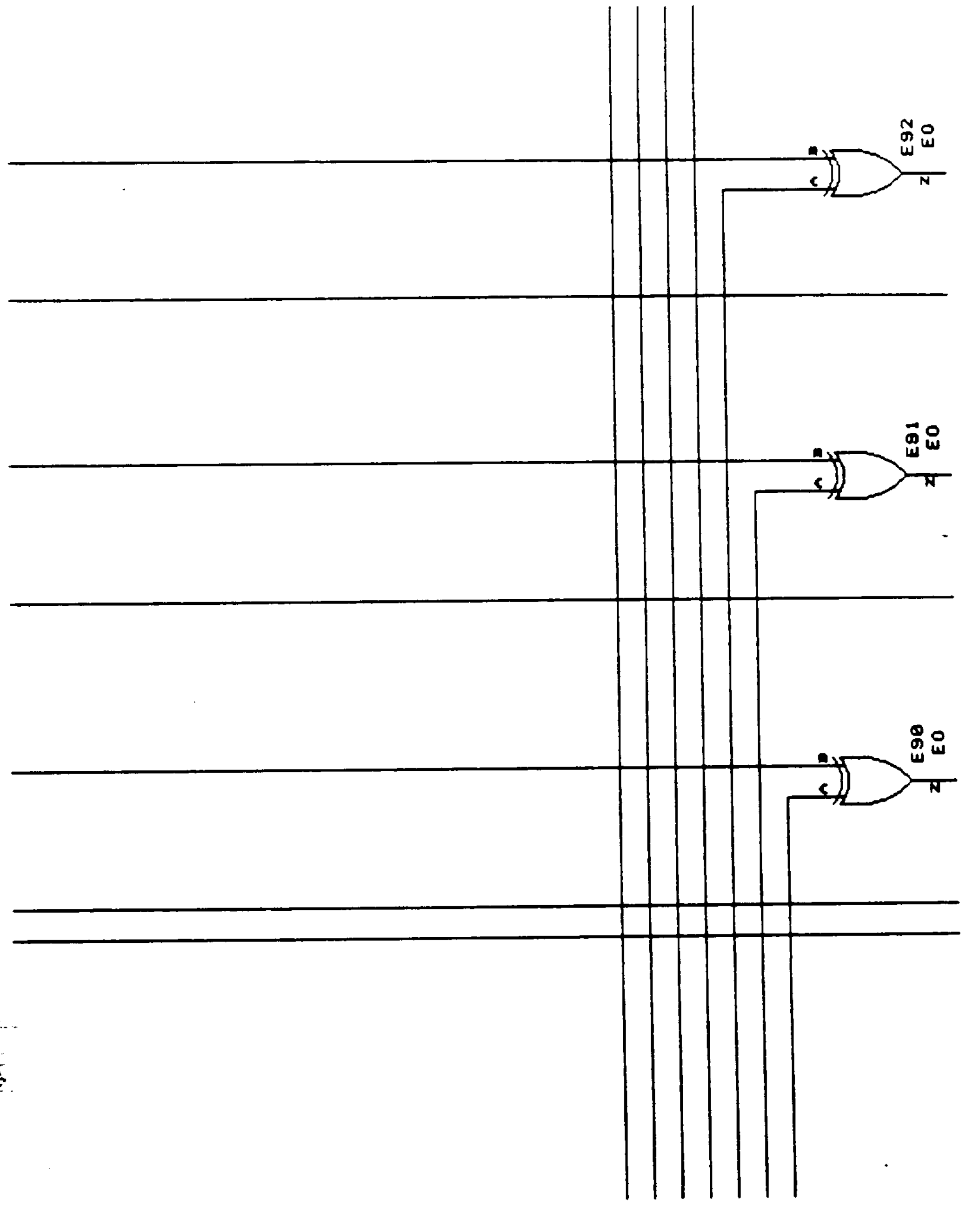


Figure 35

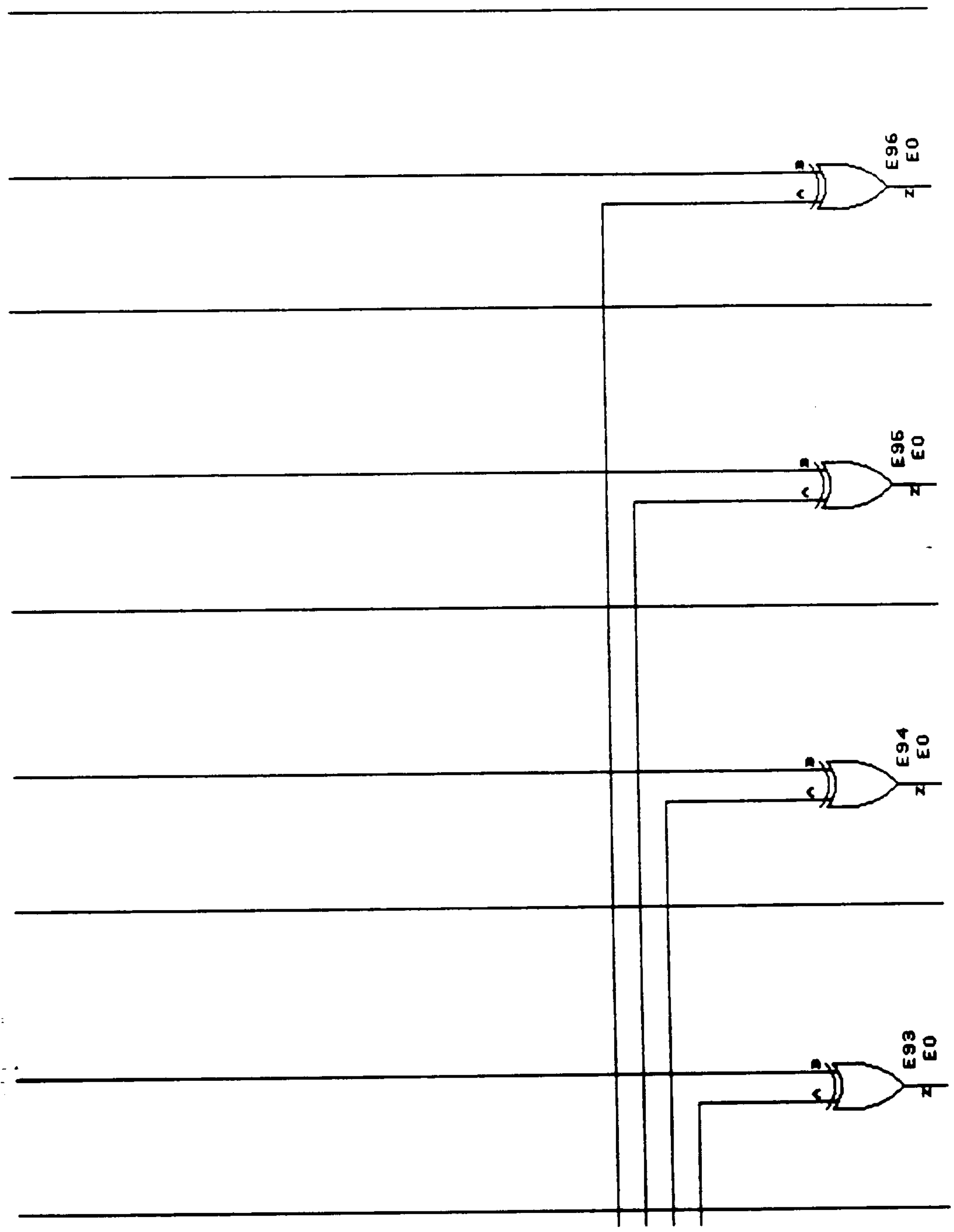


Figure 36

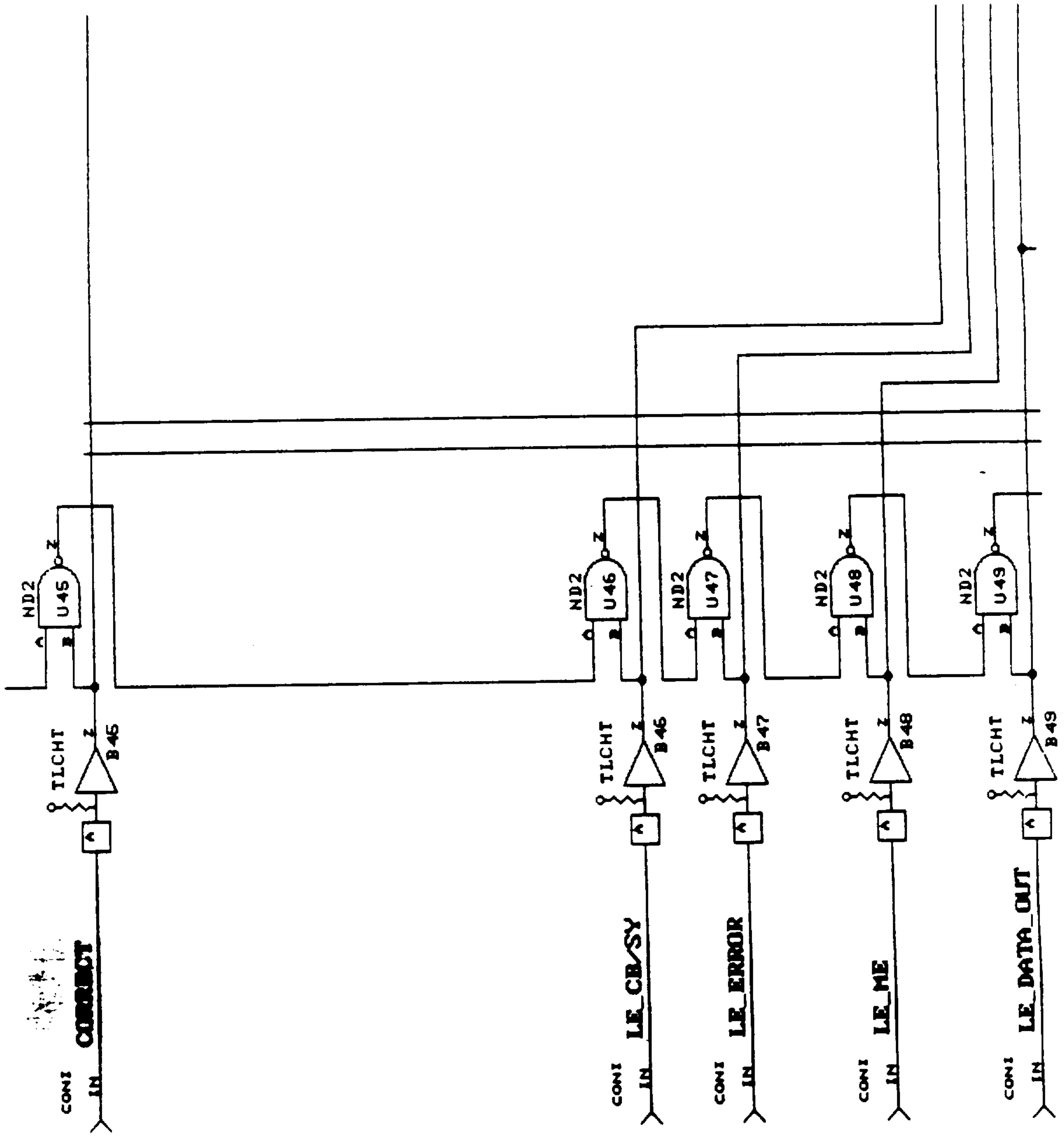

CONI
IN
CORRECT
TLCHT
B45
Z
A
ND2
U45
B
Z
CONI
IN
LE_CB/SY
TLCHT
B46
ND2
U46
CONI
IN
LE_ERROR
TLCHT
B47
ND2
U47
CONI
IN
LE_ME
TLCHT
B48
ND2
U48
CONI
IN
LE_DATA_OUT
TLCHT
B49
ND2
U49

Figure 37

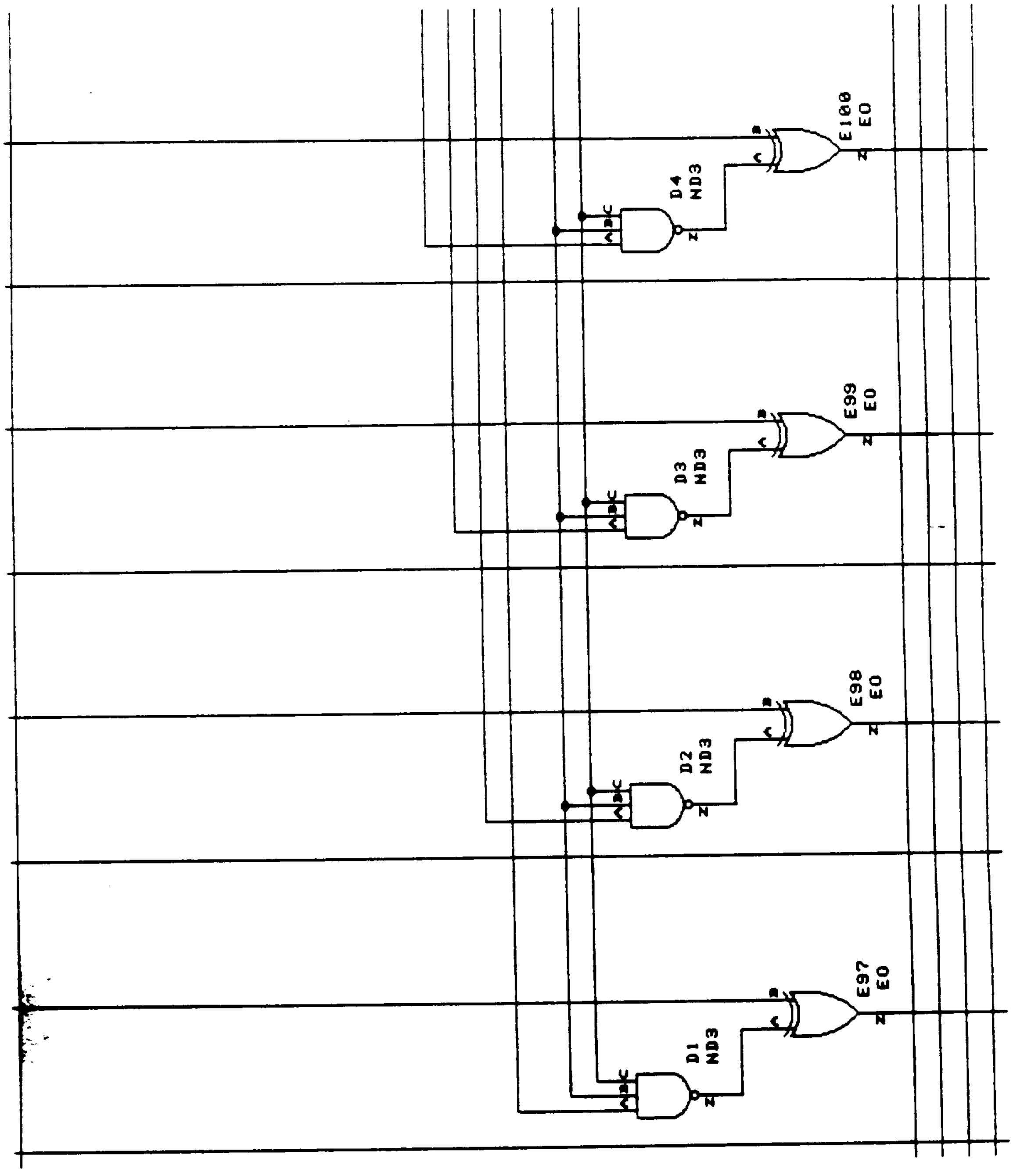


Figure 38

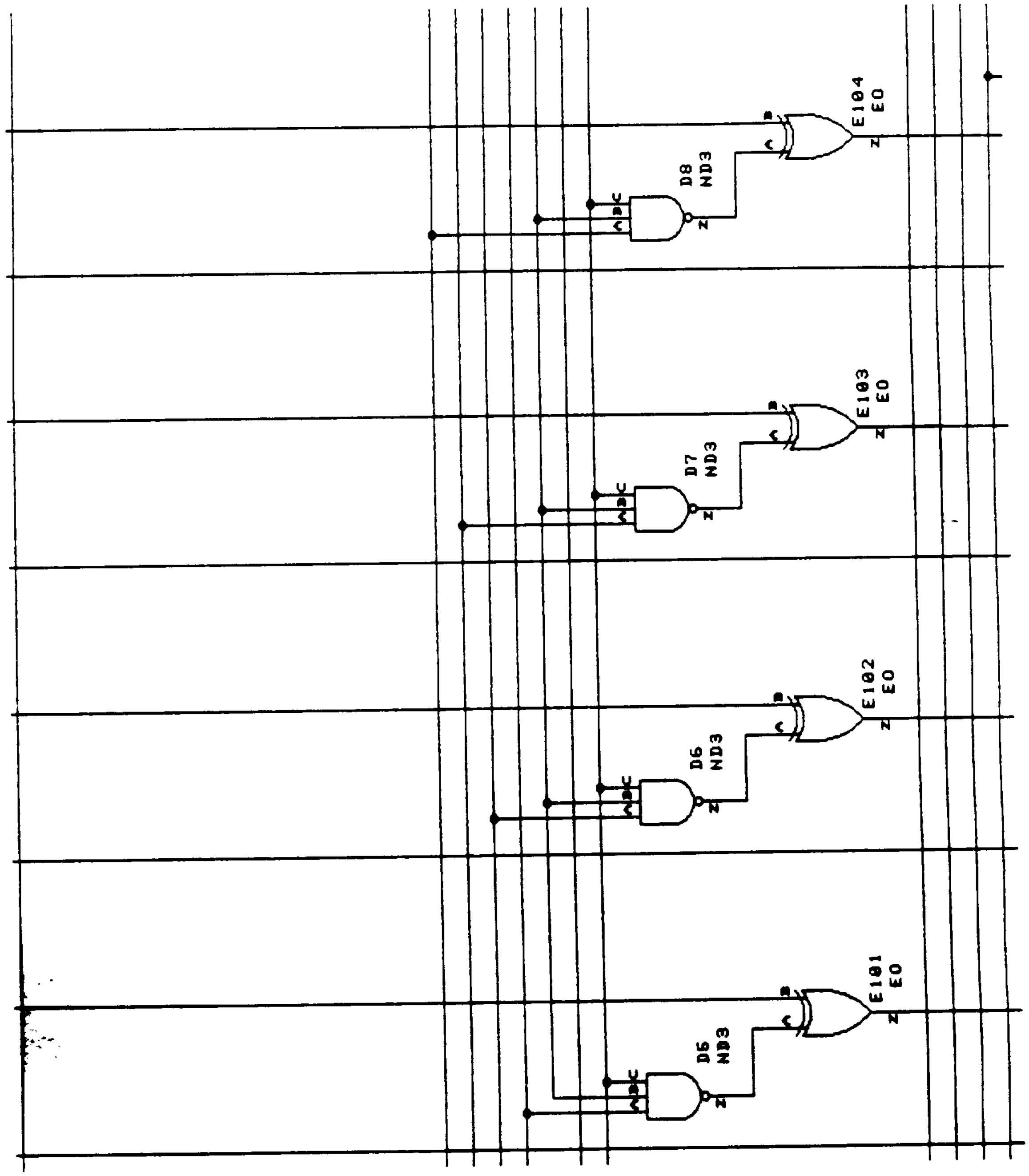

D5
ND3
A
B
C
Z
E101
EO
A
B
Z
D6
ND3
A
B
C
Z
E102
EO
A
B
Z
D7
ND3
A
B
C
Z
E103
EO
A
B
Z
D8
ND3
A
B
C
Z
E104
EO
A
B
Z

Figure 39

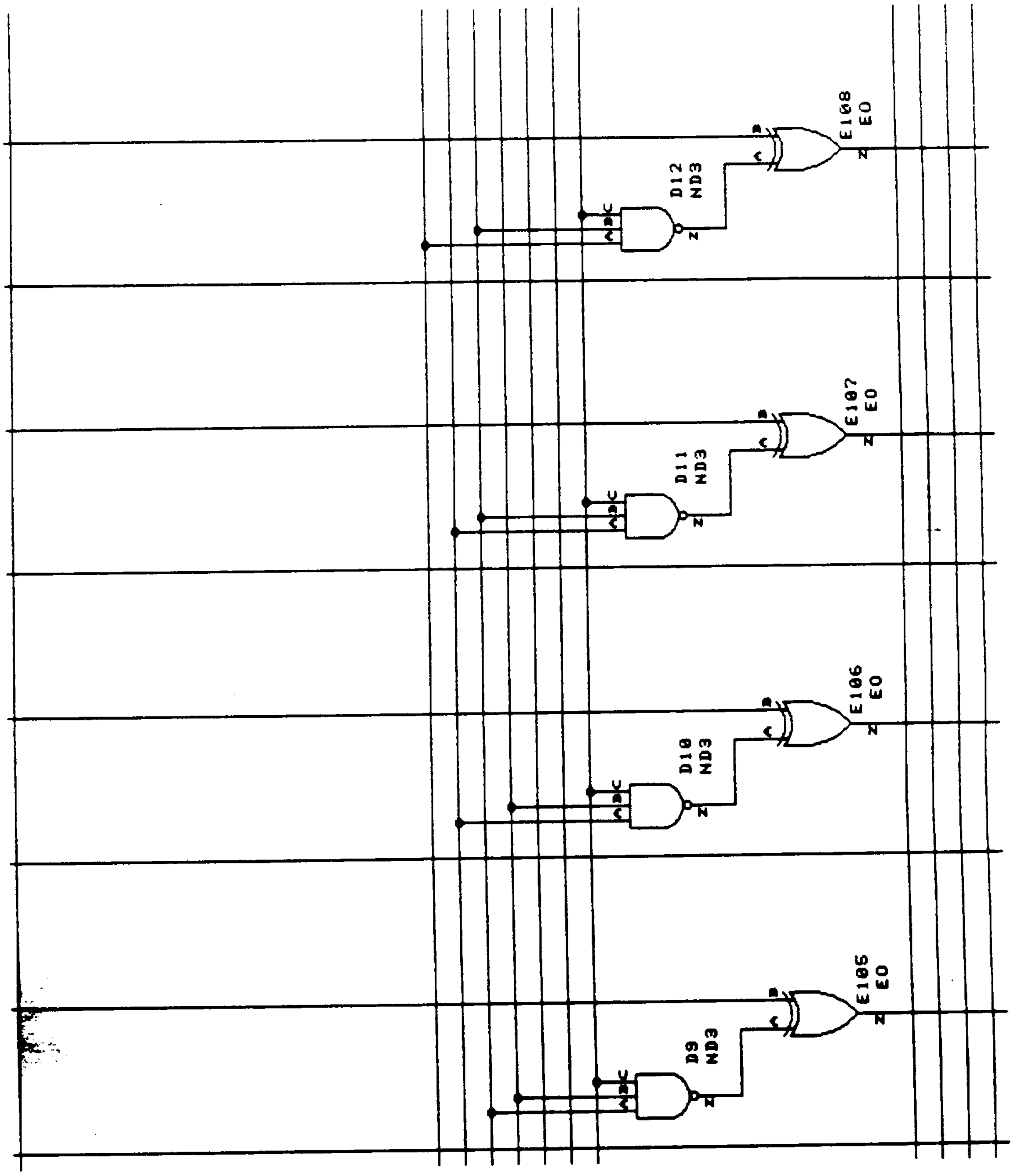

D12
ND3
E108
EO
D11
ND3
E107
EO
D10
ND3
E106
EO
D9
ND3
E105
EO
A
B
C
Z

Figure 40

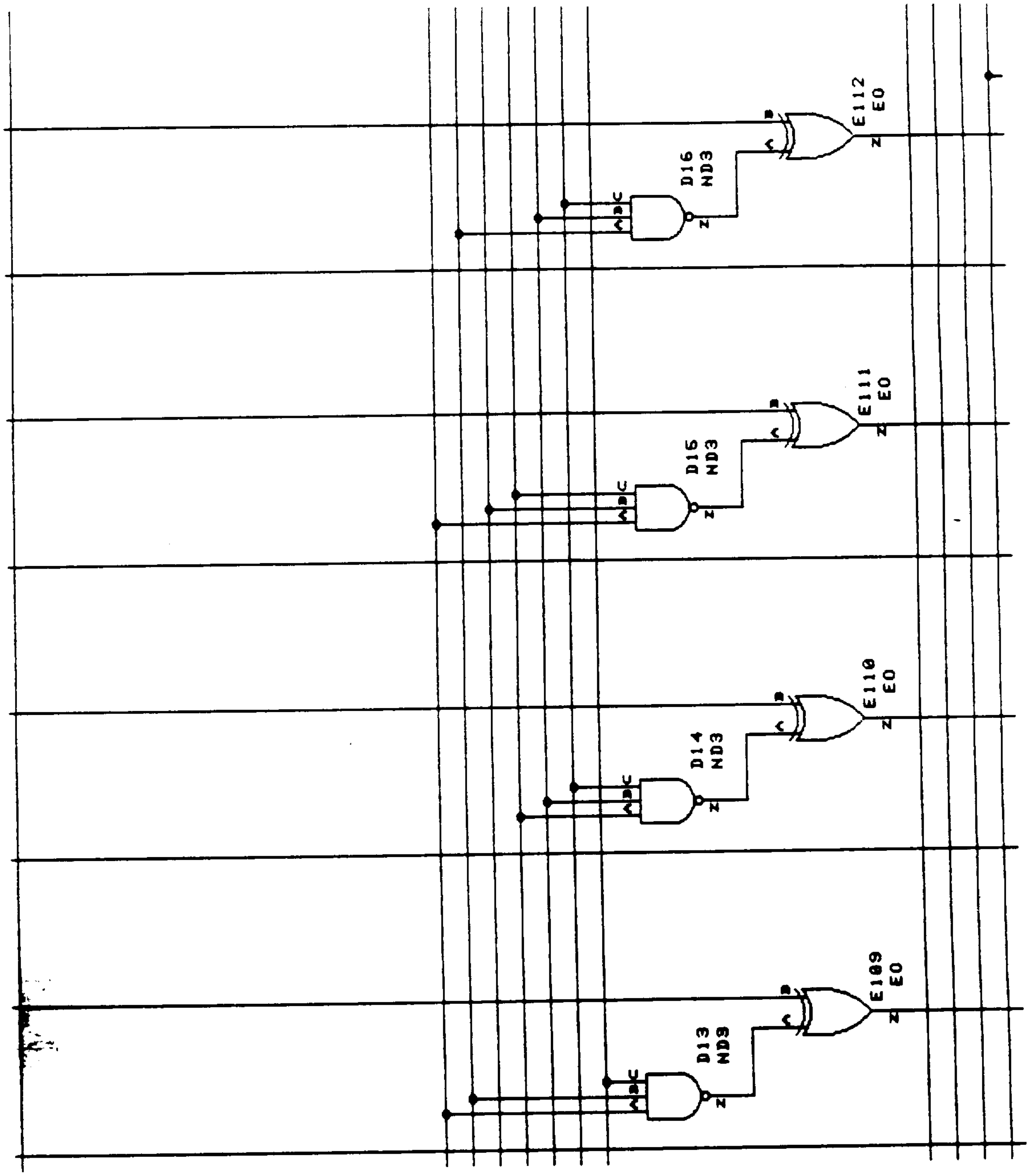

D13
ND3
E109
EO
D14
ND3
E110
EO
D15
ND3
E111
EO
D16
ND3
E112
EO
A
B
C
Z

Figure 41

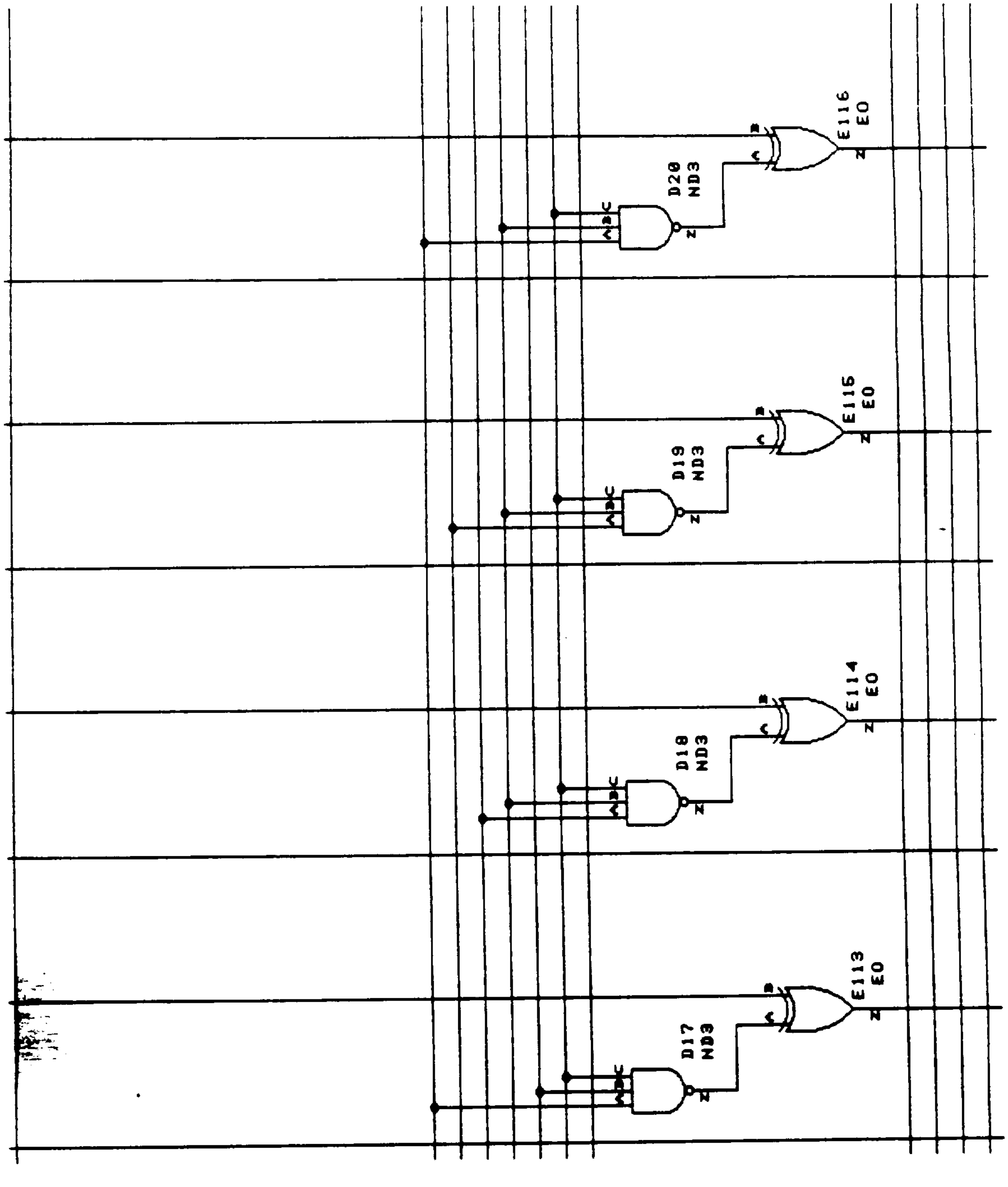

D17
ND3
E113
EO
D18
ND3
E114
EO
D19
ND3
E115
EO
D2Ø
ND3
E116
EO
A
B
C
Z

Figure 42

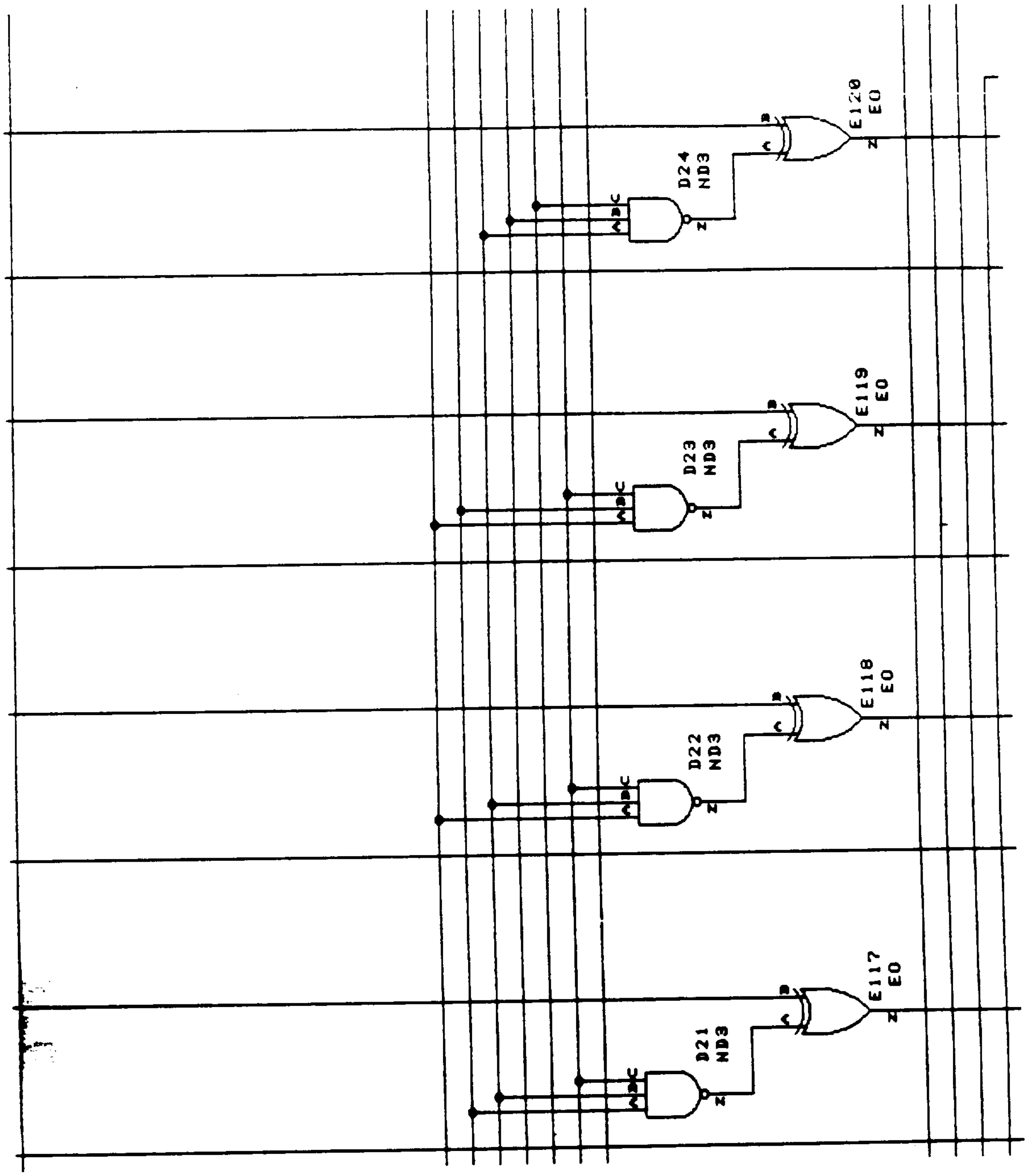

D24
ND3
E120
EO
D23
ND3
E119
EO
D22
ND3
E118
EO
D21
ND3
E117
EO
A
B
C
Z

Figure 43

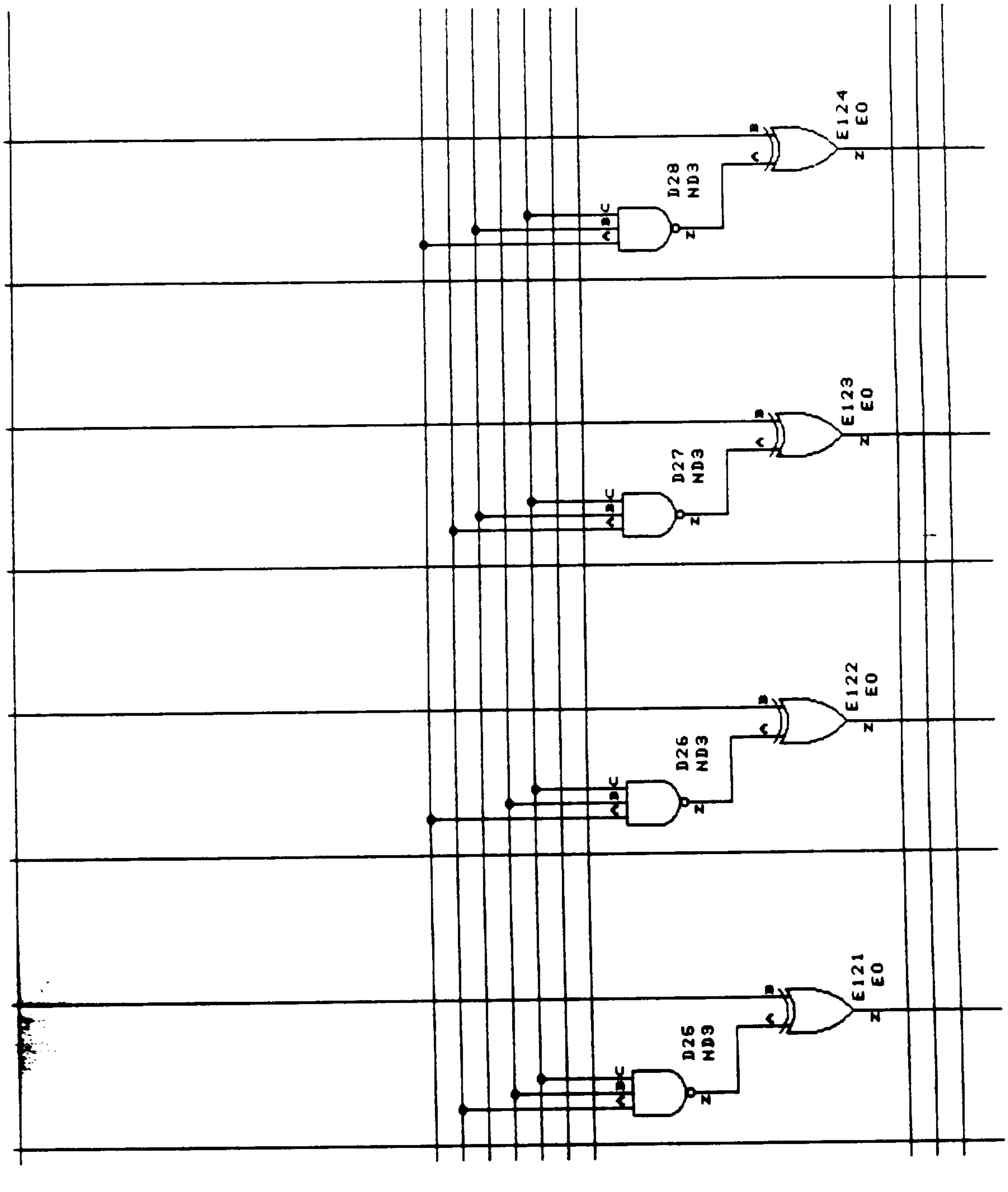

D28
ND3
A
B
C
Z
E124
EO
A
B
Z
D27
ND3
A
B
C
Z
E123
EO
A
B
Z
D26
ND3
A
B
C
Z
E122
EO
A
B
Z
D26
ND3
A
B
C
Z
E121
EO
A
B
Z

Figure 44

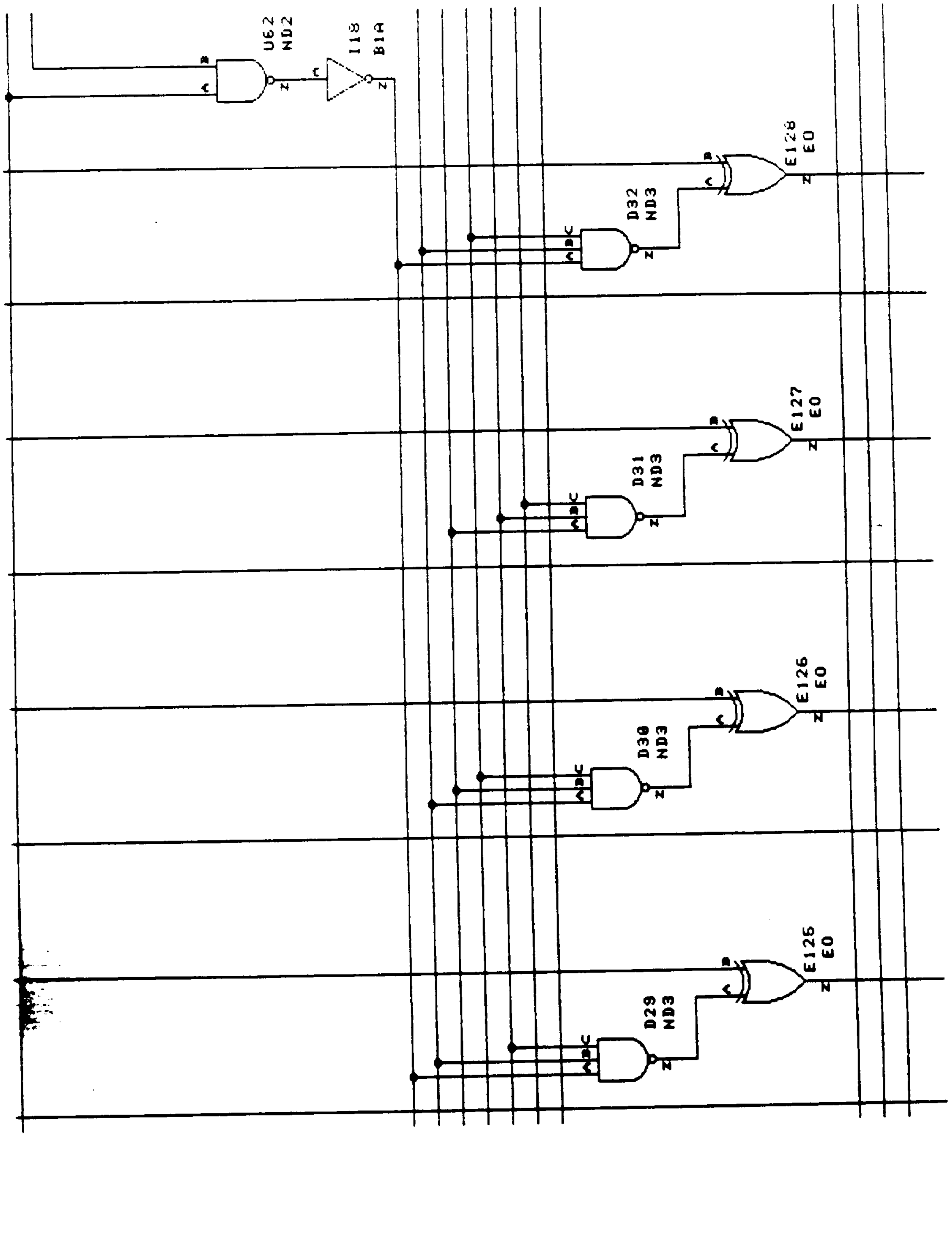

U62
ND2
I18
B1A
E128
EO
D32
ND3
E127
EO
D31
ND3
E126
EO
D30
ND3
E126
EO
D29
ND3
A
B
C
Z

Figure 45

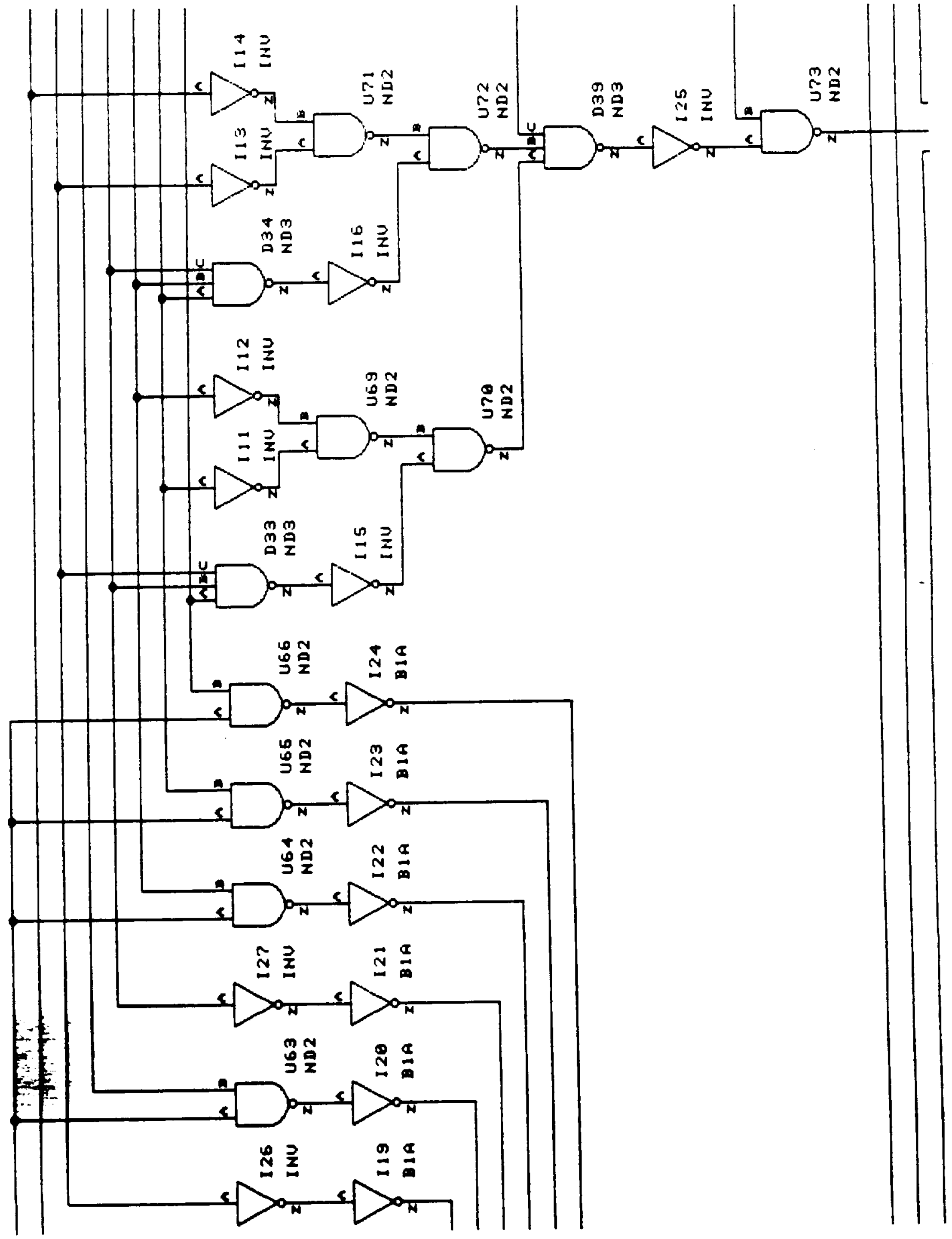

I14
INV
I13
INV
U71
ND2
U72
ND2
D39
ND3
I25
INV
U73
ND2
D34
ND3
I16
INV
I12
INV
I11
INV
U69
ND2
U70
ND2
D33
ND3
I15
INV
U66
ND2
I24
B1A
U65
ND2
I23
B1A
U64
ND2
I22
B1A
I27
INV
I21
B1A
U63
ND2
I20
B1A
I26
INV
I19
B1A

Figure 46

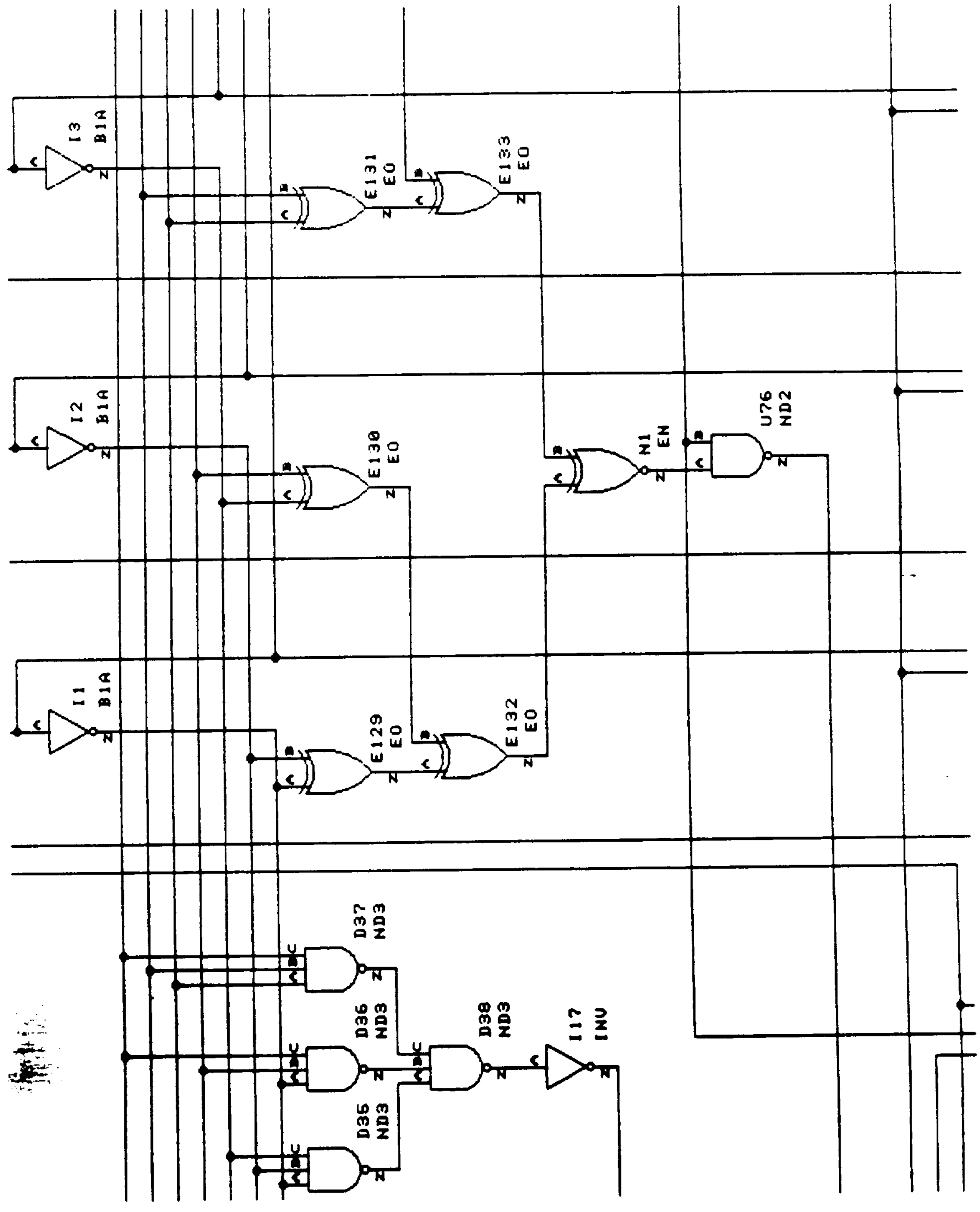

I3
B1A
E131
EO
E133
EO
I2
B1A
E130
EO
N1
EN
U76
ND2
I1
B1A
E129
EO
E132
EO
D37
ND3
D36
ND3
D38
ND3
I17
INV
D35
ND3

Figure 47

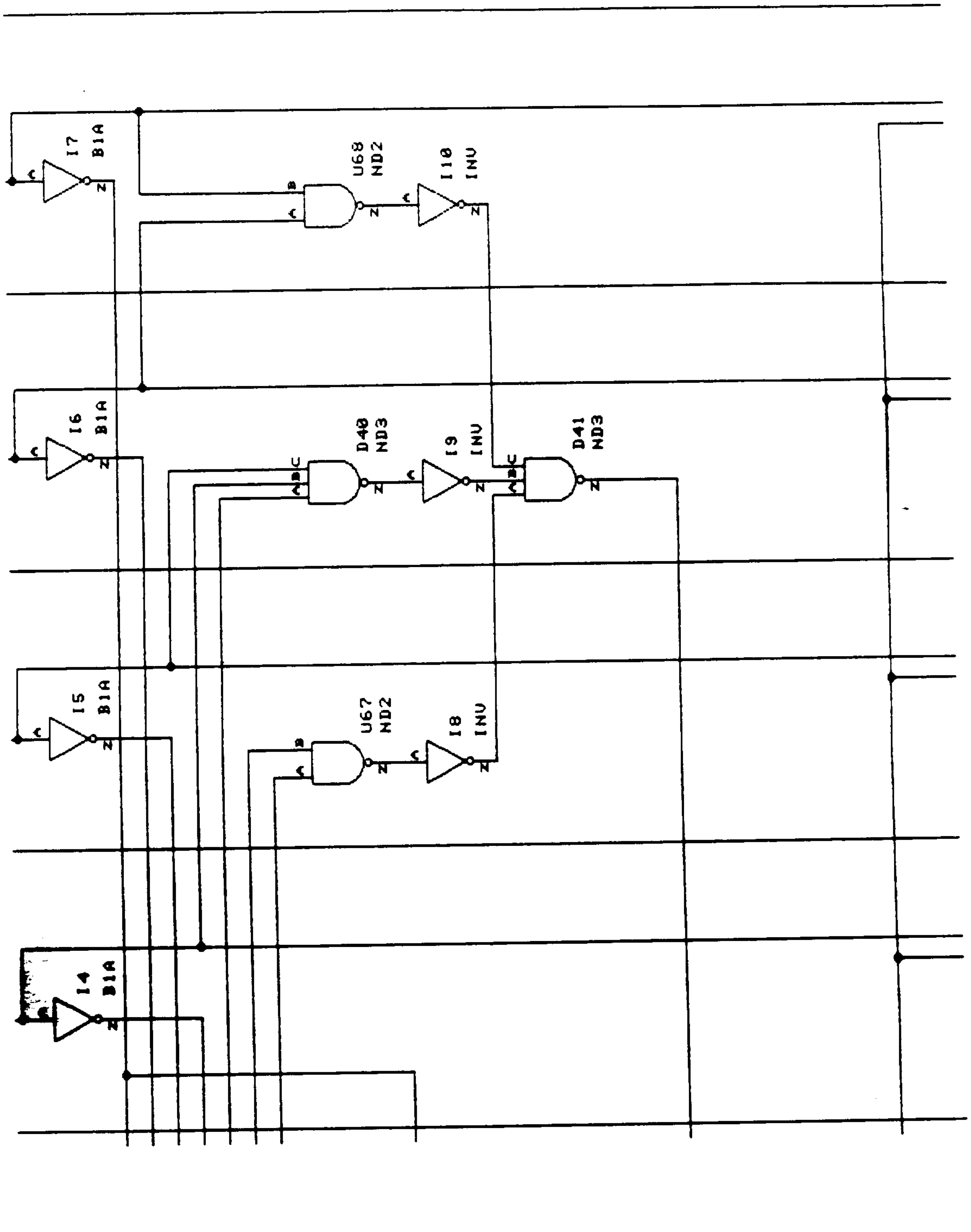
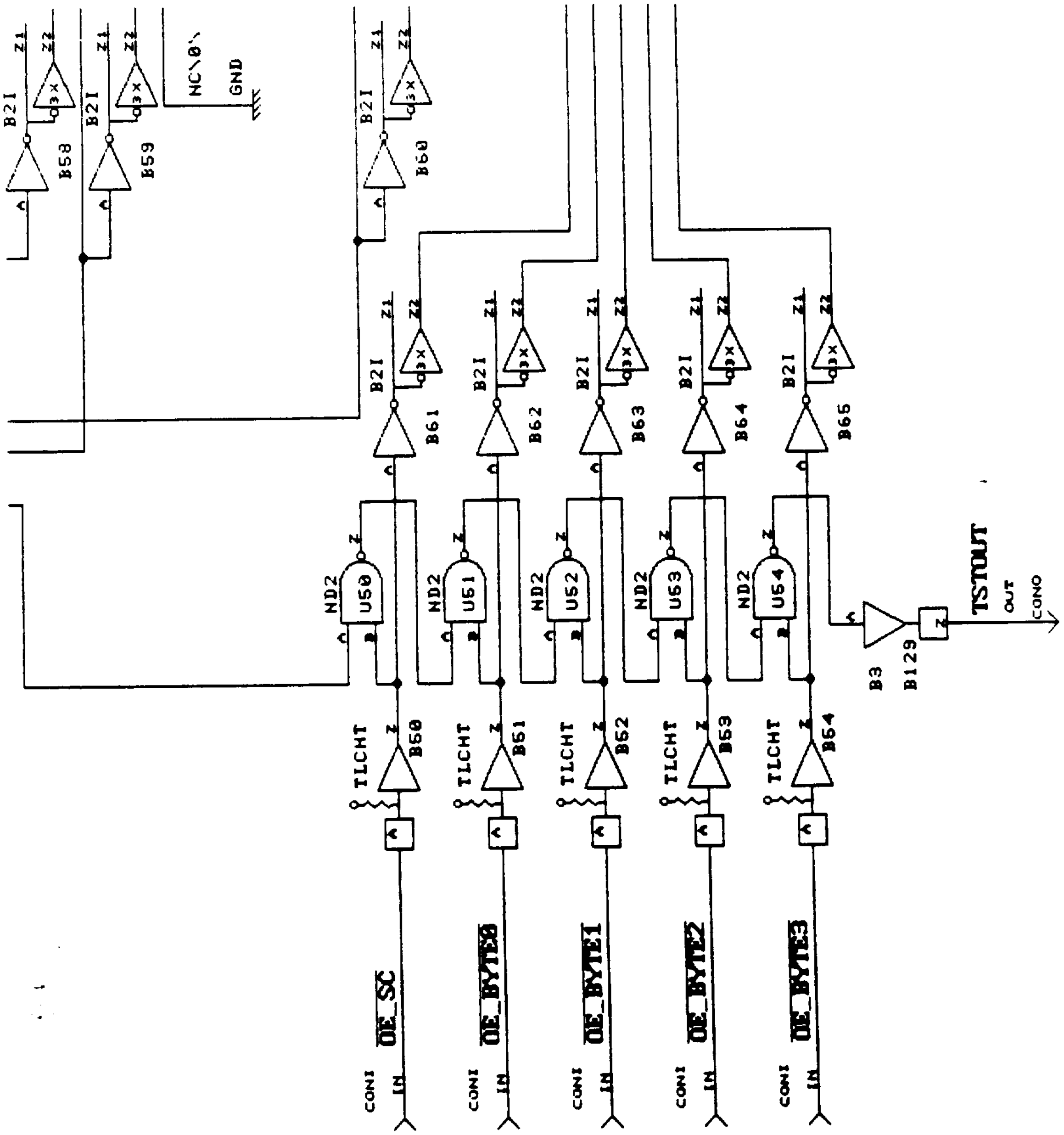

I7
B1A
U68
ND2
I10
INV
I6
B1A
D40
ND3
I9
INV
D41
ND3
I5
B1A
U67
ND2
I8
INV
I4
B1A

Figure 48



CONI
IN
OE_SC
OE_BYTE0
OE_BYTE1
OE_BYTE2
OE_BYTE3
TLCHT
B50
B51
B52
B53
B54
ND2
U50
U51
U52
U53
U54
B21
B58
B59
B60
B61
B62
B63
B64
B66
3x
Z1
Z2
NC\0'\
GND
B3
B129
TSTOUT
OUT
CONO

Figure 49

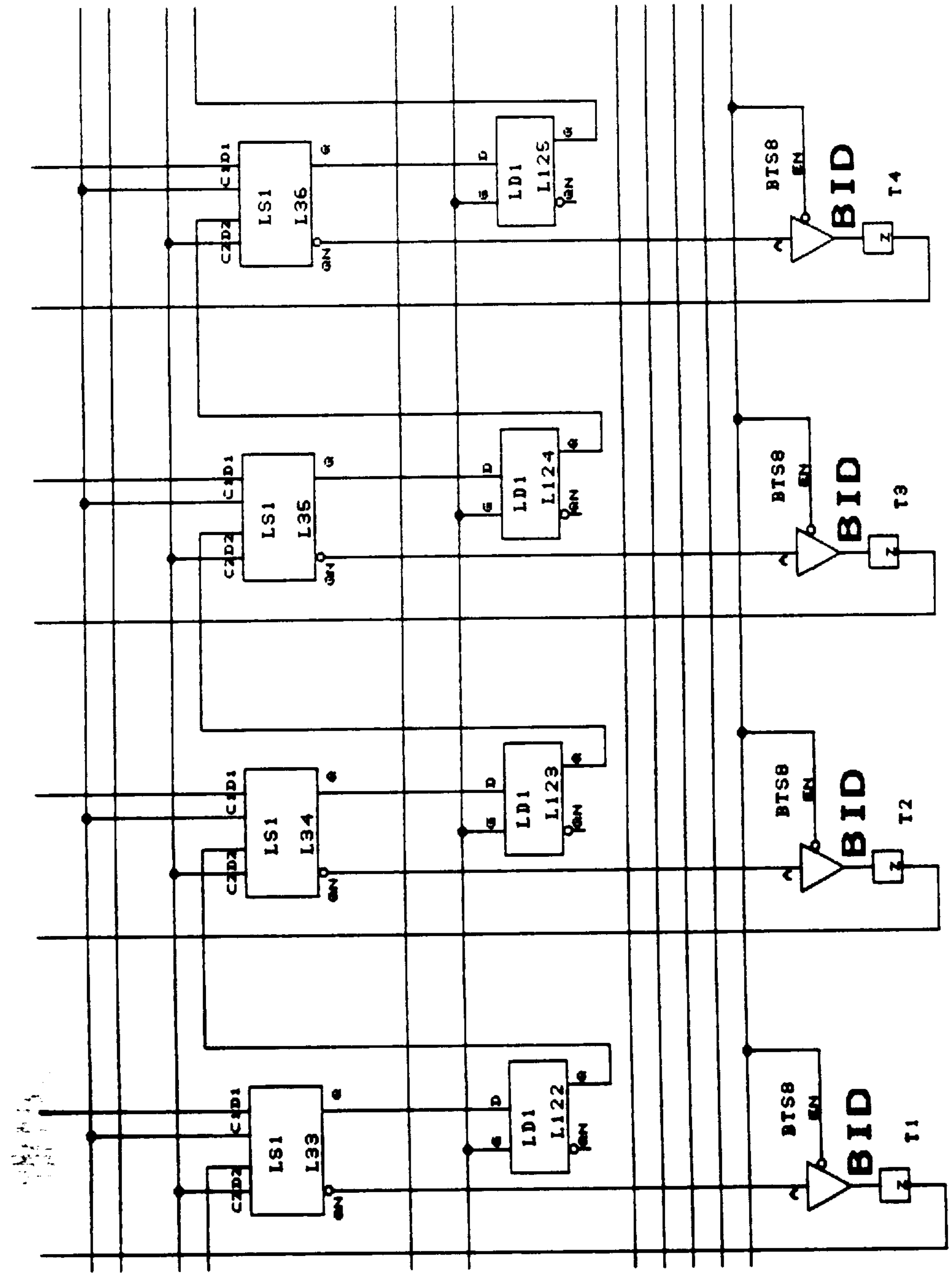

LS1
L33
LS1
L34
LS1
L35
LS1
L36
C1D1
C2D2
G
GN
LD1
L122
LD1
L123
LD1
L124
LD1
L125
D
G
GN
BTS8
EN
BID
Z
T1
T2
T3
T4

Figure 5A

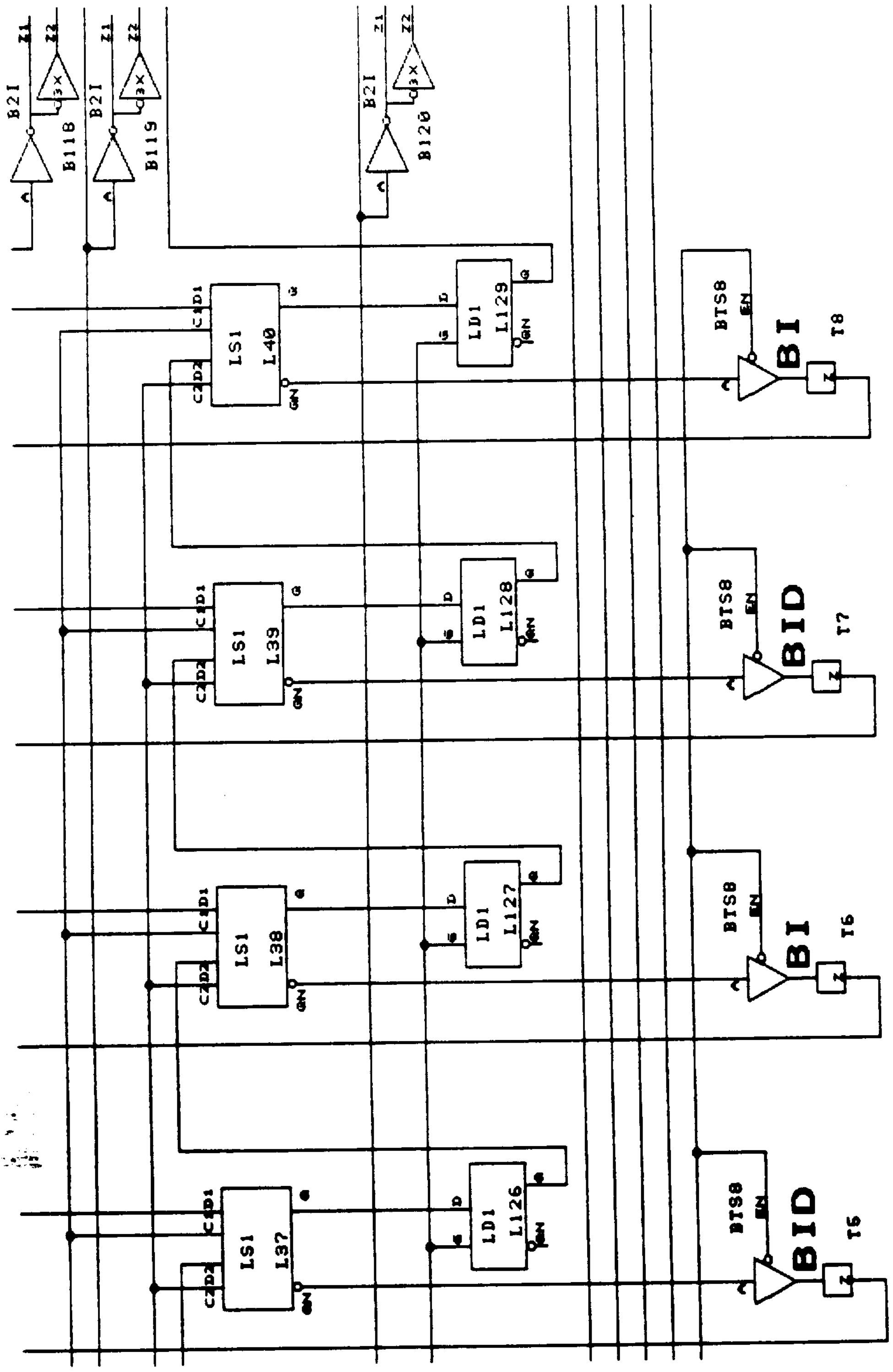
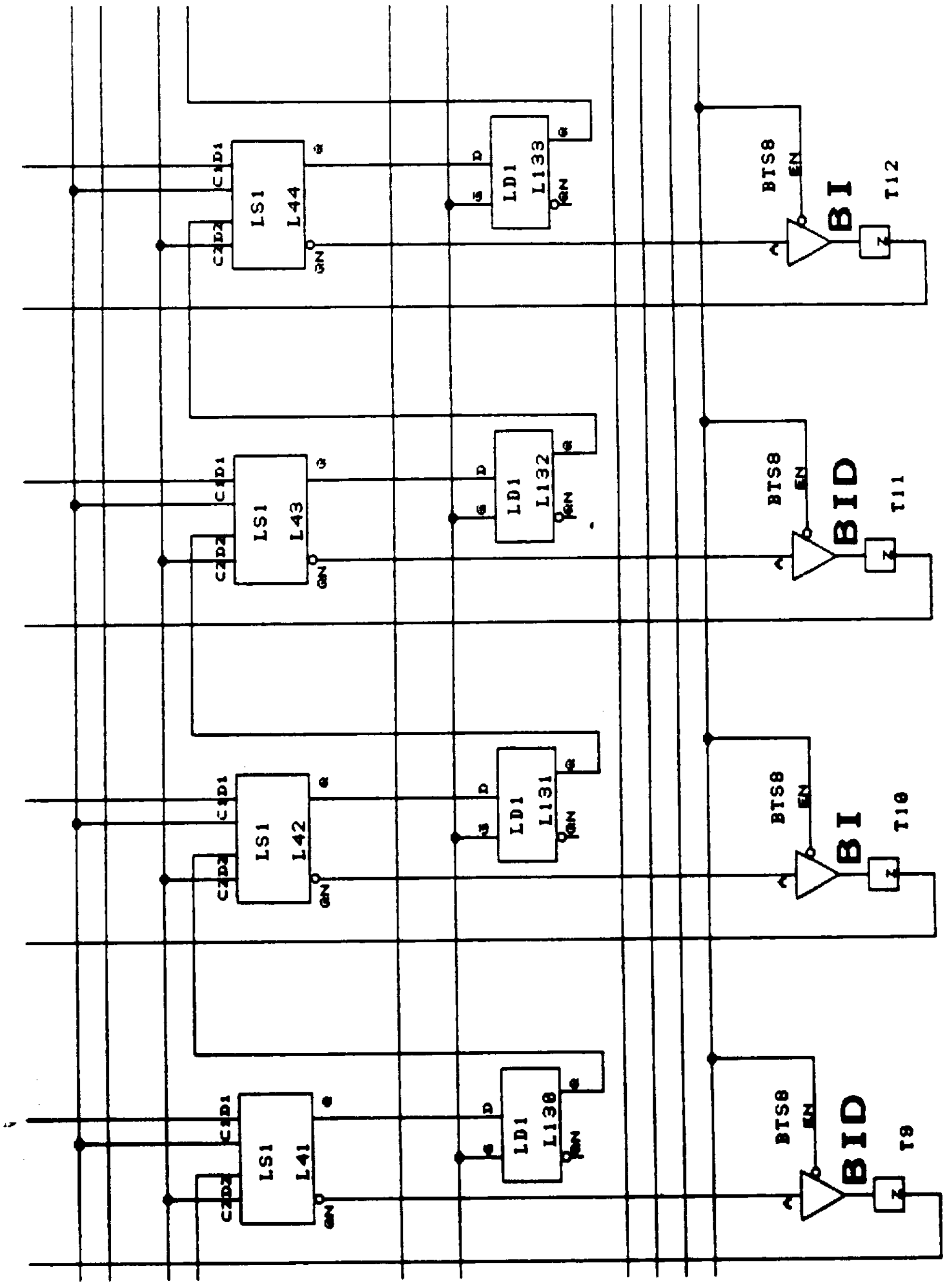

B2I
Z1
Z2
3X
B118
B2I
Z1
Z2
3X
B119
B2I
Z1
Z2
3X
B120
C1D1
C2D2
LS1
L40
G
GN
D
G
LD1
L129
GN
G
BTS8
EN
BI
T8
C1D1
C2D2
LS1
L39
G
GN
D
G
LD1
L128
GN
G
BTS8
EN
BID
T7
C1D1
C2D2
LS1
L38
G
GN
D
G
LD1
L127
GN
G
BTS8
EN
BI
T6
C1D1
C2D2
LS1
L37
G
GN
D
G
LD1
L126
GN
G
BTS8
EN
BID
T5
Z

Figure 51



C1D1
C2D2
LS1
L41
L42
L43
L44
Q
QN
D
G
GN
LD1
L130
L131
L132
L133
BTS8
EN
A
Z
BID
BI
BI
BID
T9
T10
T11
T12

Figure 52

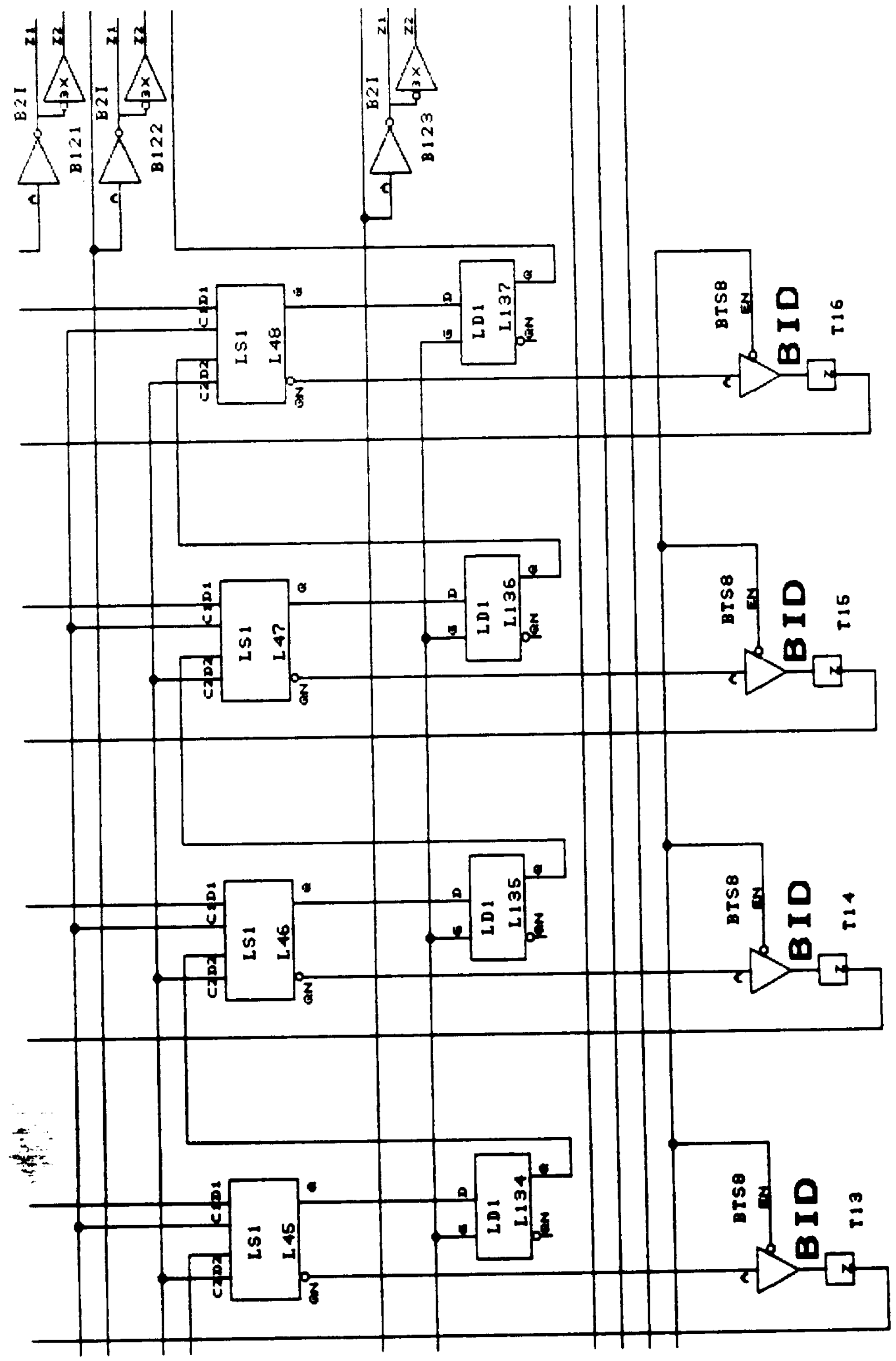

B2I
B121
B122
B123
Z1
Z2
3X
LS1
L45
L46
L47
L48
C1D1
C2D2
G
GN
LD1
L134
L135
L136
L137
D
BTS8
EN
BID
T13
T14
T15
T16
Z

Figure 53

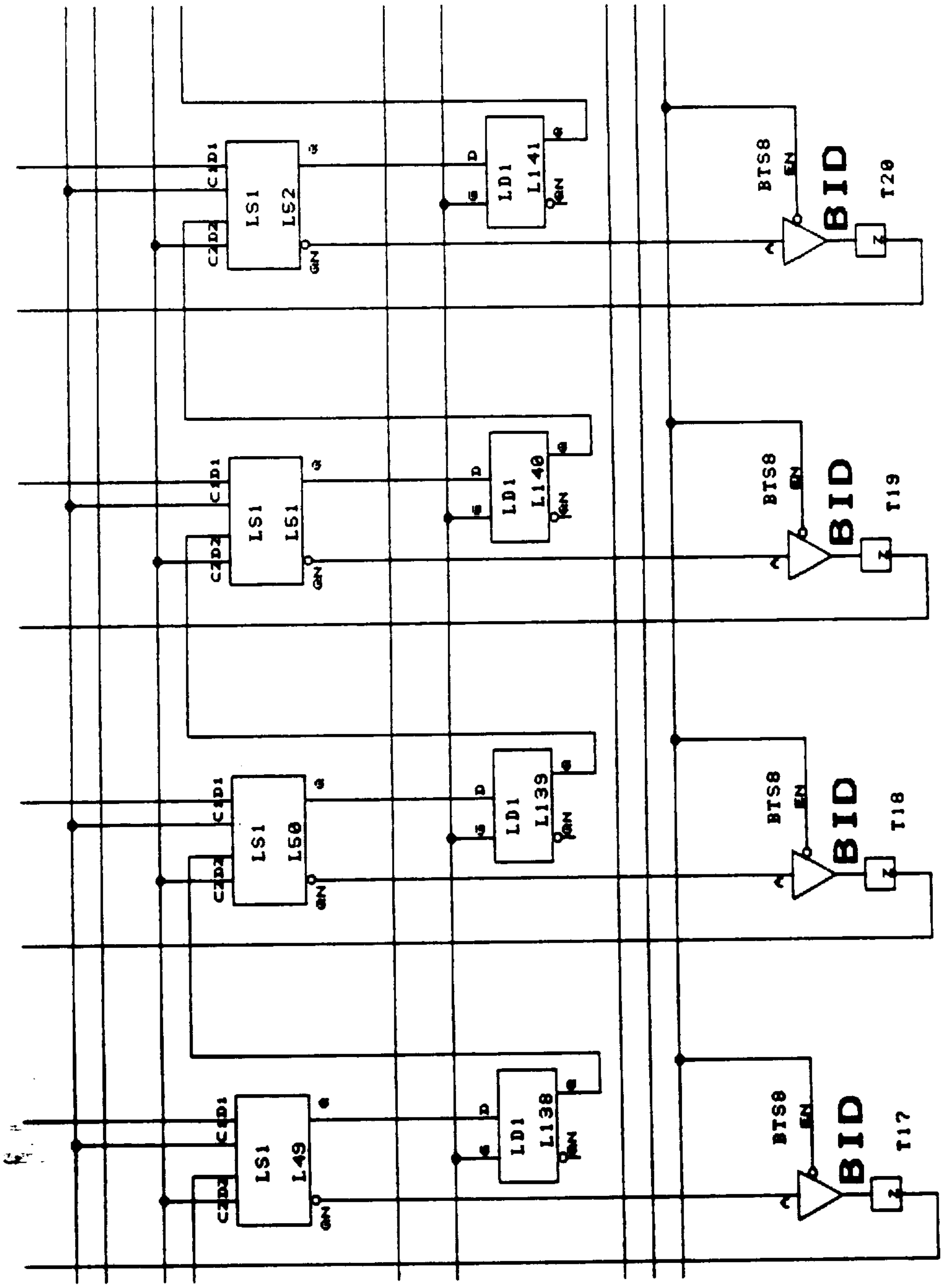

LS1
L52
C1D1
C2D2
Q
GN
LD1
L141
D
G
GN
Q
BTS8
EN
BID
T20
A
Z
LS1
L51
C1D1
C2D2
Q
GN
LD1
L140
D
G
GN
Q
BTS8
EN
BID
T19
A
Z
LS1
L50
C1D1
C2D2
Q
GN
LD1
L139
D
G
GN
Q
BTS8
EN
BID
T18
A
Z
LS1
L49
C1D1
C2D2
Q
GN
LD1
L138
D
G
GN
Q
BTS8
EN
BID
T17
A
Z

Figure 54

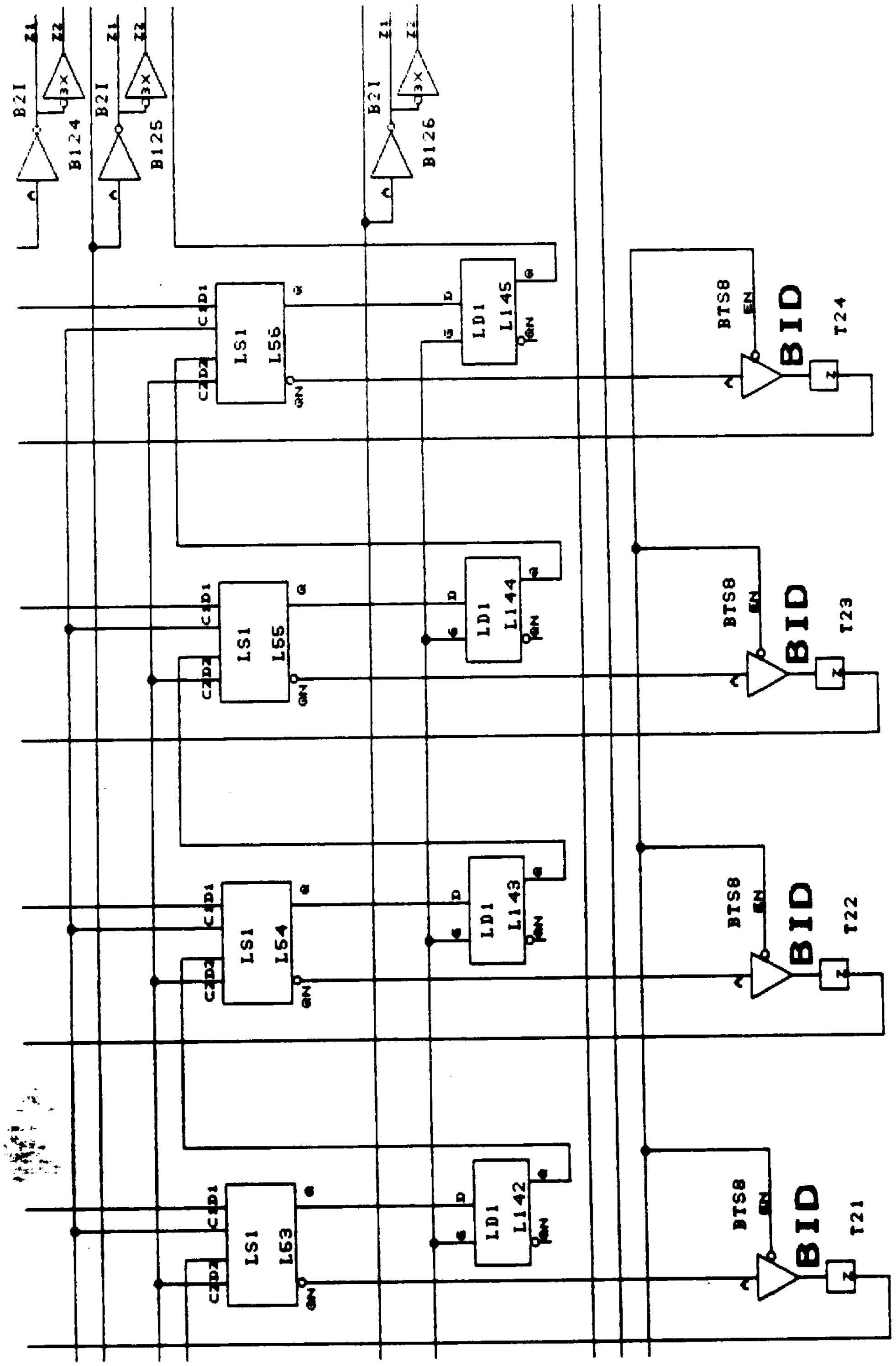

B2I
B124
B2I
B125
B2I
B126
Z1
Z2
3x
A
LS1
L56
LS1
L55
LS1
L54
LS1
L53
C1D1
C2D2
Q
QN
LD1
L145
LD1
L144
LD1
L143
LD1
L142
D
G
GN
BTS8
EN
BID
Z
T24
T23
T22
T21

Figure 55

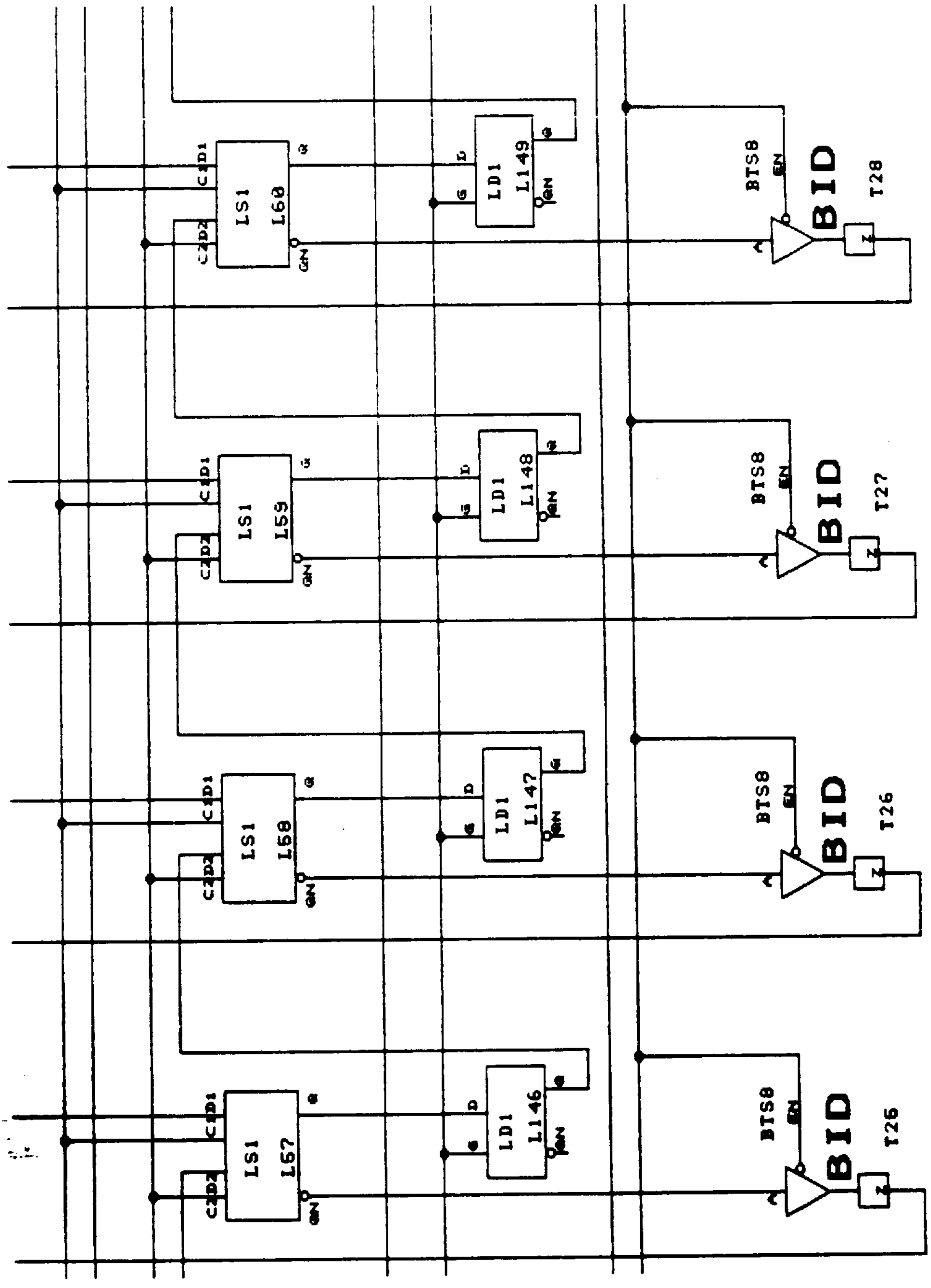

LS1
L57
LS1
L58
LS1
L59
LS1
L60
C1D1
C2D2
Q
QN
LD1
L146
LD1
L147
LD1
L148
LD1
L149
D
G
QN
BTS8
EN
BID
T26
T26
T27
T28
Z

Figure 56

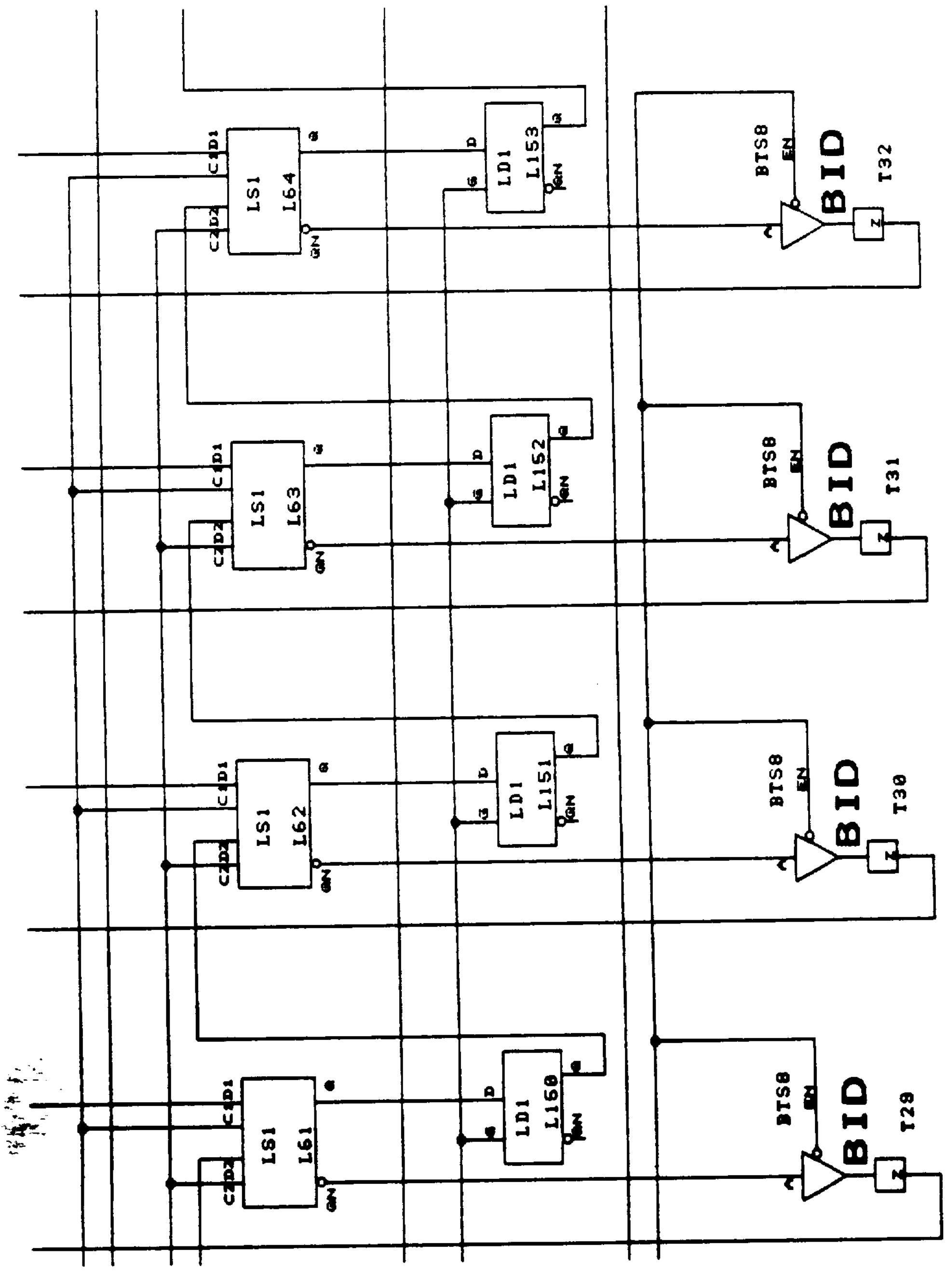

LS1
L61
L62
L63
L64
C1D1
C2D2
Q
QN
LD1
L150
L151
L152
L153
D
G
GN
BTS8
EN
A
BID
Z
T29
T30
T31
T32

Figure 57

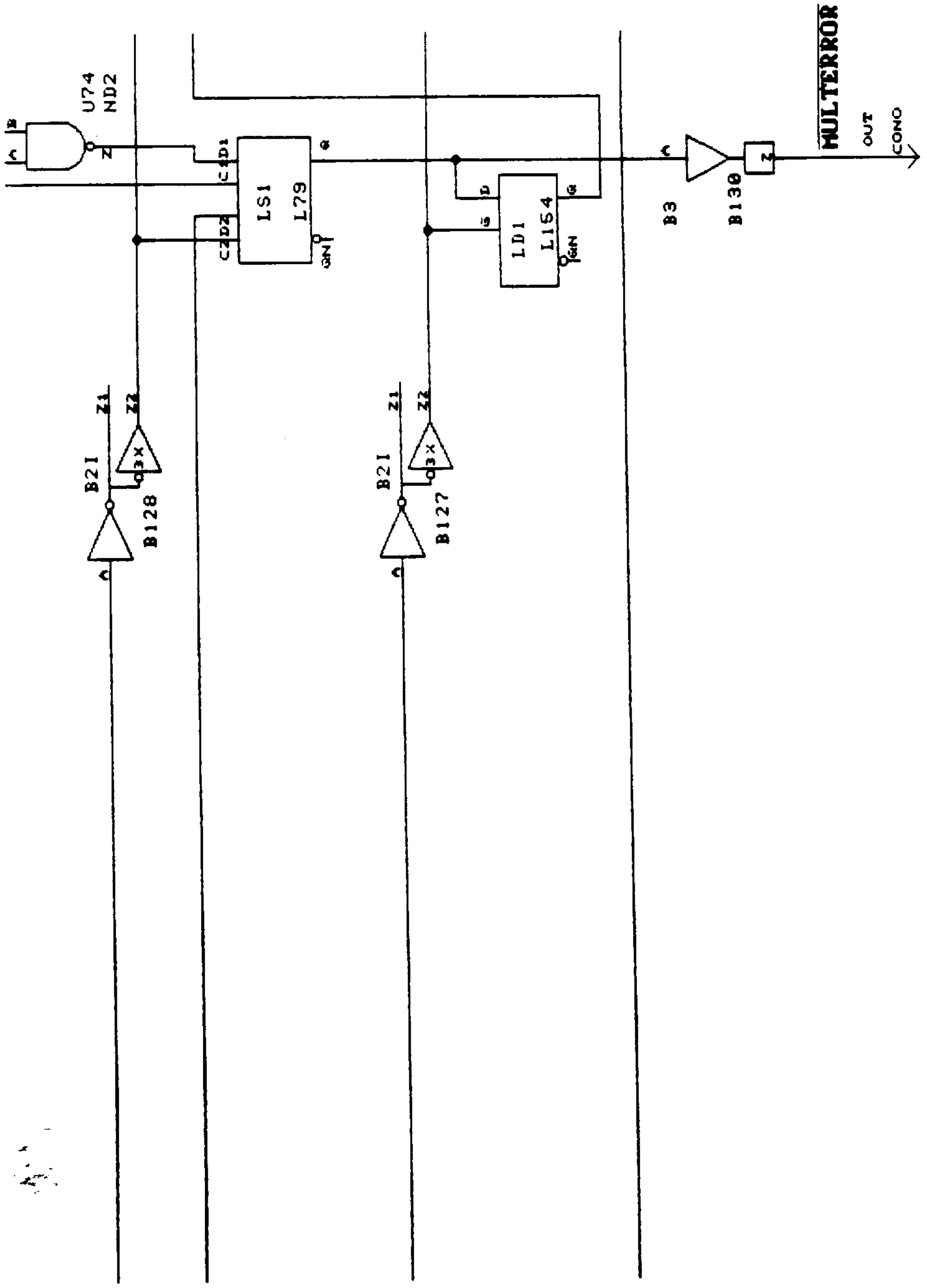

U74
ND2
B
A
Z
CSD1
CZD2
LS1
L79
Q
GN
D
G
LD1
L154
Q
GN
A
B3
Z
B130
MULTERROR
OUT
CONO
B2I
Z1
Z2
3X
B128
B2I
Z1
Z2
3X
B127
A
A

Figure 58

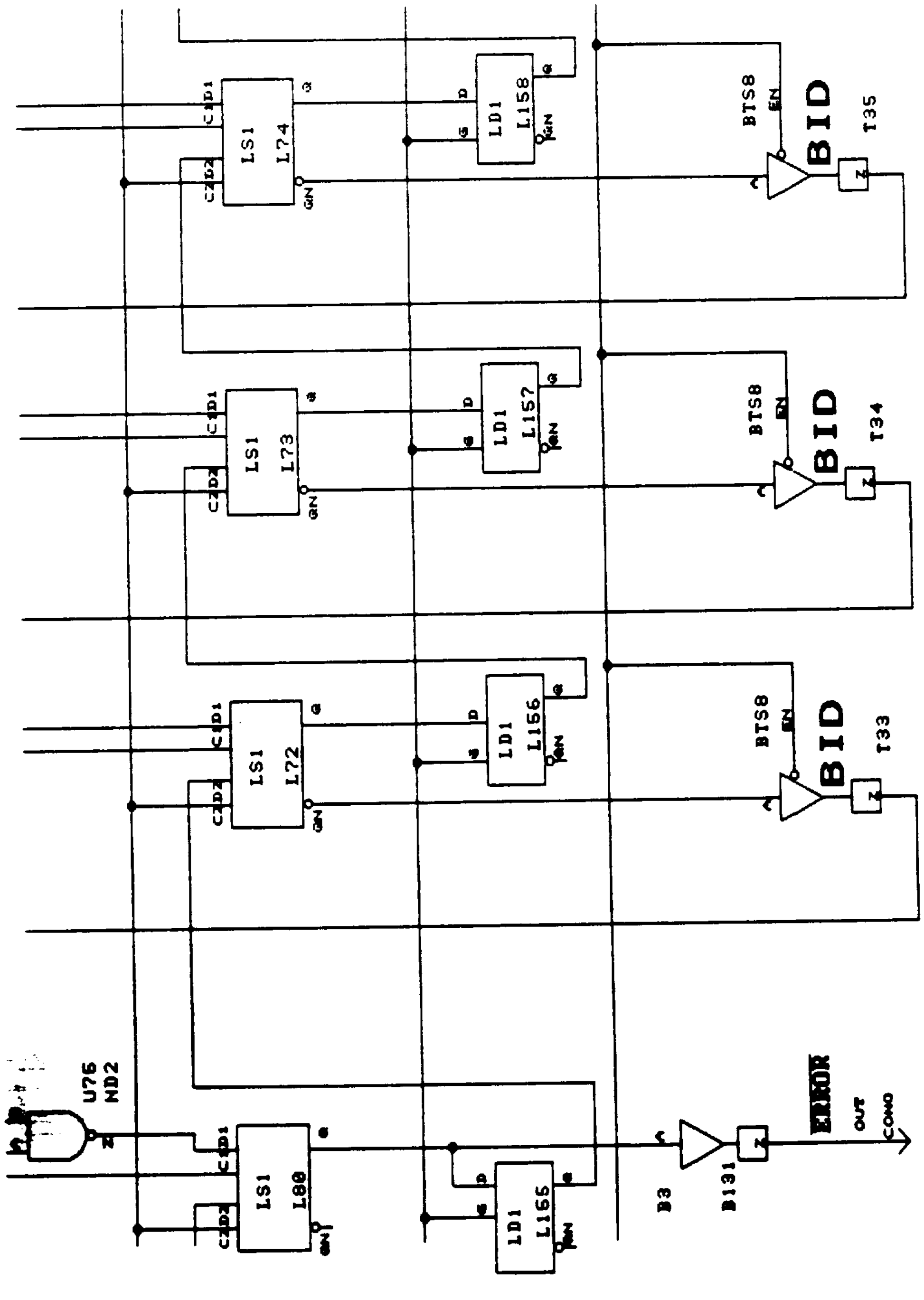
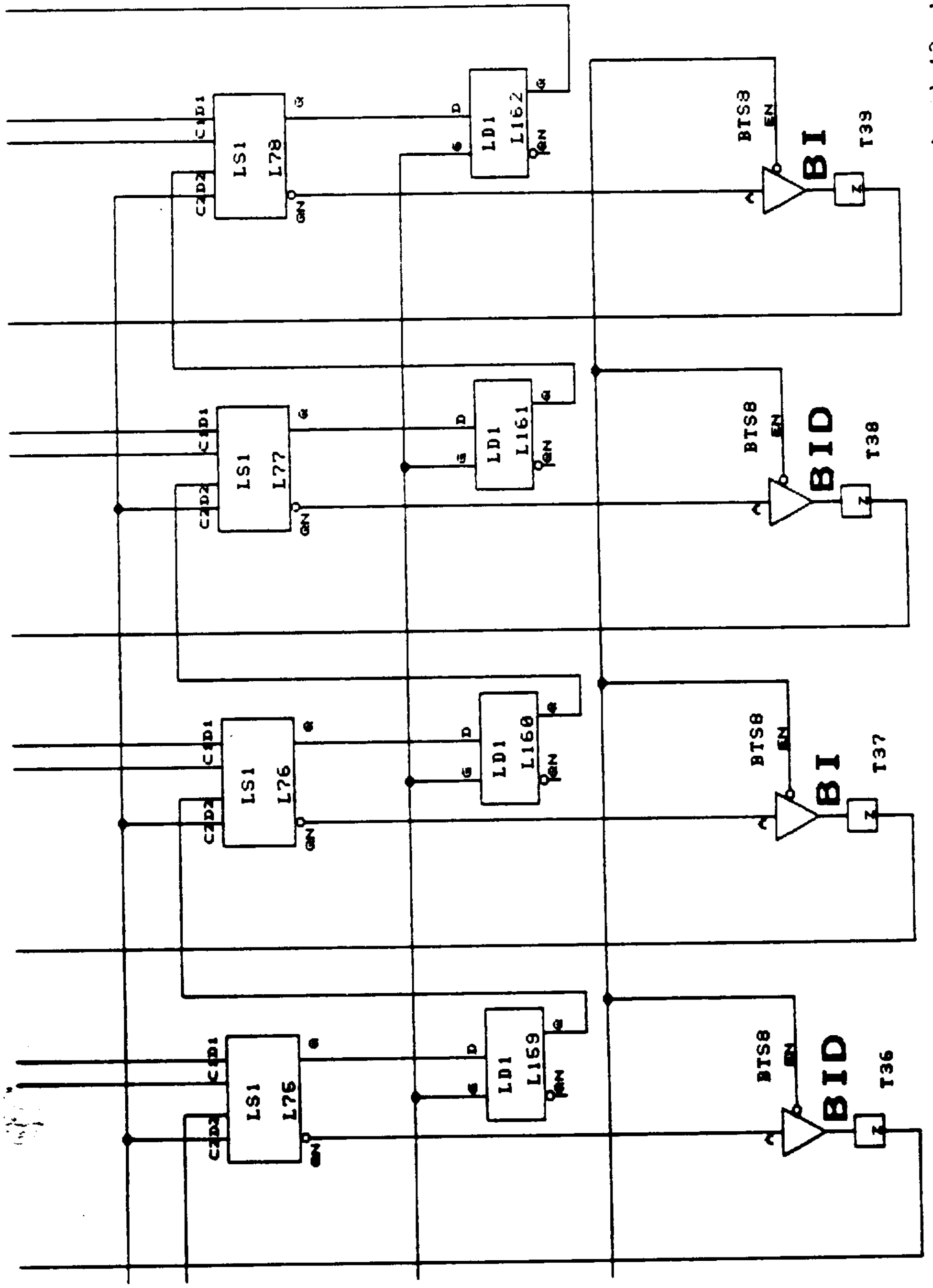

U76
ND2
LS1
L80
C1D1
C2D2
Q
GN
LD1
L155
D
G
B3
B131
ERROR
OUT
COMO
LS1
L72
LD1
L156
BTS8
EN
BID
T33
Z
LS1
L73
LD1
L157
BTS8
EN
BID
T34
LS1
L74
LD1
L158
BTS8
EN
BID
T35

Figure 59

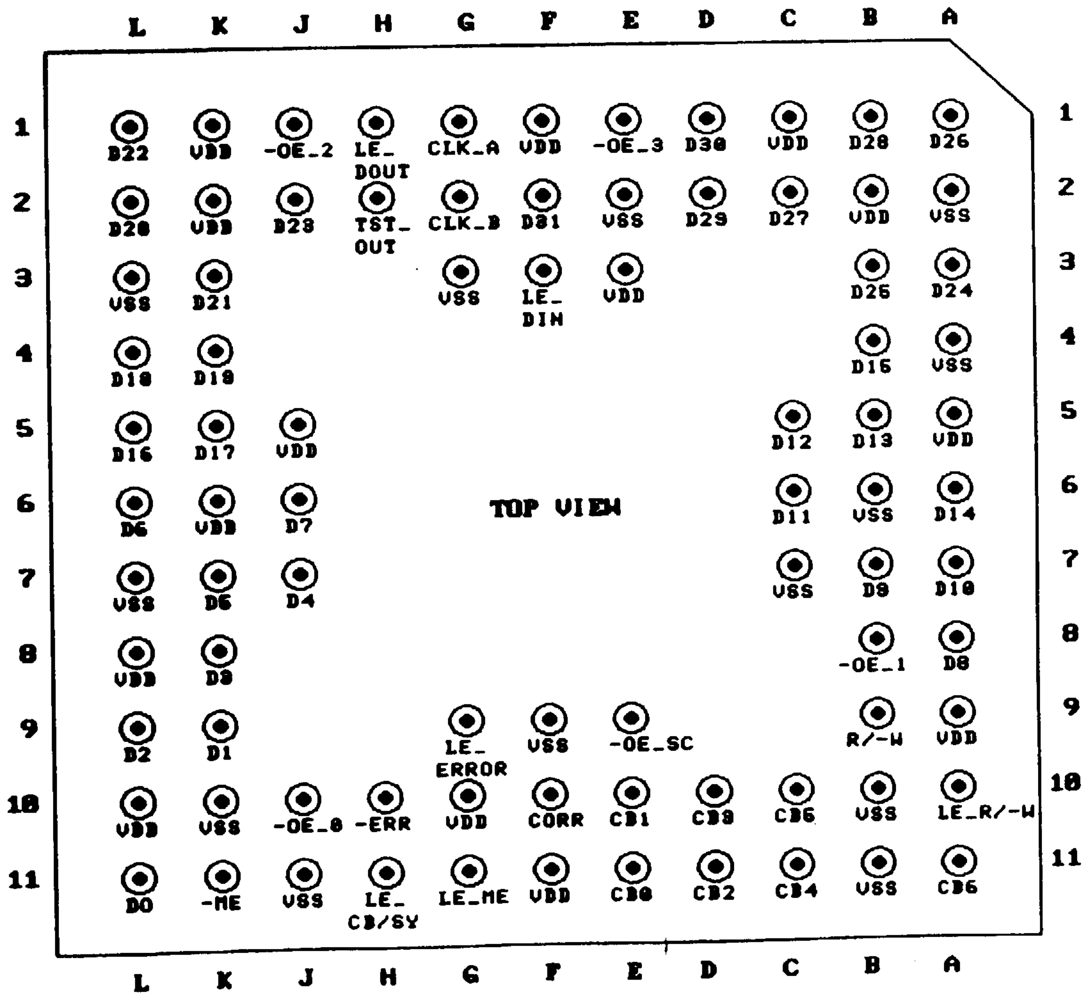


April 12, 1989

EDC.DWG

Figure 60

PIN ASSIGNMENT



TOP VIEW

	L	K	J	H	G	F	E	D	C	B	A
1	D22	VDD	-OE_2	LE_ DOUT	CLK_A	VDD	-OE_3	D30	VDD	D28	D26
2	D20	VDD	D23	TST_ OUT	CLK_B	D31	VSS	D29	D27	VDD	VSS
3	VSS	D21			VSS	LE_ DIN	VDD			D25	D24
4	D18	D19								D15	VSS
5	D16	D17	VDD						D12	D13	VDD
6	D6	VDD	D7						D11	VSS	D14
7	VSS	D5	D4						VSS	D9	D10
8	VDD	D3								-OE_1	D8
9	D2	D1			LE_ ERROR	VSS	-OE_SC			R/-W	VDD
10	VDD	VSS	-OE_0	-ERR	VDD	CORR	CB1	CB3	CB5	VSS	LE_R/-W
11	D0	-ME	VSS	LE_ CB/SY	LE_ME	VDD	CB0	CB2	CB4	VSS	CB6

EDAC PIN DWG

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. $DATA_0$ is the LSB; $DATA_{31}$ is the MSB.
CB/SY_{0-6}	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.
$R/\overline{W}$	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 $R/\overline{W}$	I	Latch Enable for $R/\overline{W}$ signal. This signal is kept HIGH during system modes (Read, Write, Pass Thru). When HIGH, the $R/\overline{W}$ latch is transparent for the $R/\overline{W}$ 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 $\overline{\text{ERROR}}$	I	Latch Enable for $\overline{\text{ERROR}}$ Output Latch.
LE $\overline{\text{ME}}$	I	Latch Enable for $\overline{\text{MULT\_ERROR}}$ 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)

$\overline{OE}$ BYTE$_{0-3}$	I	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 LOW, they enable the output buffer of the Data Output Latch. When HIGH, they force the Data Output Latch buffer into the high impedance mode.
$\overline{OE}$ SC	I	Output Enable for Syndrome or Check Bits. In the HIGH condition, the CB/SY outputs are in the high impedance state. When LOW, all CB/SY output lines are enabled.
$\overline{ERROR}$	O	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.
$\overline{MULT\_ERROR}$	O	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.
CLK_A, CLK_B	I	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.
TSTOUT	O	Test pin for parametric test.

	EDAC PROPAGATION DELAYS					
Mode	WRITE		READ			
From input	D_{0-31}	LE_CB/SY	D_{0-31}		CB/SY_{0-6}	
To output	CB/SY_{0-6} (nsec)		ME/ (nsec)	ERR/ (nsec)	ME/ (nsec)	ERR/ (nsec)
Output Load	Øpf		47pf	47pf	47pf	47pf
Part #						
1	54		68	72	57	60
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

```
/*
        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 behavier 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],goodbyte[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,syndr);
                    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)
```

```
   {
   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]);
      }
   }

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 probe1,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(probe1);
    _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);                            /* 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      */
       outp(772,control1);
       }
   else
       {
       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           */
   }


write_out(void)
   {
   control2=write_mode;
   outp(773,control2 & 0xFE);                  /* output enable data        */
   outp(772,control1 ¦ 0x02);                  /* latch R/W signal          */
   outp(772,control1);
   outp(788,0);                         /* write databyte 0        */
   outp(789,0);                         /*                1        */
   outp(790,0);                         /*                2        */
   outp(791,0);                         /*                3        */
   outp(772,control1 ¦ 0x01);                  /* latch data into data      */
   outp(772,control1);                         /* input registers           */
   outp(772,control1 ¦ 0x0C);                  /* latch CB, data into data  */
   outp(772,control1);                         /* output registers          */
   outp(773,control2);                         /* disable output            */
   }


write_err(void)
   {
   control2=read_mode;
   outp(773,control2 & 0x7E);                  /* enable output data, cb    */
   outp(772,control1 ¦ 0x02);                  /* latch R/W signal          */
   outp(772,control1);
   outp(788,0);                         /* write databyte 0        */
   outp(789,0);                         /*                1        */
```

```
  outp(790,0);           /*             2      */
  outp(791,0);           /*             3      */
  outp(779,0xff);              /* write checkbit        */
  outp(772,control1 ¦ 0x01);  /* latch in data, cb    */
  outp(772,control1);
  outp(772,control1 ¦ 0x30);  /* latch data, cb, error */
  outp(772,control1);         /* into output register  */
  outp(773,control2);         /* disable output        */
  }

write_in(void)
  {
  control2=write_mode;
  outp(773,control2 & 0x7E);  /* output enable data    */
  outp(772,control1 ¦ 0x02);  /* latch R/W signal      */
  outp(772,control1);
  outp(788,0);           /* write databyte 0   */
  outp(789,0);           /*             1      */
  outp(790,0);           /*             2      */
  outp(791,0);           /*             3      */
  outp(779,0);                /* write checkbit       */
  outp(772,control1 ¦ 0x01);  /* latch data into data  */
  outp(772,control1);         /* input registers       */
  outp(773,control2);         /* disable output        */
  }


read_in(void)
  {
  outp(773,control2 & 0xF9);  /* output enable data, cb  */
  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         */
  }

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;
            }
        }
```

```
   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 screen          */
    initialize();                          /* initialize EDAC            */
    start_time();                          /* 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();                          /* write data to EDAC    */
         calculate_checkbit();                 /* calculate checkbit    */
         read_all();                           /* read data from EDAC   */
         mask=Power_of_2(i);
         checkbit=(check & mask)/mask;
         if (checkbit != cbit)                 /* compare checkbits     */
            {
            error=true;
            ges_error=true;
            }
            if (kbhit())
               {
               key=getch();
               key=getch();
               switch (key)
                  {
                  case 67: goto finish;          /* Function key F9  */
                  case 68: goto exit;            /* Function key F10 */
                  }
               }
         }
         _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 ("                                                          F9-S
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);
```

```
    }
}
```

```
/*
        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 control1=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,syndr,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_all();                         /* write data to the EDAC */
        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);                   /* 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           */
   }

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

ORIGINAL PAGE IS
OF POOR QUALITY

```
    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);
    }

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 screen       */
   initialize();                         /* initialize EDAC         */
   start_time();                         /* start timer             */
   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();                        /* write data to the EDAC */
      for (i=0; i<4; i++)
         {
         conv_byte_to_bin(data[i]);
         for (j=0; j<8; j++)
            {
            if (vector==(8*i+j))
```

```
               {
               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;
   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          */
   }

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[4],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(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          */
                  mode=stuck_at;
                  start_program();
                  break;                            /* F5-Stuck_open           */
               case 63:
                  mode=stuck_open;
                  start_program();
                  break;                            /* F7-Clear         */
               case 65:
```

```
                        clear_fault();
                        pass=0;
                        fprintf(stdprn,"\n Signatures:\n\n");
                        for (i=0;i<=500;i++)
                            {
                            cycle[0]=i;
                            signature();
                            display_signature(11,20);
                            fprintf(stdprn,"\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(stdprn,"\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();
```

```
                              }
                           if (key==62)
                              {
                              get_fault_num();
                              inject(fault_num-1);
                              display_fdo();
                              }
                           if (key==63)
                              {
                              get_shift_num();
                              shift_fault(shift_num);
                              display_fdo();
                              }
                           if (key==64)
                              {
                              get_cycle_num();
                              signature();
                              display_signature(11,20);
                              display_fdo();
                              }
                           if (key==65)
                              {
                              outp(774,stuck_at);
                              signature();
                              outp(774,0);
                              display_signature(14,20);
                              display_fdo();
                              }
                           if (key==66)
                              {
                              outp(774,stuck_open);
                              signature();
                              /*outp(774,0);*/
                              display_signature(14,20);
                              display_fdo();
                              }

                           if (key==68)
                              {
                              endloop=true;
                              byte[0]=byte[1]=byte[2]=byte[3]=0xFF;
                              cb=0xaa;
                              fault_num=min_fault;
                              edac_screen();                        /* initialize
display */
                              signature();                          /* initial si
nature  */
                              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 67:                           /* F9-Cycle           */
```

```
            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,controll | 0x80);  /* shift fault one */
      outp(772,controll);         /* 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,"%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-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,"%71d",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,"%71d",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;
   outp(773,control2);                    /* set WRITE mode           */
   }

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,controll ¦ 0x80);        /* shift in 0s                */
      outp(772,controll);
      }
   }


inject(int pos)
   /*inject fault*/
   {
   outp(772,controll ¦ 0xC0);
   outp(772,controll);
   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,controll ¦ 0x03);         /* latch in data, R/W      */
   outp(772,controll);
   outp(773,control2);                /* disable output to EDAC  */
   outp(772,controll ¦ 0x3C);
   outp(772,controll);
   }

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);                            /* 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_seed();                           /* write seed                    */
   for (i=1; i<=cycle[pass]; i++)          /* repeat number of cycles       */
      {
      for (j=1; j<=41; j++)                /* shift LFSR 41 times           */
         {
         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                */
   if (fdo!=0 && fault_num!=max_fault)     /* check fdo output              */
      {
      fdo_error=true;
      }
   sign[0]=check;                          /*                               */
   sign[1]=me_bit+2*error_bit;             /*                               */
   sign[2]=data[0];                        /*    assign signature           */
   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="                              EDAC -  Fault Simulation
                 ";
   char *menu= "F1-Mode F2-Seed F3-Clear F4-Inject F5-Shift F6-Sign   F7-S_A F8-
S_O   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);
   }
}
```

```
/*
        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   %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(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,"%71d",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,"%71d",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,controll);                    /* set port 772 to 0        */
   outp(773,control2);                    /* set port 773 to E7hex    */
   clear_fault();                         /* 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);                       /*              3         */
   outp(779,ckb);                       /* 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       */
   }

read_signature(void)
   /* reads data, checkbits, and error signals from EDAC  */
   {
   int control_in;
   control2¦=0x81;
   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*/
   control2&=0x7E;
   outp(773,control2);                  /* disable output               */
   sign[0]=check;                       /*                              */
   sign[1]=me_bit+2*error_bit;          /*                              */
   sign[2]=data[0];                     /*   assign signature           */
   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;                       /* fault # 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);
   outp(774,mode);                       /* enable faults                   */
   outp(772,control1 ¦ 0x3C);            /* latch data into                 */
   outp(772,control1);                   /* output registers                */
   write_seed();
   for (i=1; i<=cycle[pass]; i++)        /* repeat number of cycles         */
      {
      for (j=1; j<=41; j++)              /* shift LFSR 41 times             */
         {
         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 data1,int cb1)
   /* performs fault simulation in pass 0 */
   {
   int i;
   write_data(data1,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(data1,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(data1,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="                      EDAC -  Fault Simulation
                ";
   char *menu= " F1-Mode  F2-Seed    F4-Stuck_at           F7-PrtSign   F8-Manual
      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="                    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

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

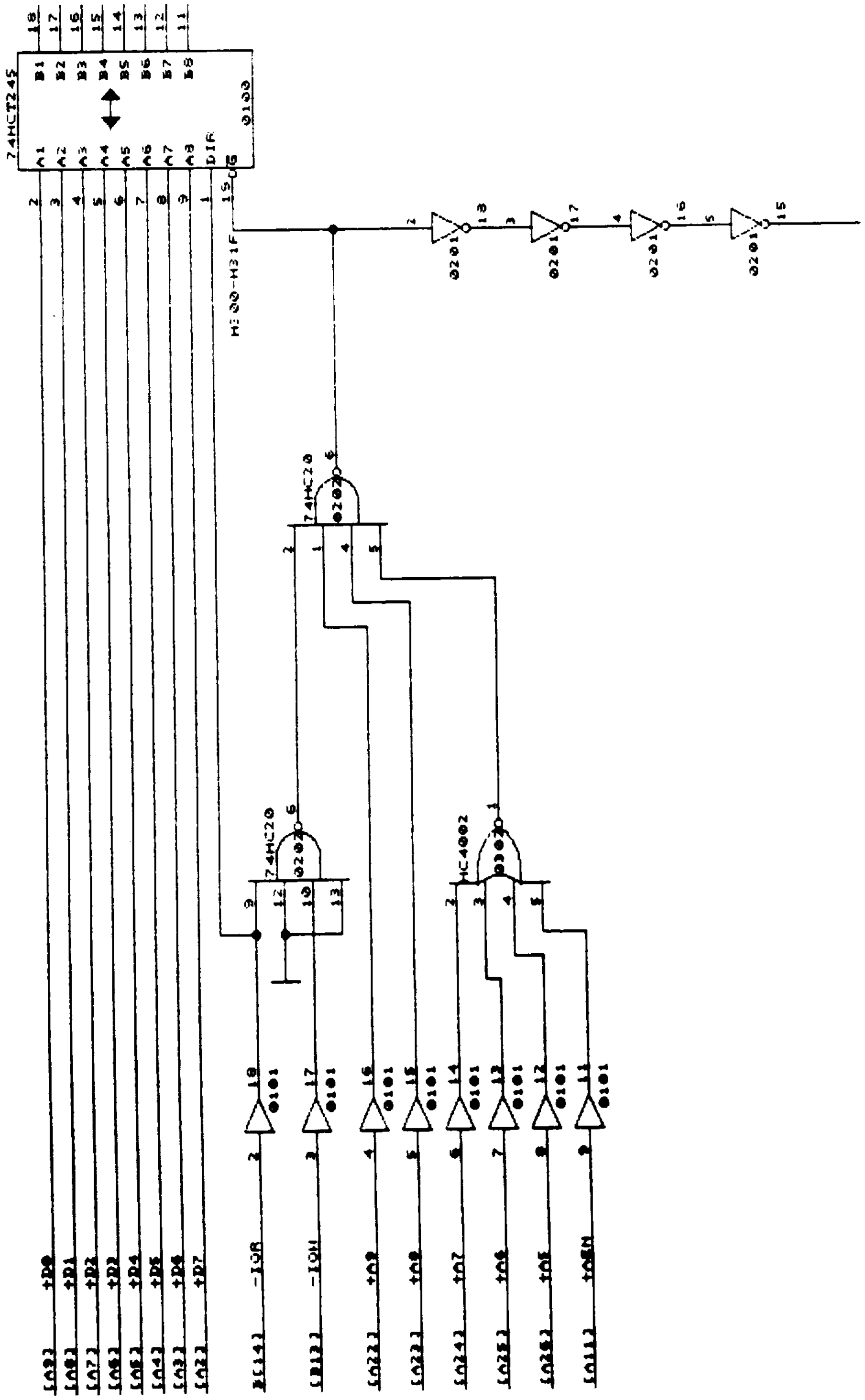

74HCT245
A1
A2
A3
A4
A5
A6
A7
A8
DIR
G
B1
B2
B3
B4
B5
B6
B7
B8
U100
H300-H31F
U201
74HC20
U202
HC4002
U203
U101
+D0
+D1
+D2
+D3
+D4
+D5
+D6
+D7
-IOR
-IOW
+A9
+A8
+A7
+A6
+A5
+AEN
[A9]
[A8]
[A7]
[A6]
[A5]
[A4]
[A3]
[A2]
B[14]
[B13]
[A22]
[A23]
[A24]
[A25]
[A26]
[A11]

Figure 1

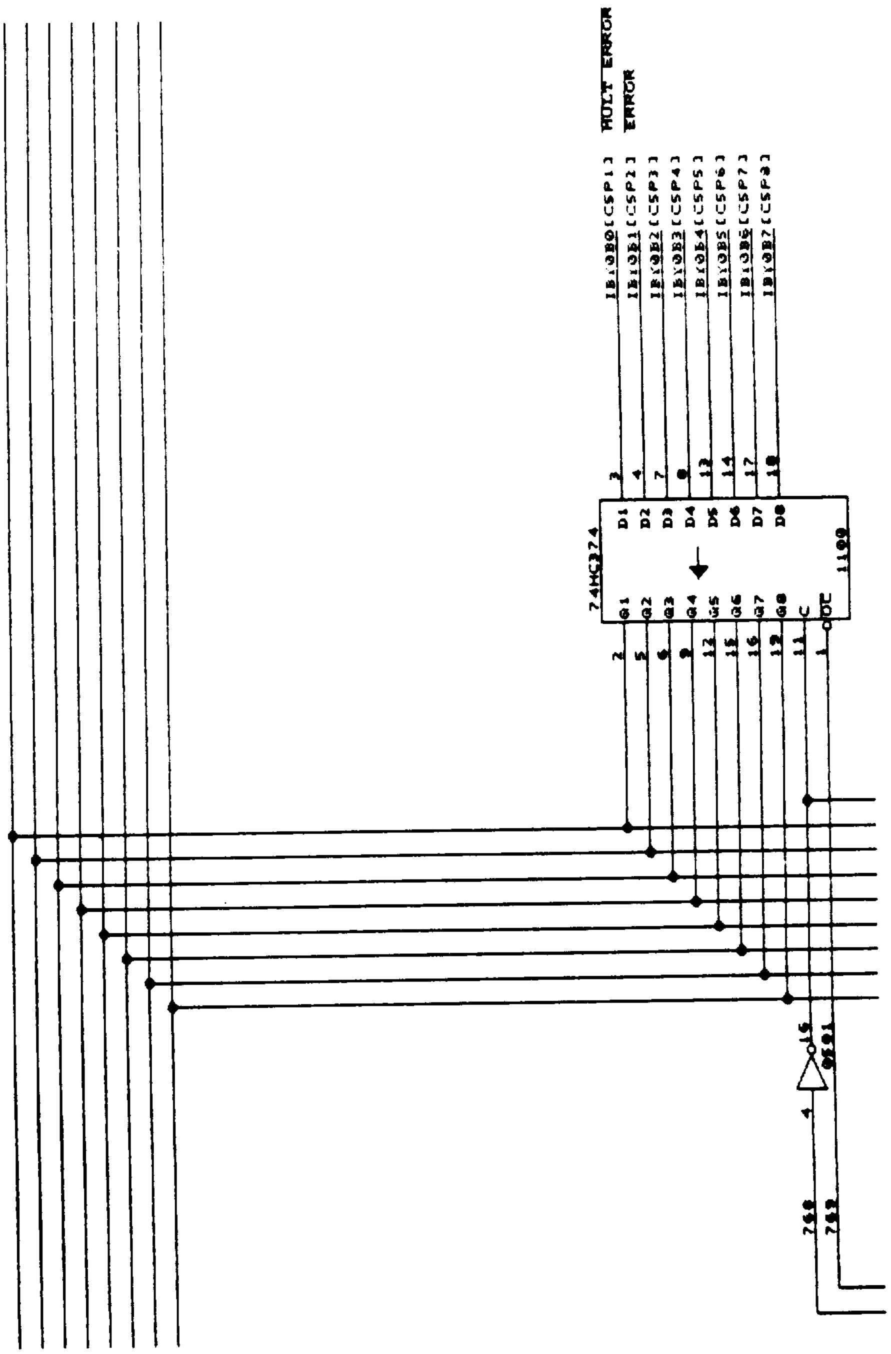

MULT ERROR
ERROR
IBYØBØ(CSP1)
IBYØB1(CSP2)
IBYØB2(CSP3)
IBYØB3(CSP4)
IBYØB4(CSP5)
IBYØB5(CSP6)
IBYØB6(CSP7)
IBYØB7(CSP8)
3
4
7
8
13
14
17
18
74HC374
D1
D2
D3
D4
D5
D6
D7
D8
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
C
OC
1100
2
5
6
9
12
15
16
19
11
1
16
0101
4
758
759

Figure 2

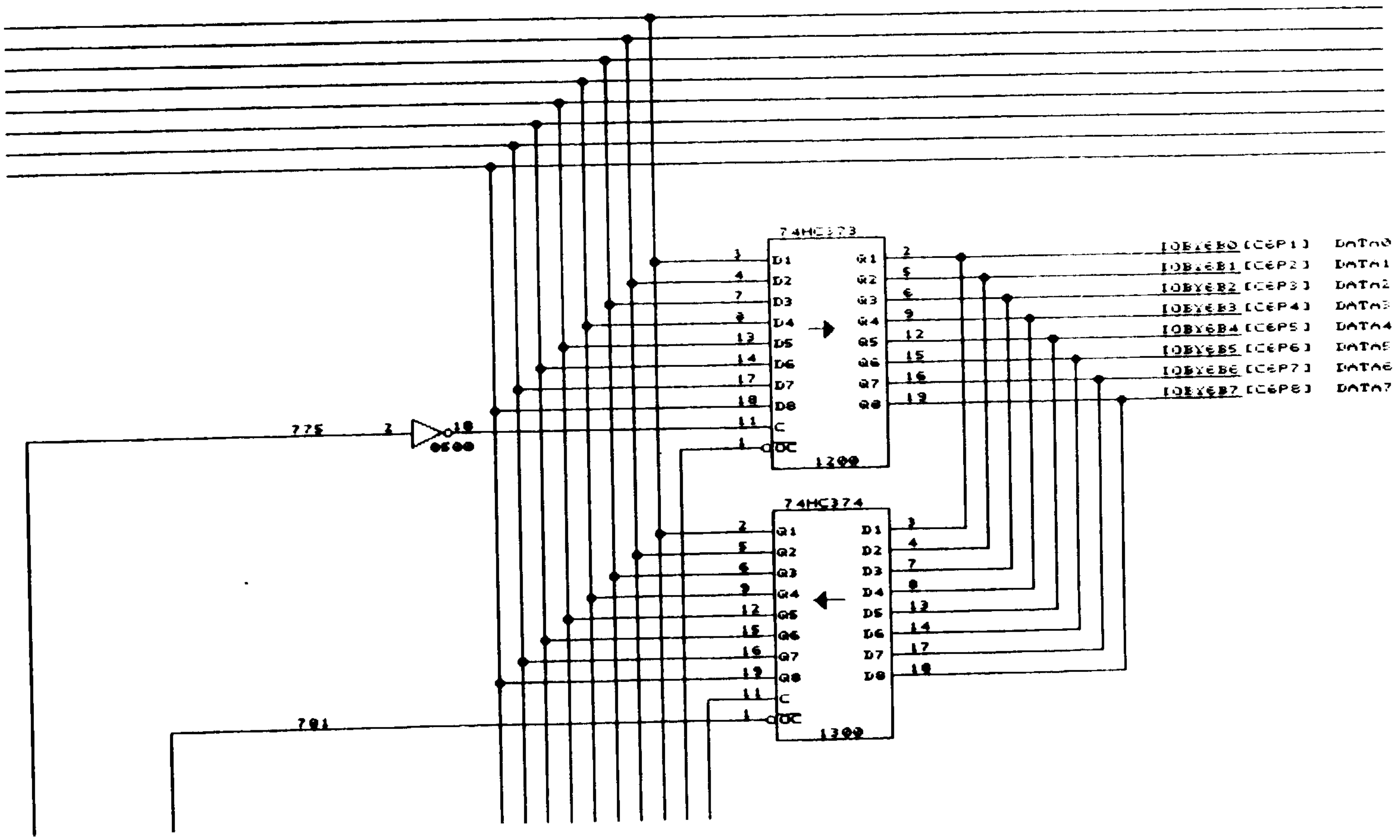

74HC373
D1
D2
D3
D4
D5
D6
D7
D8
C
OC
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
1200
74HC374
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
C
OC
D1
D2
D3
D4
D5
D6
D7
D8
1300
775
2
18
0500
701
IOBYEB0 [CEP1] DATA0
IOBYEB1 [CEP2] DATA1
IOBYEB2 [CEP3] DATA2
IOBYEB3 [CEP4] DATA3
IOBYEB4 [CEP5] DATA4
IOBYEB5 [CEP6] DATA5
IOBYEB6 [CEP7] DATA6
IOBYEB7 [CEP8] DATA7

Figure 3

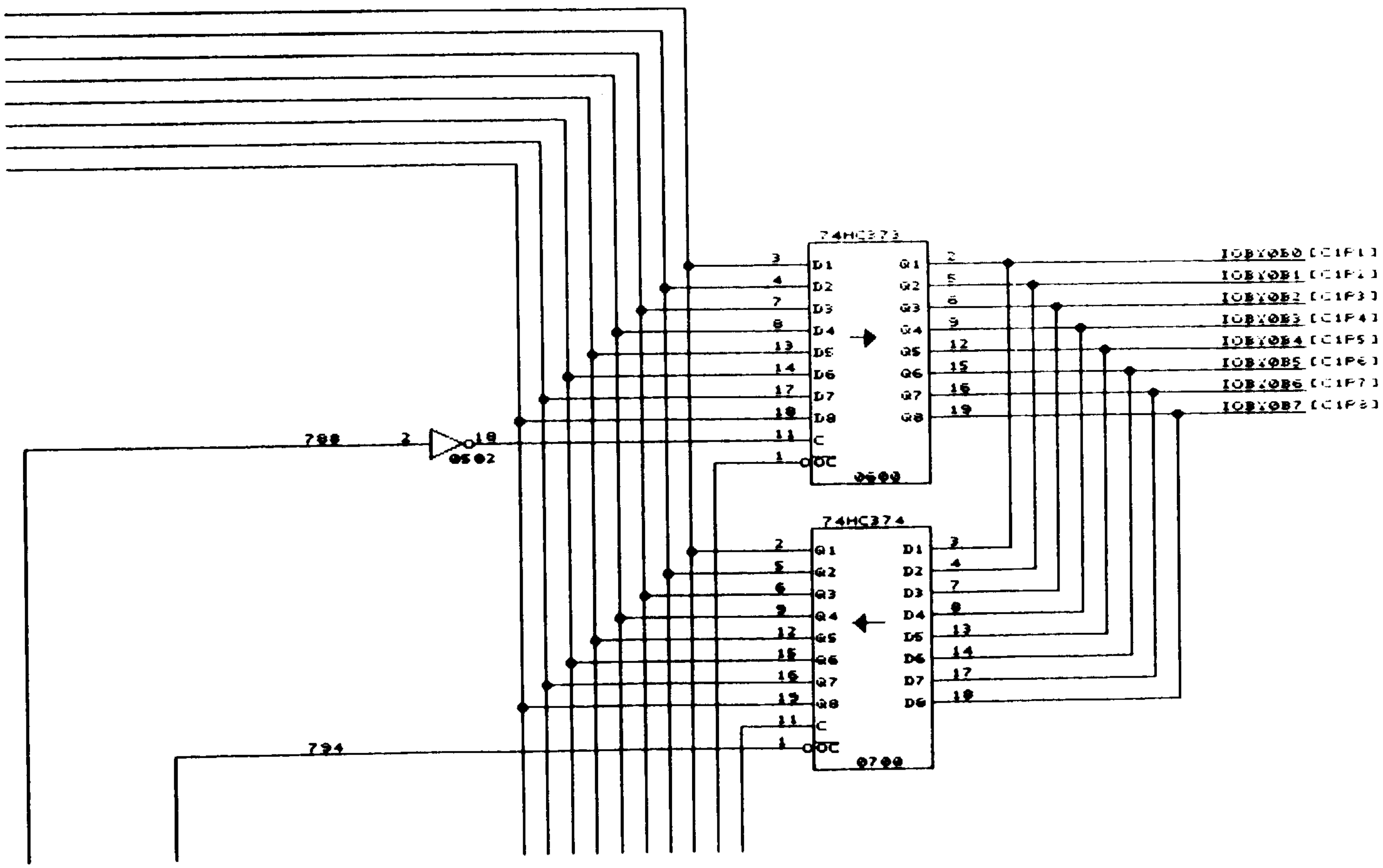

74HC373
D1
D2
D3
D4
D5
D6
D7
D8
C
OC
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
0600
74HC374
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
C
OC
D1
D2
D3
D4
D5
D6
D7
D8
0700
0602
788
794
IOBYØB0 [C1P1]
IOBYØB1 [C1P2]
IOBYØB2 [C1P3]
IOBYØB3 [C1P4]
IOBYØB4 [C1P5]
IOBYØB5 [C1P6]
IOBYØB6 [C1P7]
IOBYØB7 [C1P8]

Figure 4

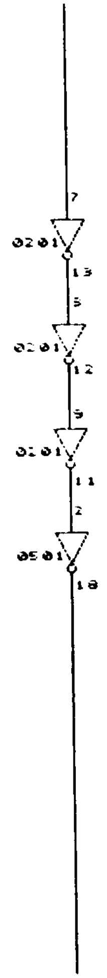


Figure 5

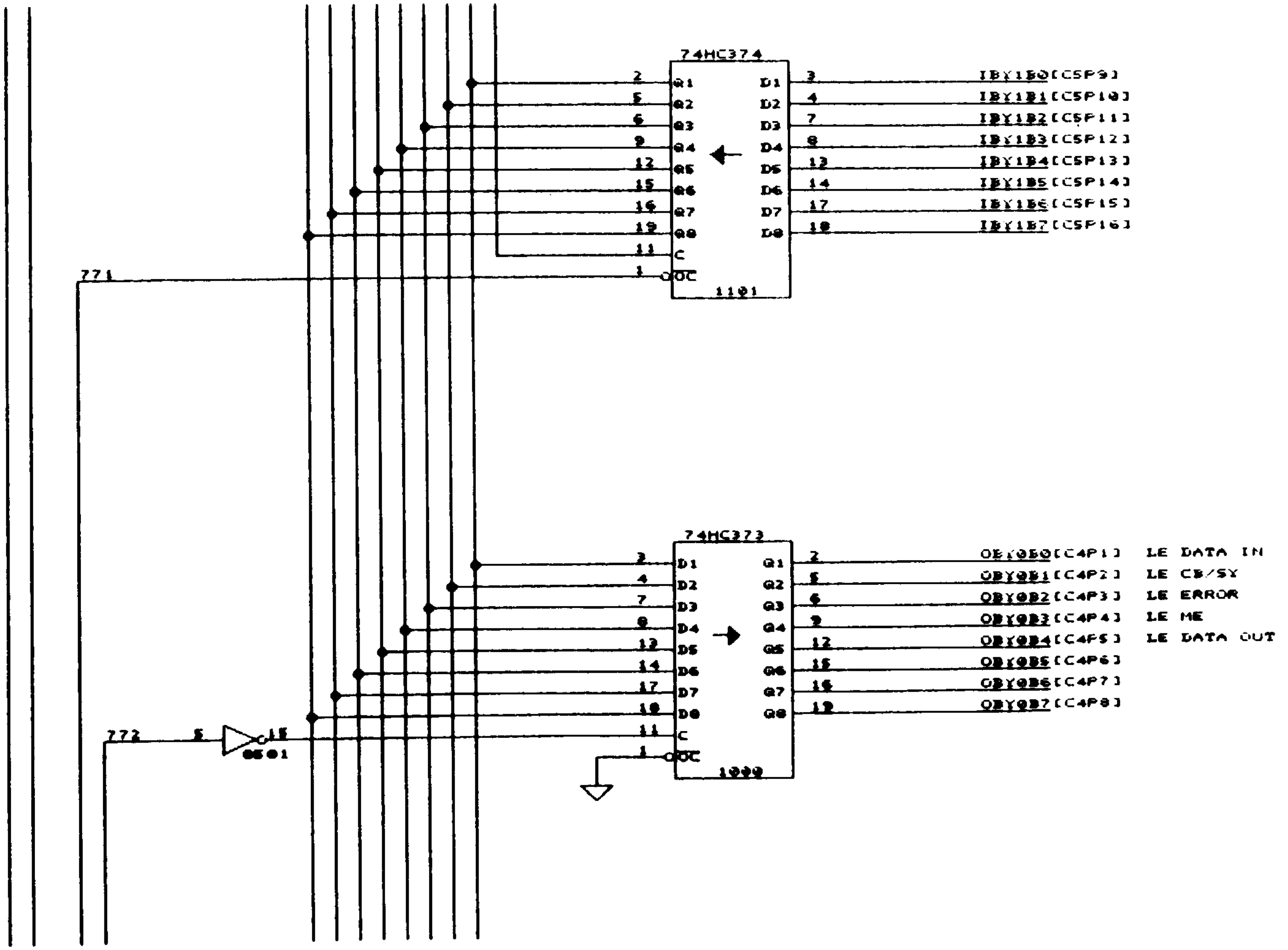

74HC374
Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 C OC
D1 D2 D3 D4 D5 D6 D7 D8
1101
IBY1B0[C5P9]
IBY1B1[C5P10]
IBY1B2[C5P11]
IBY1B3[C5P12]
IBY1B4[C5P13]
IBY1B5[C5P14]
IBY1B6[C5P15]
IBY1B7[C5P16]
771
74HC373
D1 D2 D3 D4 D5 D6 D7 D8 C OC
Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8
1000
OBY0B0[C4P1] LE DATA IN
OBY0B1[C4P2] LE CB/SY
OBY0B2[C4P3] LE ERROR
OBY0B3[C4P4] LE ME
OBY0B4[C4P5] LE DATA OUT
OBY0B5[C4P6]
OBY0B6[C4P7]
OBY0B7[C4P8]
772
0501

Figure 6

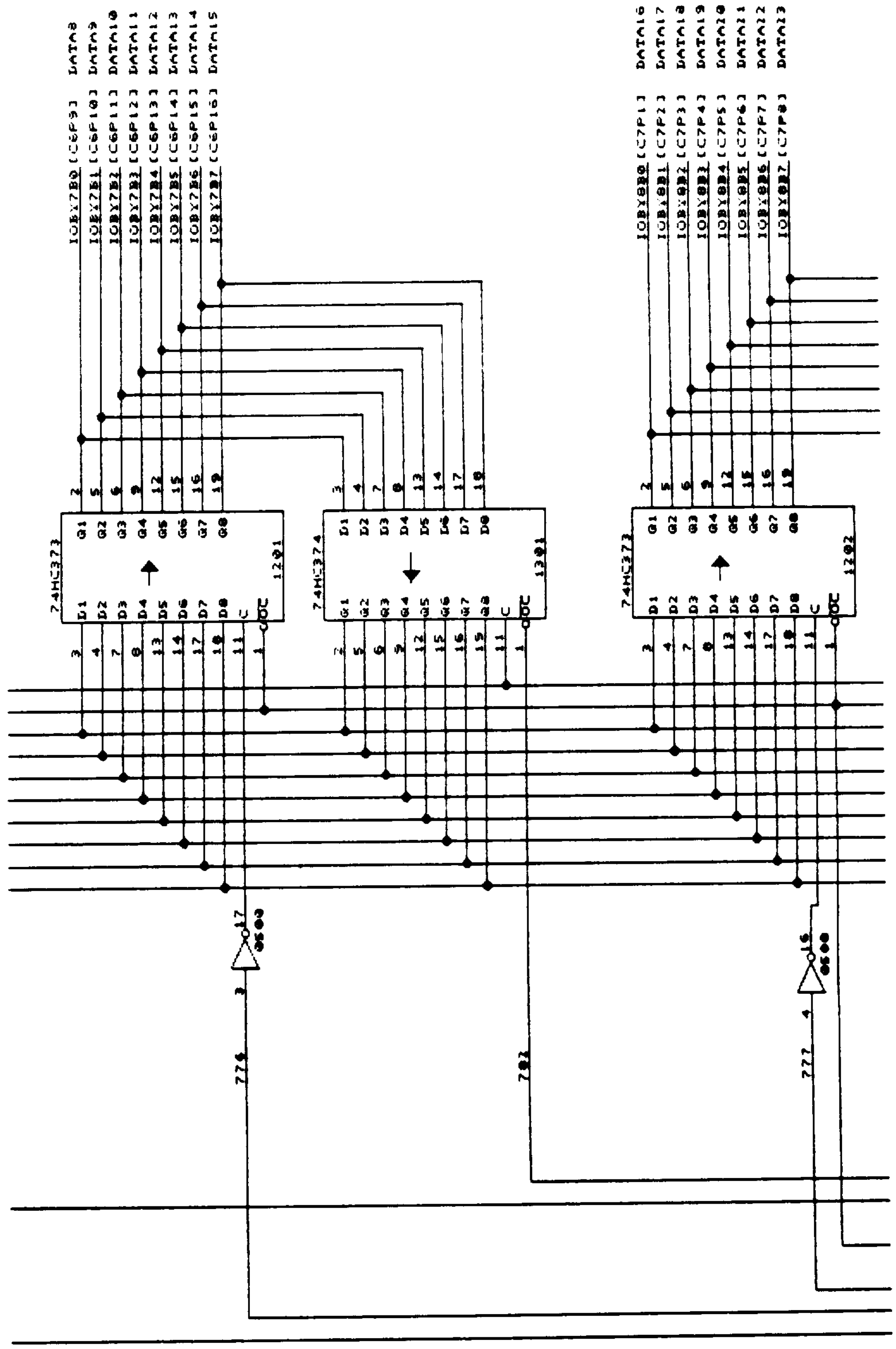

IOBY7B0 [C6P9] DATA8
IOBY7B1 [C6P10] DATA9
IOBY7B2 [C6P11] DATA10
IOBY7B3 [C6P12] DATA11
IOBY7B4 [C6P13] DATA12
IOBY7B5 [C6P14] DATA13
IOBY7B6 [C6P15] DATA14
IOBY7B7 [C6P16] DATA15
IOBY8B0 [C7P1] DATA16
IOBY8B1 [C7P2] DATA17
IOBY8B2 [C7P3] DATA18
IOBY8B3 [C7P4] DATA19
IOBY8B4 [C7P5] DATA20
IOBY8B5 [C7P6] DATA21
IOBY8B6 [C7P7] DATA22
IOBY8B7 [C7P8] DATA23
74HC373
1201
74HC374
1301
74HC373
1202
0500
0500
776
782
777

Figure 7

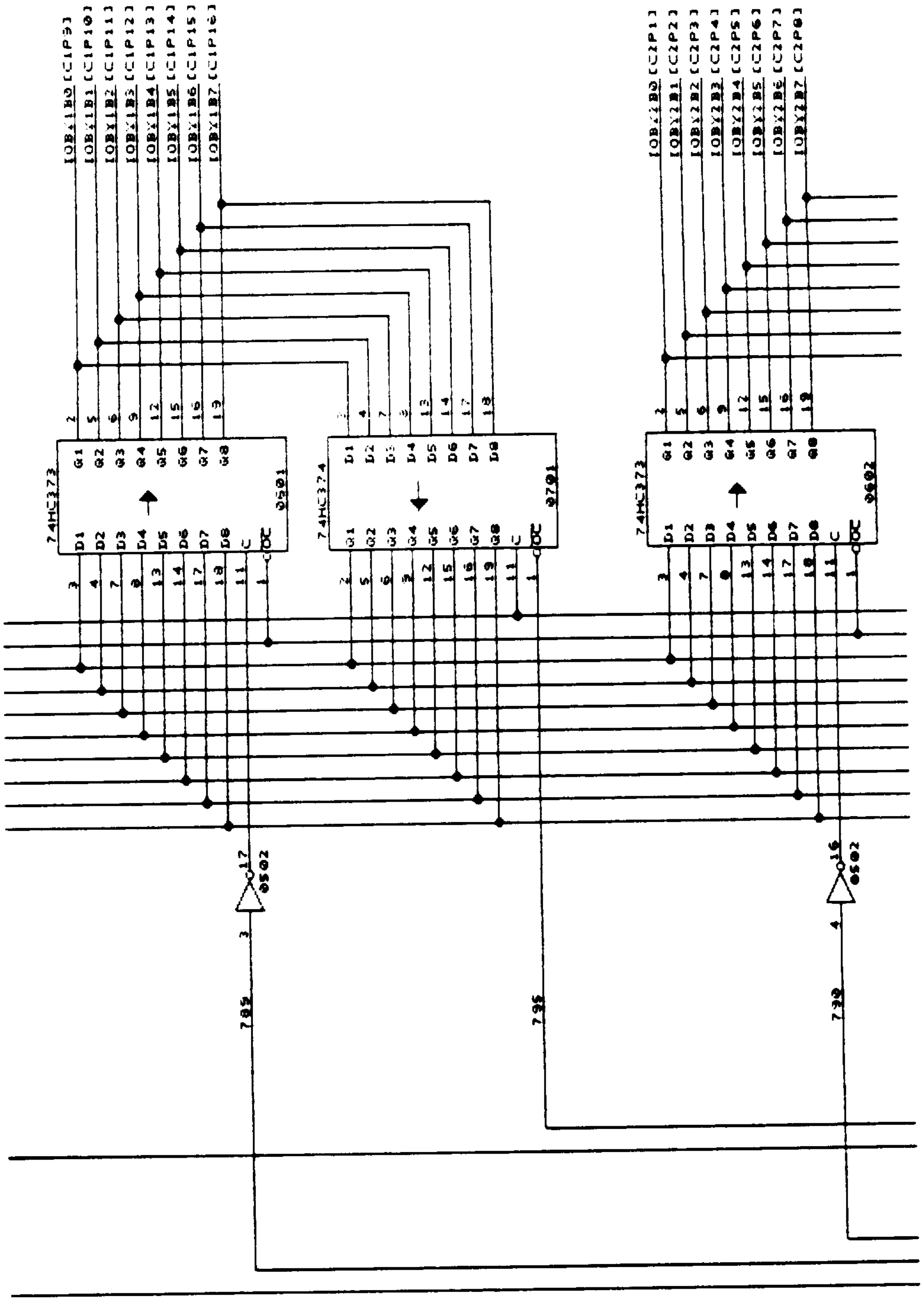

IOBY1B0 [C1P9]
IOBY1B1 [C1P10]
IOBY1B2 [C1P11]
IOBY1B3 [C1P12]
IOBY1B4 [C1P13]
IOBY1B5 [C1P14]
IOBY1B6 [C1P15]
IOBY1B7 [C1P16]
IOBY2B0 [C2P1]
IOBY2B1 [C2P2]
IOBY2B2 [C2P3]
IOBY2B3 [C2P4]
IOBY2B4 [C2P5]
IOBY2B5 [C2P6]
IOBY2B6 [C2P7]
IOBY2B7 [C2P8]
74HC373
0501
G1 G2 G3 G4 G5 G6 G7 G8
2 5 6 9 12 15 16 19
D1 D2 D3 D4 D5 D6 D7 D8 C OC
3 4 7 8 13 14 17 18 11 1
74HC374
0701
D1 D2 D3 D4 D5 D6 D7 D8
3 4 7 8 13 14 17 18
Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 C OC
2 5 6 9 12 15 16 19 11 1
74HC373
0602
Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8
2 5 6 9 12 15 16 19
D1 D2 D3 D4 D5 D6 D7 D8 C OC
3 4 7 8 13 14 17 18 11 1
17
0502
3
785
795
16
0502
4
790

Figure 8

Figure 9

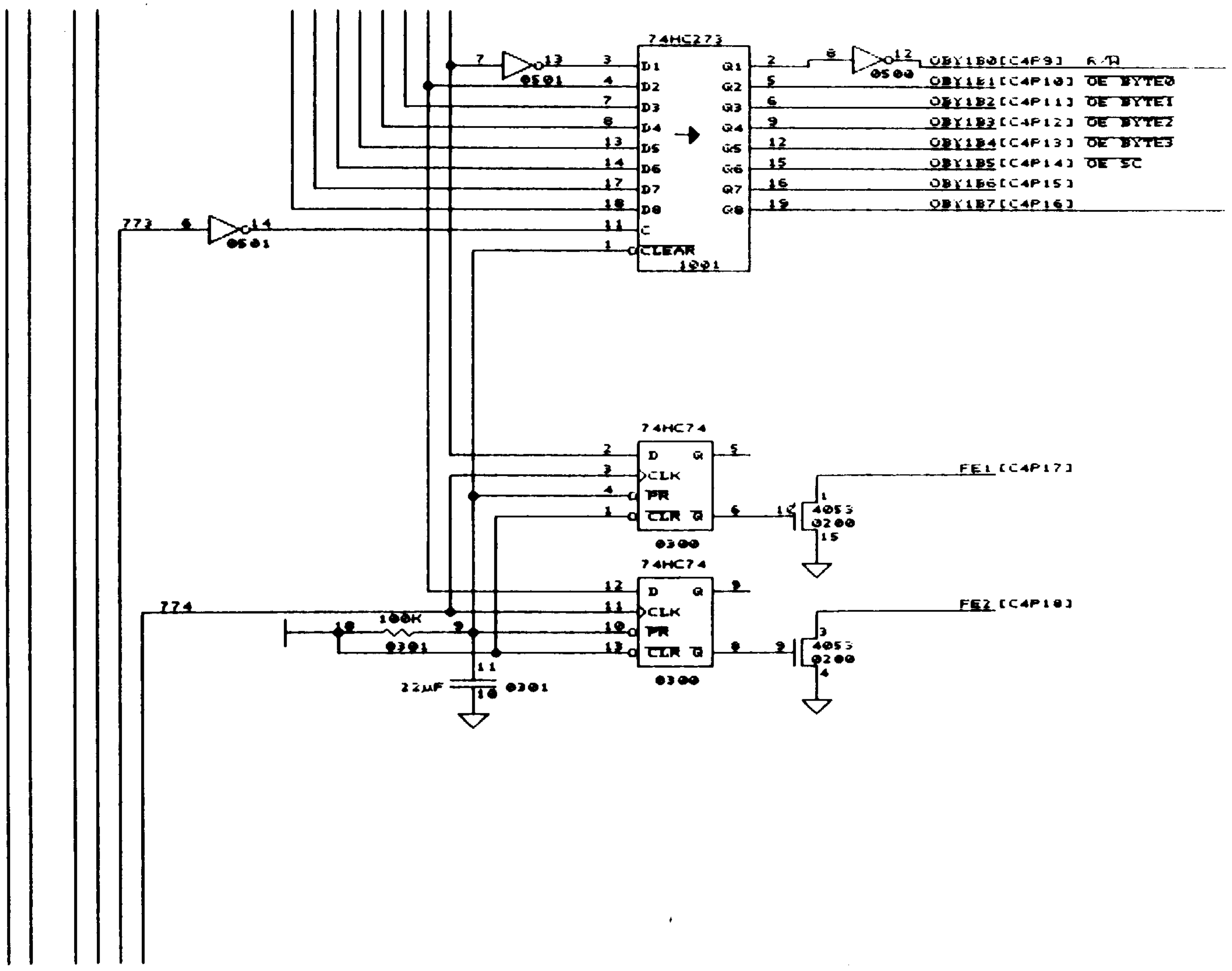
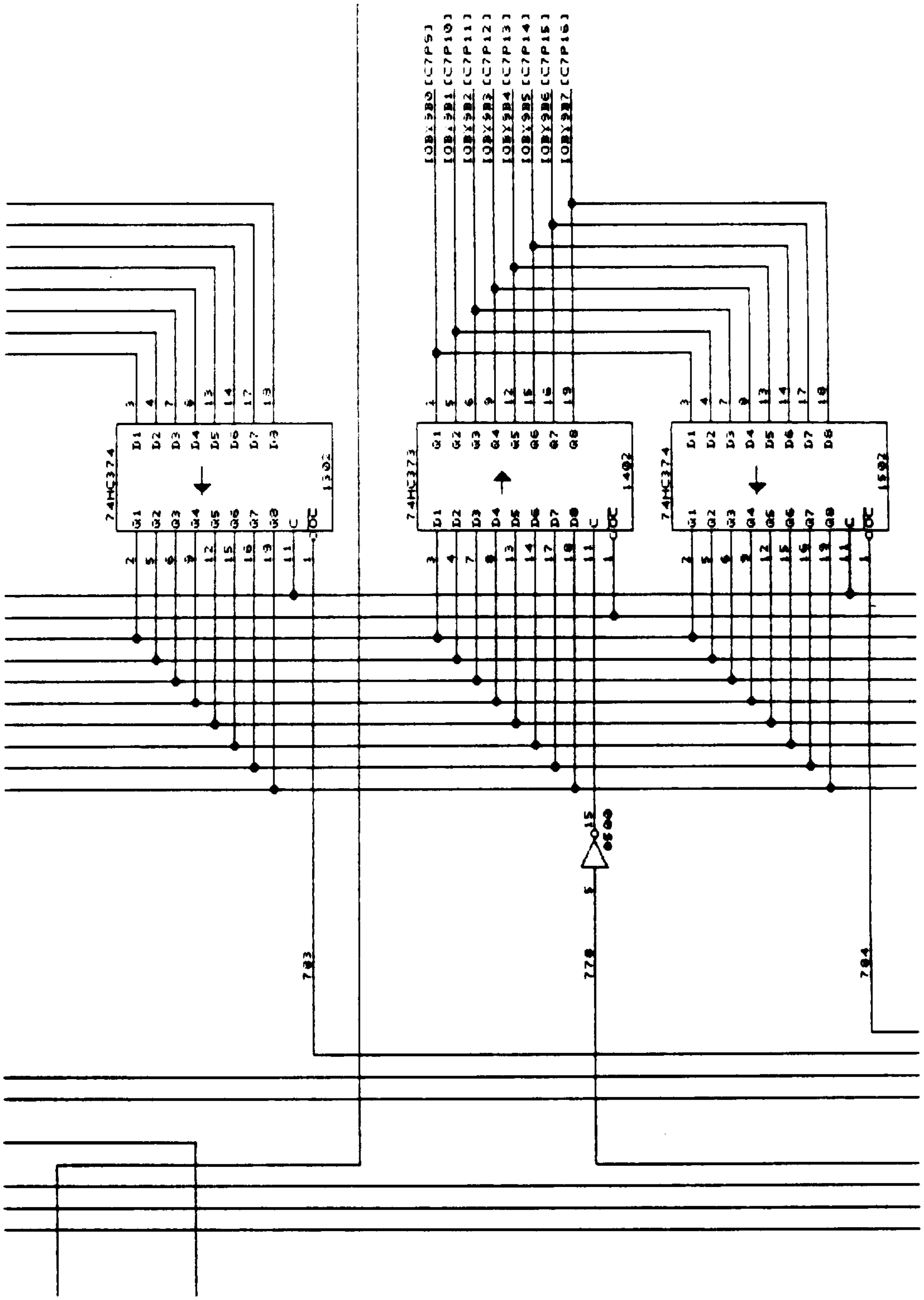

74HC273
D1
D2
D3
D4
D5
D6
D7
D8
C
CLEAR
1001
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
0501
0500
OBY1B0[C4P9] R/W
OBY1B1[C4P10] OE BYTE0
OBY1B2[C4P11] OE BYTE1
OBY1B3[C4P12] OE BYTE2
OBY1B4[C4P13] OE BYTE3
OBY1B5[C4P14] OE SC
OBY1B6[C4P15]
OBY1B7[C4P16]
773
774
74HC74
D
Q
CLK
PR
CLR
0300
4053
0200
FE1 [C4P17]
FE2 [C4P18]
100K
0301
22μF

Figure 10



IOBX9B0 [C7P9]
IOBX9B1 [C7P10]
IOBX9B2 [C7P11]
IOBX9B3 [C7P12]
IOBX9B4 [C7P13]
IOBX9B5 [C7P14]
IOBX9B6 [C7P15]
IOBX9B7 [C7P16]
74HC374
1302
74HC373
1402
74HC374
1502
D1
D2
D3
D4
D5
D6
D7
D8
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
C
OC
0500
15
5
703
770
704

Figure 11

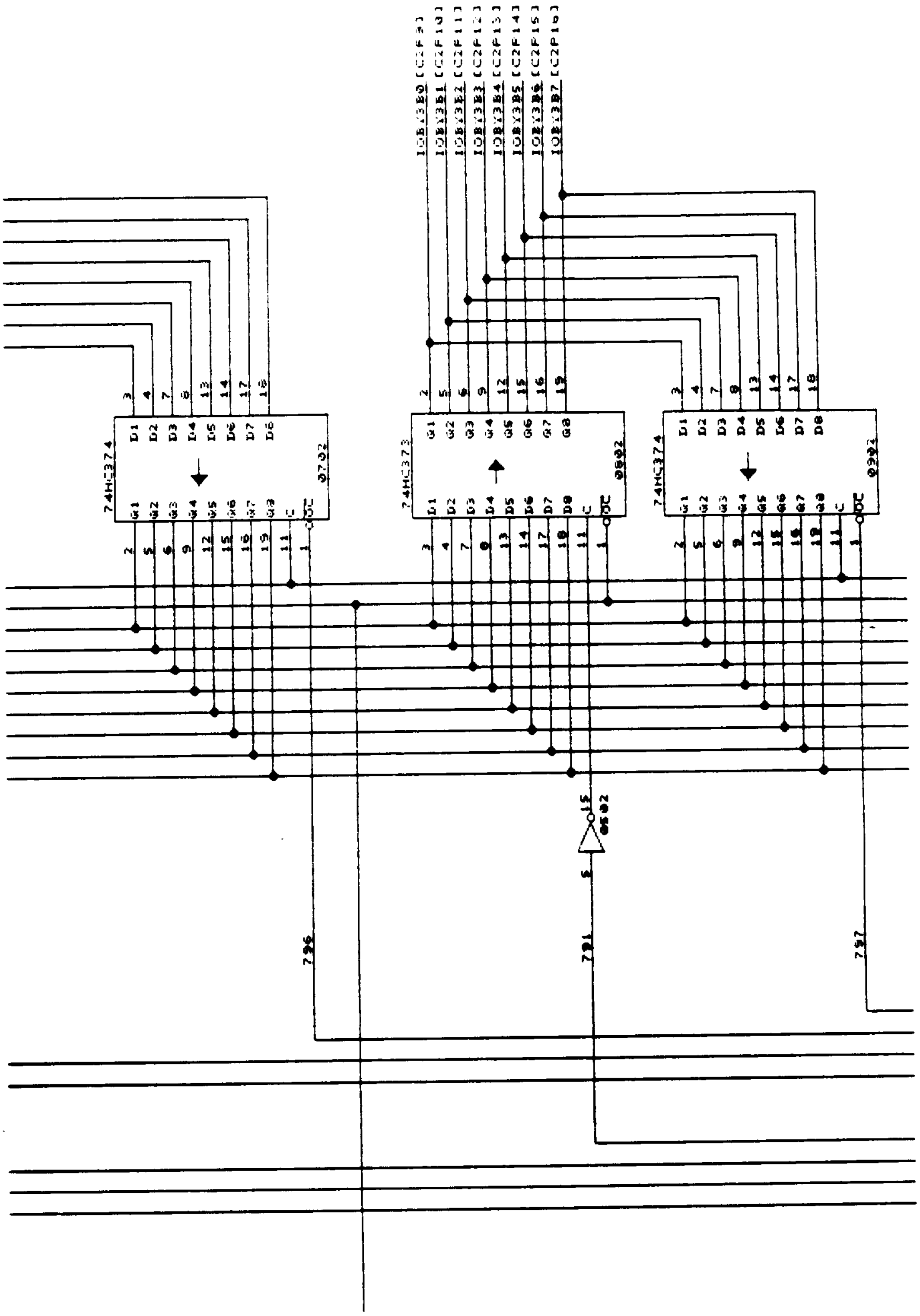

IOBX3B0 [C2F9]
IOBX3B1 [C2F10]
IOBX3B2 [C2F11]
IOBX3B3 [C2F12]
IOBX3B4 [C2F13]
IOBX3B5 [C2F14]
IOBX3B6 [C2F15]
IOBX3B7 [C2F16]
74HC374
0702
74HC373
0802
74HC374
0902
0602
D1
D2
D3
D4
D5
D6
D7
D8
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
C
OC
796
791
797

Figure 12

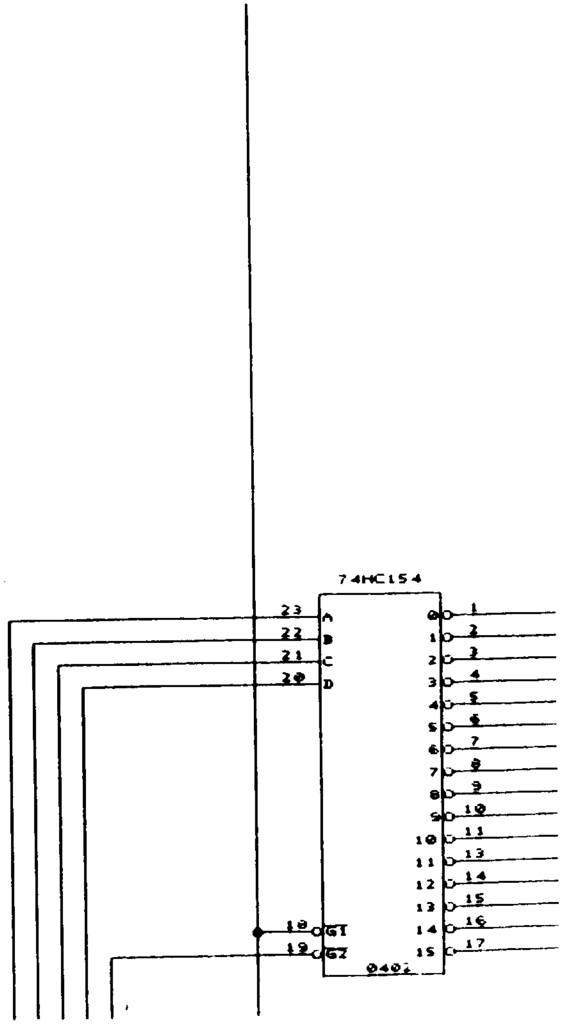

74HC154
23
22
21
20
A
B
C
D
18
19
G1
G2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
1
2
3
4
5
6
7
8
9
10
11
13
14
15
16
17
0402

Figure 13

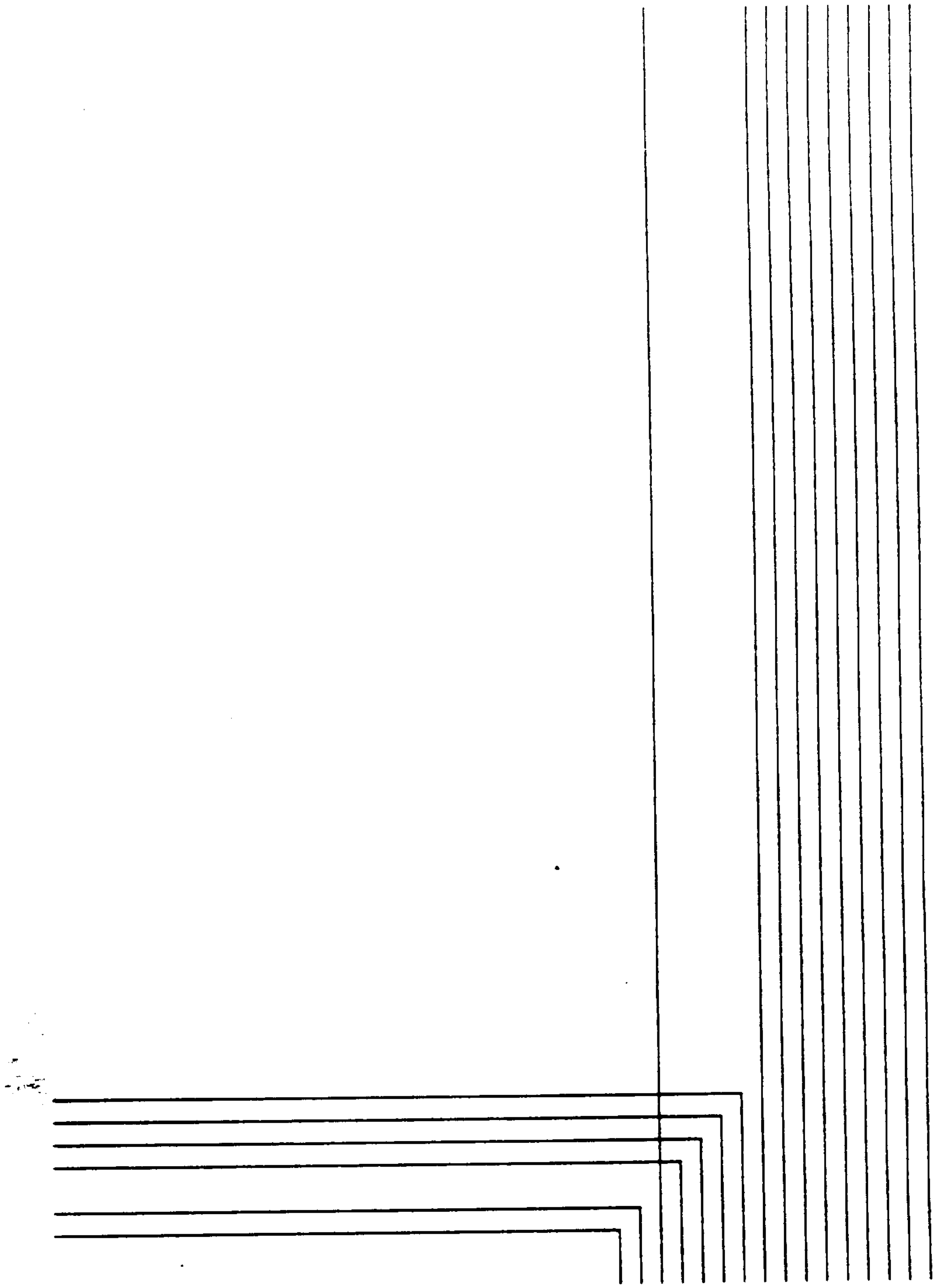


Figure 14

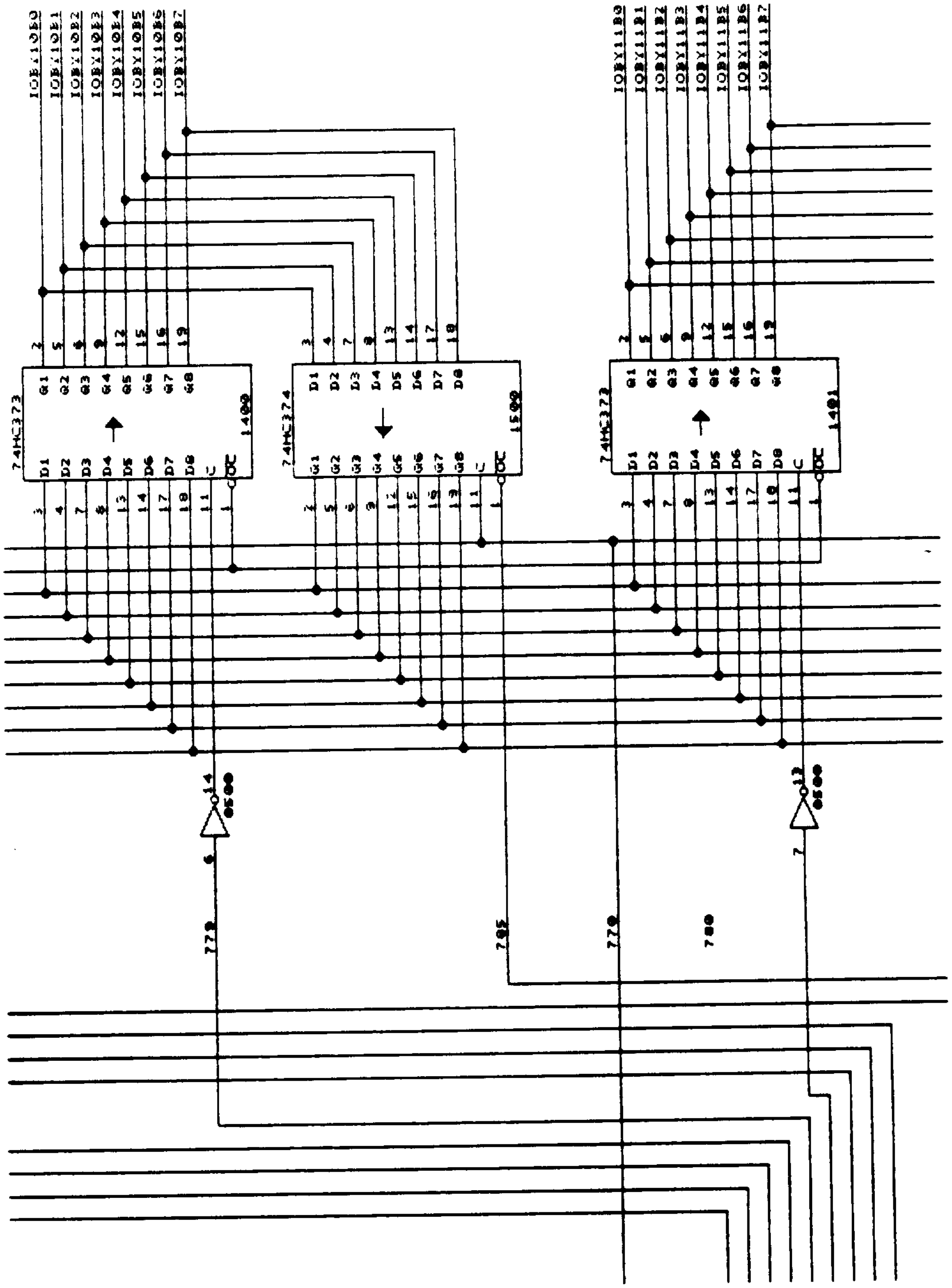
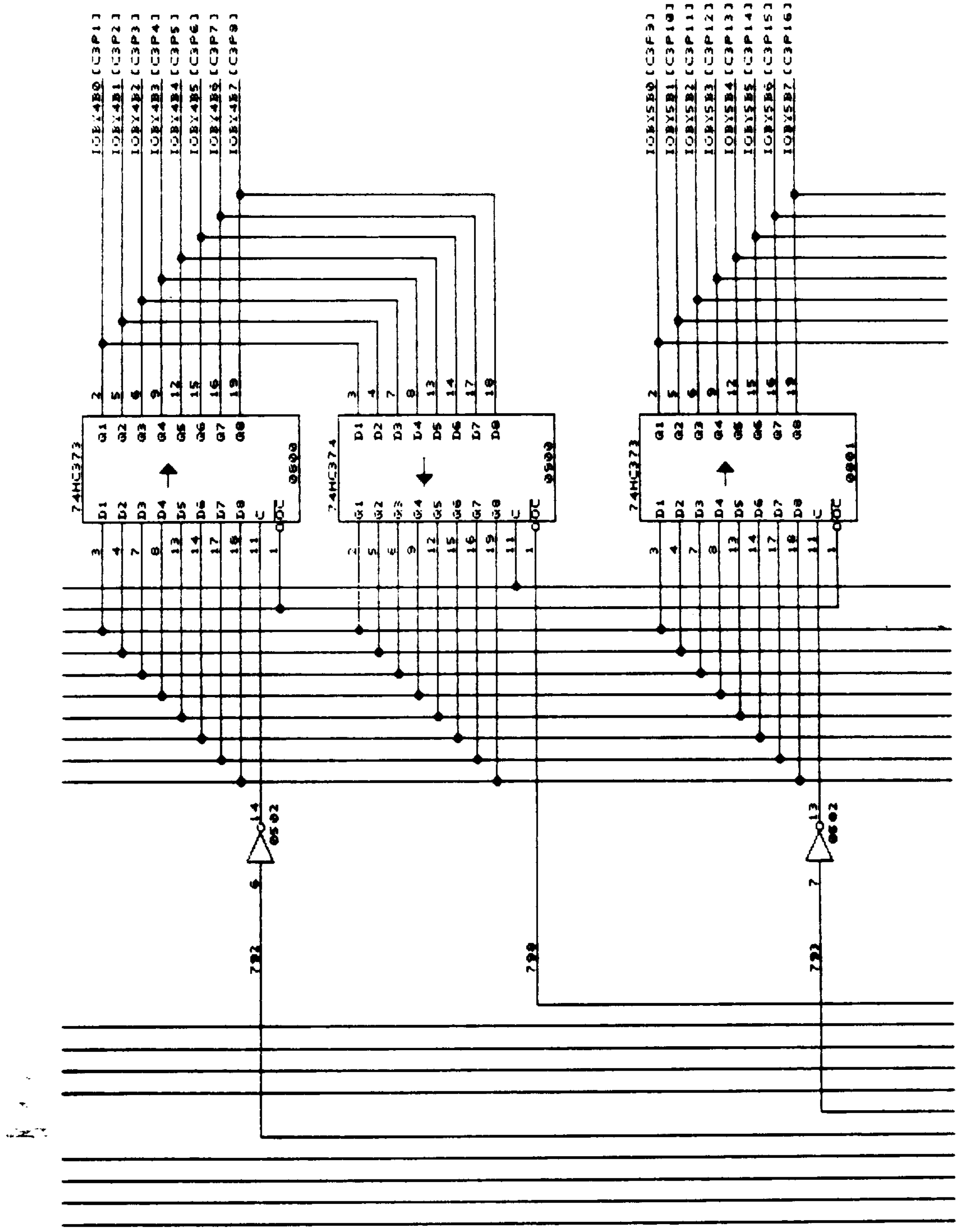

IOBY10B0
IOBY10B1
IOBY10B2
IOBY10B3
IOBY10B4
IOBY10B5
IOBY10B6
IOBY10B7
IOBY11B0
IOBY11B1
IOBY11B2
IOBY11B3
IOBY11B4
IOBY11B5
IOBY11B6
IOBY11B7
74HC373
1400
74HC374
1500
74HC373
1401
0500
0500
770
785
770
780

Figure 15



IOBY4B0 (C3P1)
IOBY4B1 (C3P2)
IOBY4B2 (C3P3)
IOBY4B3 (C3P4)
IOBY4B4 (C3P5)
IOBY4B5 (C3P6)
IOBY4B6 (C3P7)
IOBY4B7 (C3P8)
IOBY5B0 (C3F9)
IOBY5B1 (C3P10)
IOBY5B2 (C3P11)
IOBY5B3 (C3P12)
IOBY5B4 (C3P13)
IOBY5B5 (C3P14)
IOBY5B6 (C3P15)
IOBY5B7 (C3P16)
74HC373
74HC374
74HC373
0500
0500
0501
0502
0502
792
790
792
Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8
D1 D2 D3 D4 D5 D6 D7 D8
C
OC

Figure 16

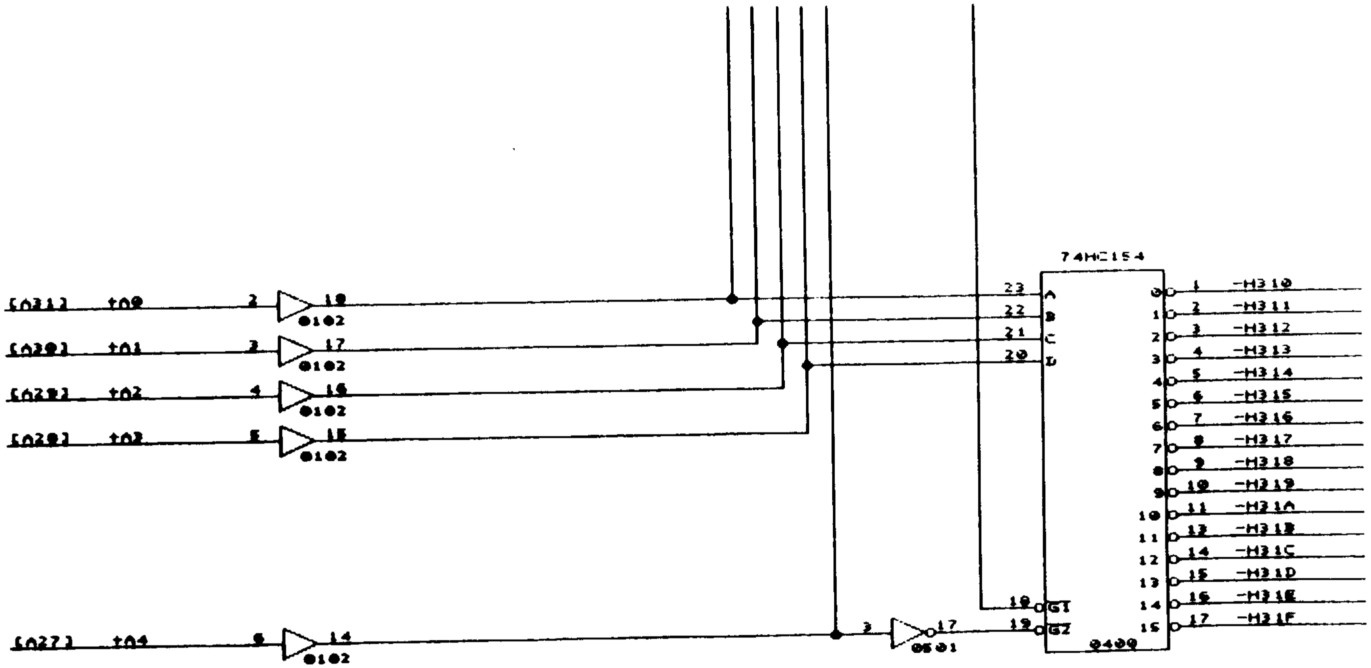

[A31] +AØ
2
1Ø
Ø1Ø2
[A3Ø] +A1
3
17
Ø1Ø2
[A29] +A2
4
16
Ø1Ø2
[A28] +A3
5
15
Ø1Ø2
[A27] +A4
6
14
Ø1Ø2
3
17
Ø5Ø1
74HC154
23 A
22 B
21 C
2Ø D
18 G1
19 G2
Ø4ØØ
Ø 1 −H31Ø
1 2 −H311
2 3 −H312
3 4 −H313
4 5 −H314
5 6 −H315
6 7 −H316
7 8 −H317
8 9 −H318
9 1Ø −H319
1Ø 11 −H31A
11 13 −H31B
12 14 −H31C
13 15 −H31D
14 16 −H31E
15 17 −H31F

Figure 17

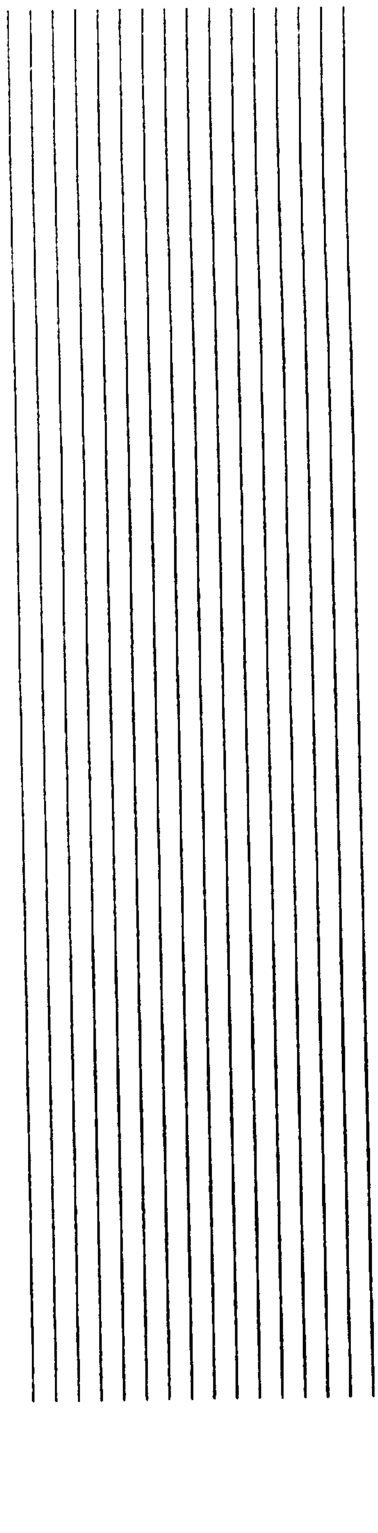


Figure 18

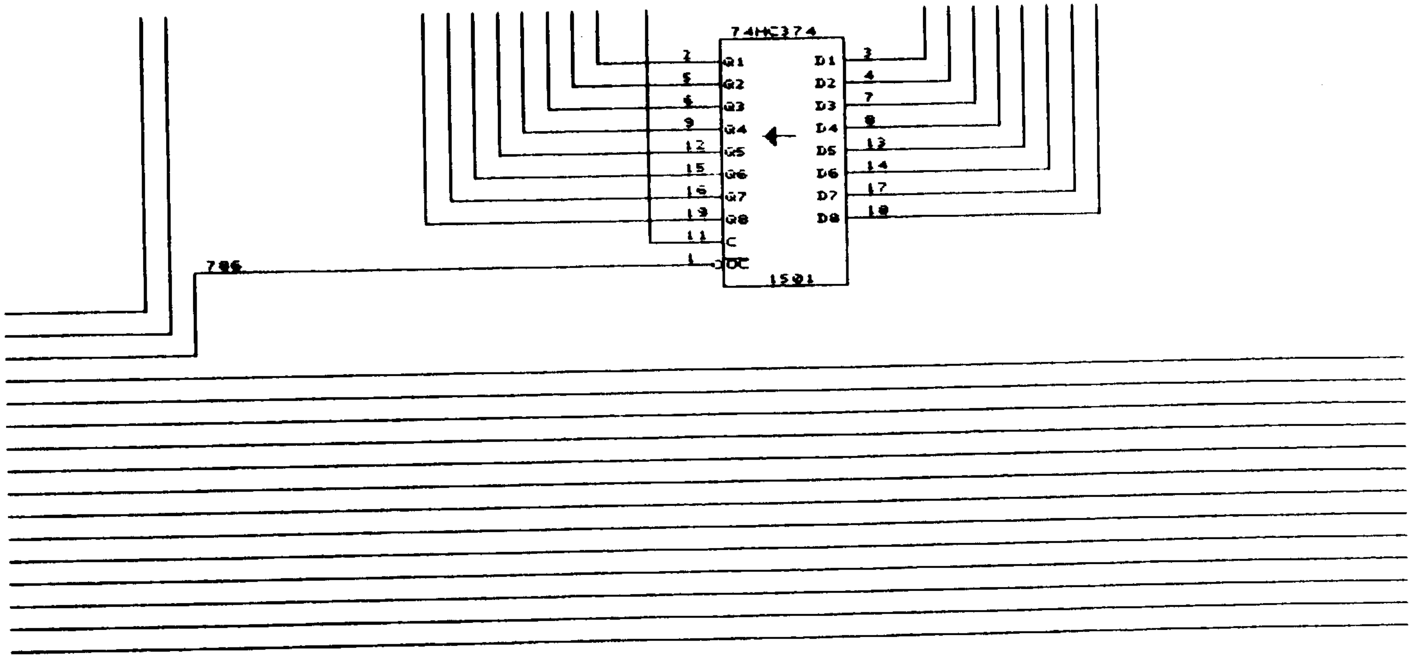

74HC374
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
C
OC
D1
D2
D3
D4
D5
D6
D7
D8
2
5
6
9
12
15
16
19
11
1
3
4
7
8
13
14
17
18
1501
786

Figure 19

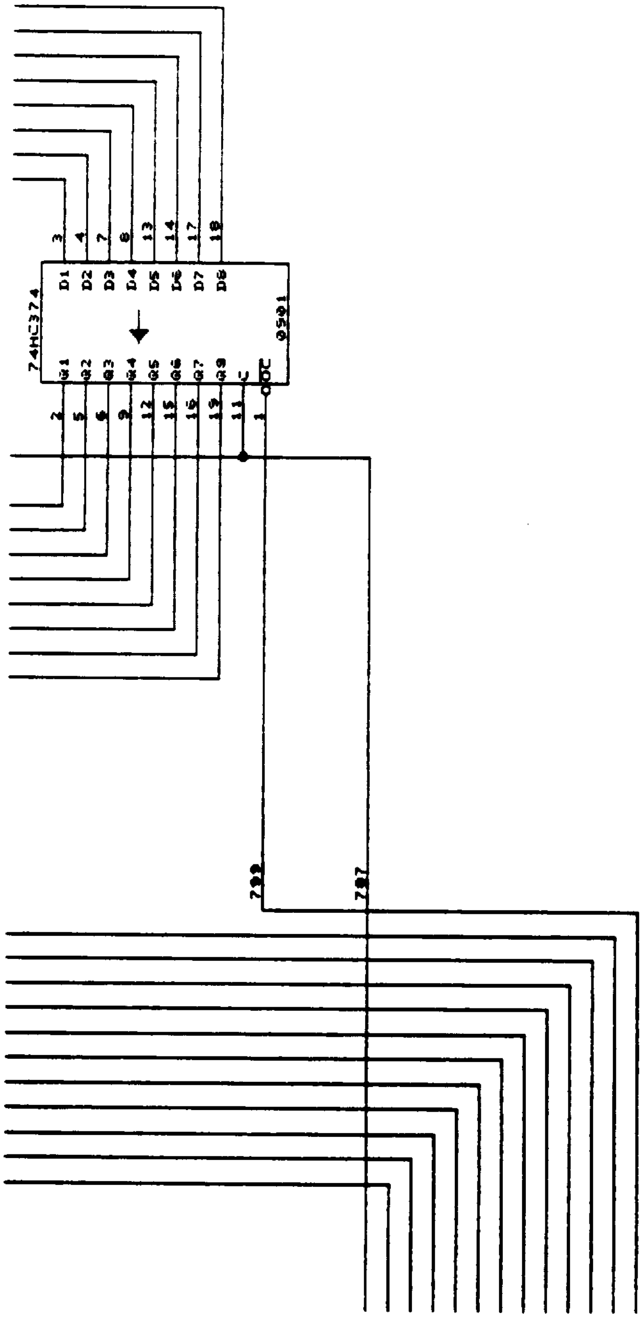

74HC374
U501
D1
D2
D3
D4
D5
D6
D7
D8
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
C
OC
3
4
7
8
13
14
17
18
2
5
6
9
12
15
16
19
11
1
799
707

Figure 20