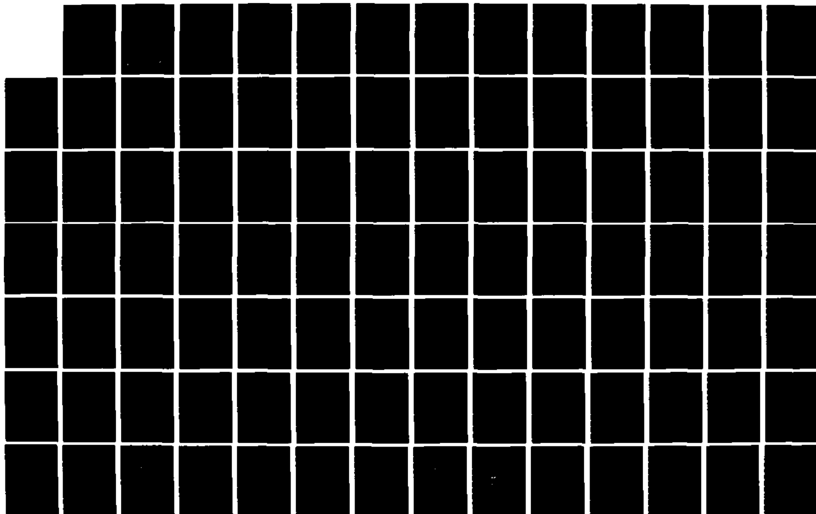
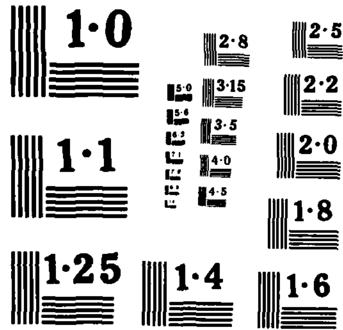
AD-A158 370 BULK CMOS VLSI TECHNOLOGY STUDIES PART 5 THE DESIGN AND IMPLEMENTATION OF..(U) MISSISSIPPI STATE UNIV MISSISSIPPI STATE DEPT OF ELECTRICAL E.. 1/2

UNCLASSIFIED J D TROTTER ET AL. 17 JUN 85 F/G 14/2 NL






NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

ARO 19839.1-EL-A

(1)

AD-A158 370

FINAL REPORT

CONTRACT DAAG29-82-K-0167

# BULK CMOS VLSI TECHNOLOGY STUDIES

# PART 5: THE DESIGN AND IMPLEMENTATION OF A HIGH SPEED INTEGRATED CIRCUIT FUNCTIONAL TESTER


Principal Investigator

J. Donald Trotter

Associate Investigator

Hoyle Dwayne Robbins

Mississippi State University
Department of Electrical Engineering
Mississippi State, Mississippi 39762


for
Defense Advance Research Projects Agency
1400 Wilson Ave.
Arlington, VA 22209

for

U. S. Army Research Office
P. O. Box 11211
Research Triangle Park, NC 27709

June 17, 1985


DTIC FILE COPY

DTIC ELECTE AUG 2 7 1985 S D

This document has been approved for public release and sale; its distribution is unlimited.

85 8 23 041

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

| Field | Entry |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION | Unclassified |
| 1b. RESTRICTIVE MARKINGS | |
| 2a. SECURITY CLASSIFICATION AUTHORITY | |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 3. DISTRIBUTION/AVAILABILITY OF REPORT | This document has been approved for public release and sale; its distribution is unlimited. |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | |
| 5. MONITORING ORGANIZATION REPORT NUMBER(S) | |
| 6a. NAME OF PERFORMING ORGANIZATION | Miss. State University |
| 6b. OFFICE SYMBOL (If applicable) | Electrical |
| 7a. NAME OF MONITORING ORGANIZATION | Georgia Institute of Technology |
| 6c. ADDRESS (City, State and ZIP Code) | Drawer EE<br>Miss. State, MS 39762 |
| 7b. ADDRESS (City, State and ZIP Code) | 206 O'Keefe Building<br>Atlanta, GA 30332 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION | U.S. Army Research Office |
| 8b. OFFICE SYMBOL (If applicable) | |
| 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER | DAAG29-82-K-0167 |
| 8c. ADDRESS (City, State and ZIP Code) | P.O. Box 12211<br>Research Triangle Park, NC 27709-2211 |
| 10. SOURCE OF FUNDING NOS. | PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO. |
| 11. TITLE (Include Security Classification) | Bulk CMOS VLSI Technology Studies |
| 12. PERSONAL AUTHOR(S) | J. Donald Trotter, Hoyle Dwayne Robbins |
| 13a. TYPE OF REPORT | Final |
| 13b. TIME COVERED | FROM 8-82 TO 2-85 |
| 14. DATE OF REPORT (Yr., Mo., Day) | June 10, 1985 |
| 15. PAGE COUNT | |
| 16. SUPPLEMENTARY NOTATION | |

| 17. COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB. GR. |
| | | |
| | | |

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

(Subtitle)
Design and Implementation of a High Speed Integrated Circuit Functional Tester

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

See page iii of Thesis.

| Field | Entry |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐ |
| 21. ABSTRACT SECURITY CLASSIFICATION | Unclassified |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL | |
| 22b. TELEPHONE NUMBER (Include Area Code) | |
| 22c. OFFICE SYMBOL | |

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

FINAL REPORT

CONTRACT DAAG29-82-K-0167

BULK CMOS VLSI TECHNOLOGY STUDIES

PART 5: THE DESIGN AND IMPLEMENTATION OF A HIGH SPEED

INTEGRATED CIRCUIT FUNCTIONAL TESTER


Principal Investigator

J. Donald Trotter

Associate Investigator

Hoyle Dwayne Robbins

Mississippi State University
Department of Electrical Engineering
Mississippi State, Mississippi 39762


for
Defense Advance Research Projects Agency
1400 Wilson Ave.
Arlington, VA 22209

for

U. S. Army Research Office
P. O. Box 11211
Research Triangle Park, NC 27709

per Form 50 on file.

June 17, 1985

A-1

# ABSTRACT

Hoyle Dwayne Robbins, Master of Science, December 1984

Major: Electrical Engineering, Department of Electrical Engineering

Title of Thesis: The Design and Implementation of a High-Speed Integrated Circuit Functional Tester

Directed by: Dr. William A. Hornfeck

Pages in Thesis: 183 Words in Abstract: 260

## ABSTRACT

This thesis project discusses the design and implementation of a functional tester to be used in a university laboratory facility for integrated circuit development. The following tester capabilities were desired:

1) 100KHz - 10MHz TEST FREQUENCY
2) 2K x 64Bit TEST VECTOR SIZE
3) FOUR USER PROGRAMMABLE CLOCKS
4) TEST VECTOR INPUT USING HIGH-LEVEL LANGUAGE
5) TEST DATA MANIPULATION USING A HIGH-LEVEL LANGUAGE

The functional tester receives test vector data from a Hewlett Packard HP9920A computer and loads this data into functional tester buffer memory. After the data is down-loaded, including certain operational information such as test clock frequency, programmable clock waveform information, data direction control, etc., the tester initiates the test using "random-logic" control circuitry to achieve

the desired high speeds. The random-logic control circuitry indicates the completion of the test, at which time the resultant data, stored in buffer memory, is up-loaded to the HP9920A for processing. This design approach to a functional tester for laboratory use differs from --and improves upon-- previous methods, in that random-logic control circuitry is used during the test phase to provide greater operating speeds than systems which use microprocessor-control for the complete test. Also, a computer with a high-level language is used for storing and transferring the test vectors and processing the resultant data. Data pipeline registers are used during the test phase so that the speed of the system is not completely dependent on the access time of the memory elements used for buffer memory. Programmable clocks, synchronized to the test clock, are integrated into the system, programmable on both their low-to-high and high-to-low transitions.

# TABLE OF CONTENTS

ABSTRACT . . . . . . . . . . . . . . . . . . . . . . . . . . iii

CHAPTER

I. INTRODUCTION . . . . . . . . . . . . . . . . . . . . 1

II. HARDWARE DESIGN . . . . . . . . . . . . . . . . . . 8

Computer Interface . . . . . . . . . . . . . . . . . 15
Memory Boards . . . . . . . . . . . . . . . . . . . 19
Programmable Clocks . . . . . . . . . . . . . . . . 23
Test-Phase Control . . . . . . . . . . . . . . . . . 27
Device-Under-Test Interface . . . . . . . . . . . . 33

III. SOFTWARE DESIGN . . . . . . . . . . . . . . . . . . 40

Assembly Language Programs . . . . . . . . . . . . . 41
PASCAL Programs . . . . . . . . . . . . . . . . . . 44

IV. OPERATION . . . . . . . . . . . . . . . . . . . . . 51

Electrical Connections . . . . . . . . . . . . . . . 52
Power-Up Initialization . . . . . . . . . . . . . . 53
Test Execution . . . . . . . . . . . . . . . . . . . 55
Results Evaluation . . . . . . . . . . . . . . . . . 60

V. EVALUATION AND DISCUSSION . . . . . . . . . . . . . 61

APPENDICES

Appendix A: Test Station . . . . . . . . . . . . . . 67

Appendix B: Functional Tester Schematics . . . . . . 69

System Configuration . . . . . . . . . . . . . . . 70
Computer Interface . . . . . . . . . . . . . . . . 71
Memory Board 1 . . . . . . . . . . . . . . . . . . 75
Memory Board 2 . . . . . . . . . . . . . . . . . . 81
Programmable Clocks . . . . . . . . . . . . . . . . 86
Test-Phase Control . . . . . . . . . . . . . . . . 91
DUT Interface . . . . . . . . . . . . . . . . . . . 94

Table of Contents (Cont'd)

Appendix C: Assembly Language Programs . . . . . . . . 101

Utility Program Flowchart . . . . . . . . . . . . . . . 102
Utility Program . . . . . . . . . . . . . . . . . . . . 104
Control Program Flowchart . . . . . . . . . . . . . . . 106
Control Program . . . . . . . . . . . . . . . . . . . . 111

Appendix D: PASCAL Program . . . . . . . . . . . . . . . 115

Flowcharts . . . . . . . . . . . . . . . . . . . . . . 116
PASCAL Program . . . . . . . . . . . . . . . . . . . . 137

Appendix E: Test Results . . . . . . . . . . . . . . . . 168

2114 RAM Test . . . . . . . . . . . . . . . . . . . . . 169
7404 Inverter Test . . . . . . . . . . . . . . . . . . 171
7482 Binary Adder Test . . . . . . . . . . . . . . . . 174

REFERENCES . . . . . . . . . . . . . . . . . . . . . . . . . 176

# CHAPTER I

# INTRODUCTION

Rapidly advancing semiconductor fabrication technology makes it possible to integrate more sophisticated electronic systems (one-million or more transistors) on a single semiconductor integrated circuit. Not only are very large scale integration (VLSI) circuits becoming more complex and more prevalent, but the advent of very high speed integrated circuits (VHSIC) is increasing the speeds of these circuits tremendously. Some of the most advanced VLSI designs are being produced through research in the academic environment. The furthur development of computer-aided design (CAD) tools makes it possible for universities to use computer graphics systems with supporting software to undertake the design and development of these complex circuits. One of the problems facing universities is the ability to adequately test these VLSI designs. Most automated test equipment is very expensive and is geared toward industries which produce mass quantities of integrated circuits, rather than a facility which may produce only a few designs a year.

The testing requirements may be broken down into two categories - parameter testing and functional testing, although the timing parameters of a device may be determined through functional testing.

One use of parameter testing is determining the nature of the defect when integrated circuits are discovered to be faulty. Since most universities "ship-out" their VLSI designs to silicon foundries to be manufactured, the university has little control over the fabrication techniques. This information is useful to the designer, however, since differing design tehniques may produce differing yield. Furthurmore, the particular nature of the defect is often indicative of poor design methods, rather than purely fabrication faults. The primary use of parameter testing for the VLSI designer is in gathering information about different design patterns, which in turn produces better design guidelines. An example of this is the research currently being done using based on the 1.2 micron CMOS technology. This technology is new enough that the design rules have not been verified, and the information from D.C. parameter testing is required to evaluate alternative design rules.

One use of functional testing is in differentiating faulty integrated circuits from operable circuits at the silicon wafer level. This is important even though universities may use silicon foundries to manufacture their circuits, since the functionally good slice must be identified for packaging. The other important use of functional testing is measuring the transient response of an integrated circuit to aid in design. Through testing at increasing speeds until erroneous resultant data is produced, the maximum operating speed of a device can be determined. The propagation

delay and data setup delay for a device can be determined by varying the functional tester data output enable and data input strobe pulses until erroneous data is produced.

The project undertaken at Mississippi State University was the construction of a complete integrated circuit test station which centered around the Hewlett Packard 9920A computer. The test station was comprised of the following equipment:

HP9920A COMPUTER

HP9133B HARD/FLOPPY DISK DRIVE

HP82913 VIDEO MONITOR

HP KEYBOARD

HP7475A PLOTTER

HP4145 PARAMETER ANALYZER

HP4275A LCR METER

HP3455A DIGITAL VOLTMETER

IDS P-132 PRISM PRINTER

MATRIX SYSTEMS 3111 COAXIAL SWITCHING MATRIX

ELECTROGLAS 900 SEMI-AUTOMATIC PROBE STATION

MSU FUNCTIONAL TESTER

Appendix A presents pictures of the completed test station. Also included in Appendix A is a system diagram showing the interconnections between the various components of the system outlined above.

This thesis project is concerned with the design of the Functional Tester. The interconnection of the parametric test instruments and the development of the controlling software has been developed using the ACUTEST test language as a seperate project. This thesis project involved the design and implementation of a suitable functional tester at an affordable price. The following tester capabilities were desired:

1) 100KHz - 10MHz TEST FREQUENCY
2) 64 BIT TEST VECTOR WIDTH
3) 2KBytes TEST VECTOR DEPTH
4) FOUR USER PROGRAMMABLE CLOCKS
5). TEST VECTOR INPUT USING HIGH-LEVEL LANGUAGE
6) TEST (RESULTANT) DATA MANIPULATION USING HIGH-LEVEL LANGUAGE

To achieve these goals, a functional tester was designed which receives test vector data from a Hewlett Packard HP9920A computer and loads this data into buffer memory. The test vector data file is created using a program written in PASCAL on the HP9920A computer, which provides "user-friendly" input of the test vectors by the user. For example, data is entered in binary, hexadecimal, or decimal format. Several inputs of the same number, or group of numbers are entered using a multiplier. Ten instances of the pattern A5A5H followed by 5A5AH are entered by: (10(A5A5H,5A5AH)). Sequential inputs are entered by giving the start and end values.

To enter an address count from 0 to 2047 is done by entering: [00H..7FFH]. The data is transferred over a general purpose input/output (GPIO) bus with a direct memory access controller. To achieve the high transfer rates desired and to be compatible with the GPIO bus, the Functional Tester-GPIO interface uses a 16-bit microprocessor, the Intel 8086. The functional tester provides 2Kx64bits of RAM to buffer the test vectors, along with 2Kx64bits of RAM to store the test results. Another 2kx8bits of RAM is provided to buffer the direction bits which allow directional control of the 64 data lines in groups of eight during each cycle of the test. Registers are provided to hold the information for the programmable clocks and control signals. After all the data is down-loaded, which includes operational information such as test clock frequency, programmable clock pattern information, data direction, etc., the tester initiates the test using "random-logic" control circuitry to achieve the desired high speeds. The random-logic control circuitry uses a base oscillator at 20MHz to derive a master test clock programmable from 100KHz to 10MHz. The random-logic circuitry includes the write signal used by the resultant data RAM, along with the output data strobe pulse which strobes data from the test vector RAM into the output pipeline registers. Six clocks were also generated which can be programmed with low-to-high and high-to-low transitions occuring anywhere within the test cycle desired in 50 nanosecond increments. Two of the programmable clocks are used by the functional tester, with one clock used as the test vector data

control signals to/from the 8086. The address decoding necessary for the input and output ports is also included. This backplane interface circuit is shown in Appendix B.

Memory Boards

The purpose of the two memory boards is to buffer data prior to transfer to/from the HP9920A computer. The HM-65162S-9 2048 x 8bits, or 2KBytes, CMOS random-access memory (RAM) elements are used with read and write cycle times of approximately 55-nanoseconds. One memory board, "MEMORY BOARD 1", is used to hold the test vectors transferred from the HP9920A, and the other memory board, "MEMORY BOARD 2", is used to hold the resultant test data. MEMORY BOARD 1 also contains 2KBytes of RAM for the direction bits used during the test. During the transfer-phase between the functional tester and the HP9920A computer, the memory appears to the 8086 to be 8Kx16bits on MEMORY BOARD 2 located at addresses E0000H - E1FFFH, and 10Kx16bits on MEMORY BOARD 1 located at addresses F0000H - F1FFFH, with the extra 2K being the direction bits. During the test-phase, the data buffer memory on the two memory boards is configured as 2Kx64bits on each memory board, with an extra 2Kx8bits on MEMORY BOARD 1 for direction control.

The circuits for the two memory boards are presented in Appendix B. The block diagrams of the two memory boards are presented on the

interface to the Hewlett Packard GPIO bus, and to interface to the functional tester backplane. The interface to the GPIO bus was required to allow 16-bit transfers over two seperate 16-bit buses using two Intel 8255 input/output interfaces. This interface routes the GPIO timing interface signals PCTL and PFLAG to the correct control inputs of the interface 8255's, depending on whether an input or an output operation is in progress. Both Port A's of the two 8255's are used for inputting data from the Hewlett Packard computer to the functional tester, and both Port B's are used for outputing data from the functional tester to the Hewlett Packard computer. Port C of both 8255's is used as a control port for data transfers. Port A and Port B of 8255-P1 are the low-order bytes. The GPIO signal "I/O'" which is high for data transfers to the Hewlett Packard computer and low for transfers in the opposite direction, is used to route the necesary control signals. This interface circuit is shown in Appendix B. As shown, pull-up resistors are required on all the GPIO lines. This circuit does not present all of the Intel 8086 Single-Board Computer circuit since this is presented in the Intel MCS-86 User's Manual.

The backplane interface board is required to route the 8086 control, data, and address signals to the backplane of the functional tester module, and provide the necessary buffering. The board also includes address decoding for the two memory boards (due to a lack of space on the memory boards) and input and output ports for

some cases impossible to obtain in CMOS, and were prohibitively expensive. The design is included in Appendix B, since future availability of support circuitry for the 80C86 would make the design feasible. The other reason for not implementing the design involved time. As already indicated, parts to support the design using an 80C86 microprocessor were impossible to obtain, and obtaining a non-CMOS 8086 microprocessor, along with its support circuitry, was too expensive and required too much lead time. The alternative to this problem involved the use of an Intel SDK-86 System Design Kit which was available, and included monitor software to aid in development.

As previously discussed, the Intel SDK-86 was chosen due to time considerations, and ease in software development. Although no 8086 assembler was available, the Intel SDK-86 did allow examining memory locations, and performing input/output operations from the keypad. The discussion of the 8086 software written, and subsequently burned into EPROM, will be presented in Chapter III. The hardware and software documentation for the Intel SDK-86 is contained in the following manuals, which will be referenced in the discussion which follows.

1) SDK-86 MCS-86 System Design Kit Assembly Manual

2) MCS-86 User's Manual

3) 8086 Assembly Language Programming Manual

The Intel 8086 Single-Board Computer required interface circuits to

microprocessor design involved address space. The microprocessor used in the design (the Motorola 6800) allowed a total of only 65536bytes - 64KBytes - of address space. The original design criteria was for 4096 x 64bits, or 32KBytes, of output data memory, and 32KBytes of input data memory, for a total of 64KBytes of data memory, which resulted in no space for the program, or for registers used to hold test control information. For these reasons, a 16-bit microprocessor was deemed necessary for the interface and control board.

Several circuits were considered for the interface and control scheme, based on the Intel 8086 and the Motorola 68000 microprocessors. The Intel 8086 microprocessor was chosen for several reasons. One of the primary reasons for choosing the 8086 was personal experience in interfacing and programming the 8086 versus the 68000. Another important reason was availability of the 8086 over the 68000. A CMOS version of the 8086 - the 80C86 - was available from Harris Corporation as a donation, which made the 8086 highly desirable from a monetary standpoint. An interface circuit was designed using the 80C86, but was not implemented due to a problem obtaining supportive hardware for the 80C86, and time considerations. Harris Corporation donated the 80C86 microprocessor and scme support circuitry, but was not able ,due to manufacturing reasons, to furnish all of the parts required to support the 80C86 in the application circuit. The additional necessary parts were in

Computer Interface

The GPIO interface controls data transfers from the Hewlett-Packard HP9920A computer to the local memory of the functional tester. The interface is required to provide memory address and control signals to the local memory, as well as interface control signals to the GPIO bus. The Hewlett Packard GPIO bus is controlled by a direct-memory access (DMA) board which can transfer data at rates up to 750KHertz, or 750,000 transfers per second. This transfer rate was desired for the interface control circuit.

Based primarily on price considerations, design and testing was performed using an 8-bit microprocessor and supportive DMA to transfer the data. A design was produced which appeared feasible, but was later discarded for several reasons. Even though the GPIO controller could be configured to transfer 8-bit data, transferring 16-bit data was required in order to achieve the transfer rates desired. The extra control logic necessary to transfer 16-bit data using an 8-bit microprocessor was prohibitive for implementation reasons, and caused inter-communication problems between the microprocesor and the Hewlett Packard computer. This communication was necessary to provide varying modes of operation of the functional tester without excessive control requirements from the functional tester operator. Another problem with the 8-bit

to synchronize the programmable clocks with the master clock. The "STBOUT" signal strobes data from MEMORY BOARD 1 into the pipeline registers on the DUT INTERFACE board. This sigal determines the time data is available to the device-under-test, and is generated as soon as data is available from the RAM on MEMORY BOARD 1. The time delay from the master clock low-to-high transition to data valid at the device-under-test includes the propagation delay of the SN74LS163 address counter required to generate a valid address to the RAM; the propagation delay of the RAM from address valid to valid output; and the set-up and propagation delay of the SN74HC373 output register. From the device specifications, this time is approxiately 80-nanoseconds. As can be seen from the timing diagram, the functional tester can still be operated at speeds up to 10MHz since the output data is valid at the device-under-test from the high-to-low transition of the STBOUT signal until the next low-to-high transition of STBOUT. The time during which output data is actually enabled from the output latches to the device-under-test is a programmable signal - "PDOUTEN" - which is programmable in 50-nanosecond increments. The time during which input data from the device-under-test is written into the input latches is a prgrammable signal - STBIN- which is programmable in 50-nanosecond increments. These control signals, along with the 64-bits of output test data, enable a user to fully test the fuctionality and timing characteristics of a device, such as set-up time, propagation delay, etc..

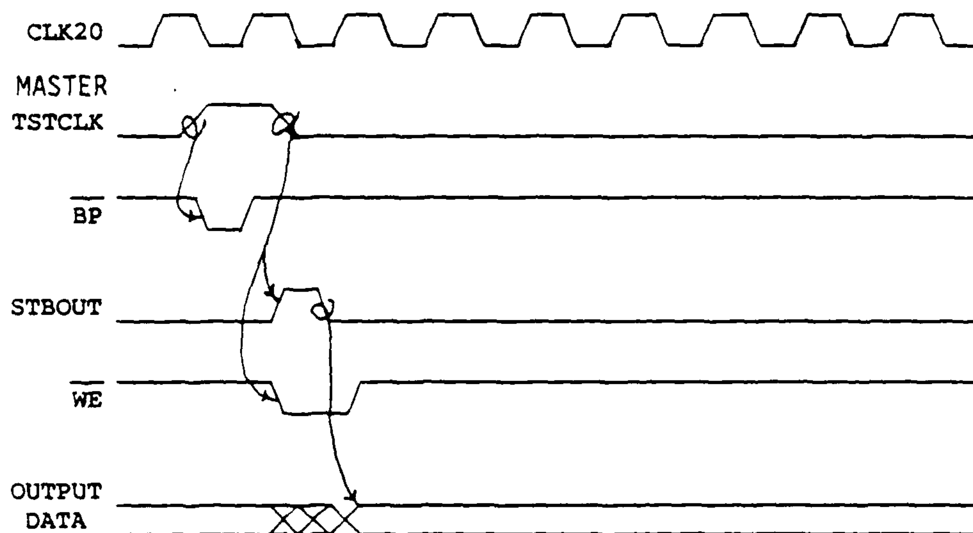



Figure 2: Timing Diagram

ability to test the memory. When the functional tester is in the test mode, the 8086 signals are removed from the tester module bus so that random-logic circuitry can generate the address and control signals at a higher rate of speed than possible with the 8086. During test mode, the address and control signals are provided by the TEST-PHASE CONTROL board. A programmable-frequency clock on the TEST-PHASE CONTROL board is used as the master clock, and generates the write signal to the RAM on MEMORY BOARD 2, the STBOUT signal to the output pipeline regsters on the DUT INTERFACE board, as well as driving a 12-bit counter for the address to all the RAM on both memory boards. During each cycle of this master clock, test vector data and directional data are read from the RAM on MEMORY BOARD 1 into output pipeline regsters on the DUT INTERFACE board, and data contained in input pipeline regsters on the DUT INTERFACE board is written to the RAM on MEMORY BOARD 2. The timing diagram shown in Figure 2, Timing Diagram, on the followig page, illustrates the timing relationship between the various control signals for a test frequency of 1MHz. The master clock, programmable clocks, and all control signals are derived from a 20MHz oscillator. The master clock is programmable from 100KHz to 10MHz, and is used to initiate the other signals. The high-to-low and low-to-high transitions of the programmable clocks can be programmed individualy in 50-nanosecond increments anywhere within one cycle of the master clock; or can be programmed to occur once every other cycle of the master clock. The "BP" signal generated by the master clock is used

| | COMPONENT SIDE | | | CIRCUIT SIDE | |
|---|---|---|---|---|---|
| | PIN | FUNCTION | | PIN | FUNCTION |
| LOGIC POWER BUS | 1 | P5V | | 2 | P5V |
| | 3 | GND | | 4 | GND |
| | 5 | N5V | | 6 | N5V |
| DATA BUS | 7 | D7 | | 8 | D15 |
| | 9 | D6 | | 10 | D14 |
| | 11 | D5 | | 12 | D13 |
| | 13 | D4 | | 14 | D12 |
| | 15 | D3 | | 16 | D11 |
| | 17 | D2 | | 18 | D10 |
| | 19 | D1 | | 20 | D9 |
| | 21 | D0 | | 22 | D8 |
| ADDRESS BUS | 23 | A8 | | 24 | M/IO' |
| | 25 | A7 | | 26 | A15 |
| | 27 | A6 | | 28 | A14 |
| | 29 | A5 | | 30 | A13 |
| | 31 | A4 | | 32 | A12 |
| | 33 | A3 | | 34 | A11 |
| | 35 | A2 | | 36 | A10 |
| | 37 | A1 | | 38 | A9 |
| CONTROL BUS | 39 | A17 | | 40 | MSEL1' |
| | 41 | A16 | | 42 | MSEL2' |
| | 43 | A0 | | 44 | TSTEND' |
| | 45 | WE' | | 46 | RD' |
| | 47 | TSTCLK | | 48 | TSTSTRT |
| | 49 | TSTMD' | | 50 | TSTMD |
| | 51 | * | | 52 | * |
| AUX POWER BUS | 53 | AUXGND | | 54 | AUXGND |
| | 55 | P12V | | 56 | N12V |

TABLE 1: BUS PIN ASSIGNMENT

entity from the other boards in the system. The two 64-pin ribbon cables labeled "JA" and "JB" connect the Intel single-board computer to the backplane of the module containing other Functional Tester boards through the "BACKPLANE INTERFACE" card. The module and wire-wrap plug boards used to fabricate the Functional Tester are "STANDARD BUS" size; however, the "STANDARD BUS" backplane pin asignment is not used. Table 1: Bus Pin Assignment, on the following page, shows the pin assignment used. The standard pin assignment is not used because a 16-bit data bus and a 20-bit address bus are required. The STANDARD BUS size module and plug boards are used because the typical module size for 16-bit data bus applications was considered too large to be placed in close proximity to the probe station, which may be required for the functional tester in order to test integrated circuits on silicon wafers. The BACKPLANE INTERFACE card also provides an input and an output port for control signals to/from the 8086 used to initiate, and determine the end of, the test. The address, data, and control signals from the 8086 are connected directly to the backplane of the tester module through buffers when the functional tester is in transfer mode. While in transfer mode, the 8086 transfers test vectors and directional control data to MEMORY BOARD 1; test frequency control data to the TEST-PHASE CONTROL board; and programmable clock control data to the PROGRAMMABLE CLOCKS board. In this mode, the 8086 can write or read any of these locations in addition to the memory locations on MEMORY BOARD 2, and may use this

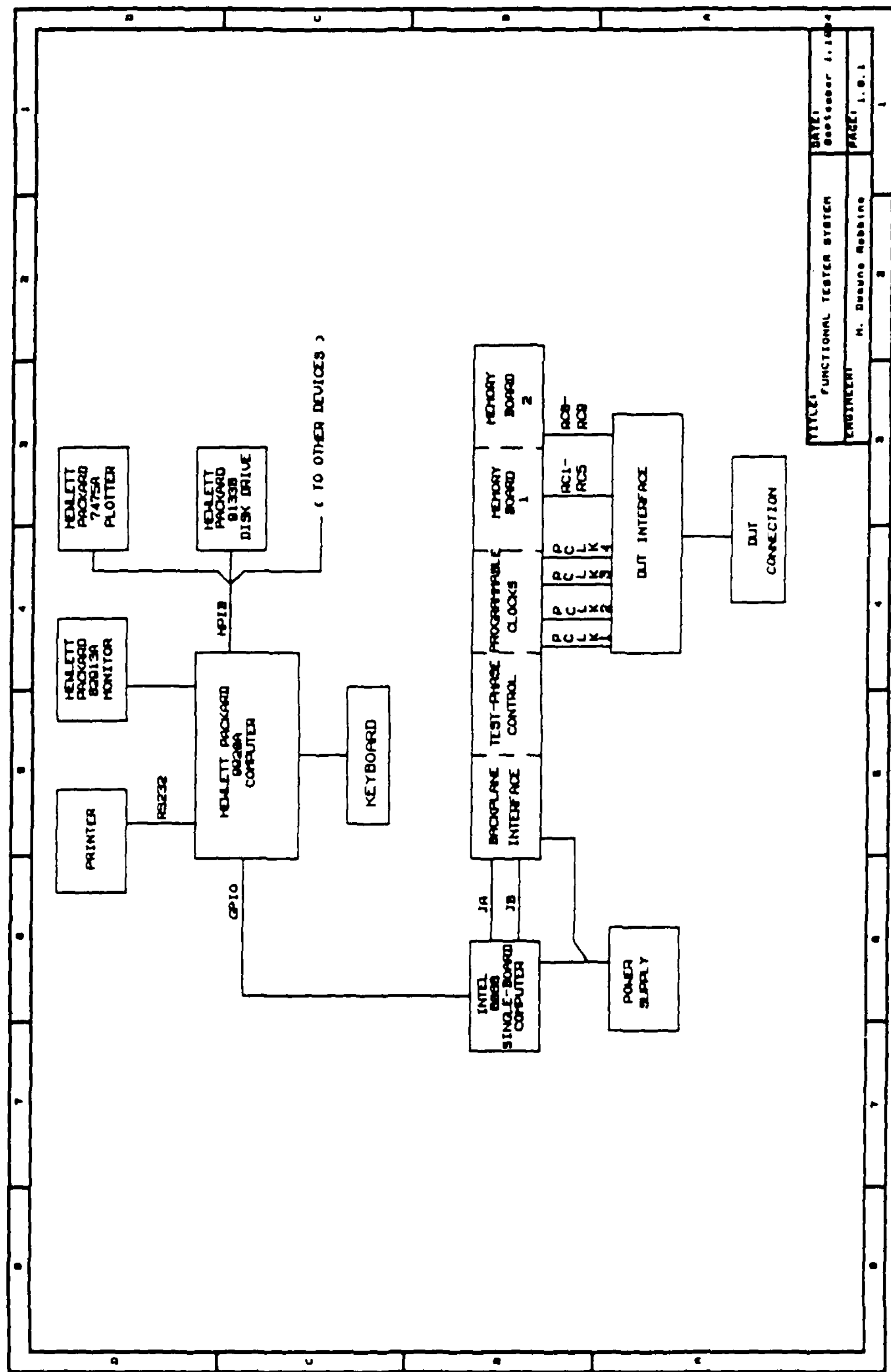



Figure 1: Functional Tester System

# CHAPTER II.

## Hardware Design

A system configuration diagram for the functional tester is presented in Appendix B, and also in Figure 1: Functional Tester System, on the following page. As can be seen from the block diagram, the system can be broken into sections pertaining to functionality. Each section will be discussed separatly, along with the interaction of the sections. The first section consists of the computer interface, interfacing the functional tester with the HP9920A GPIO Bus. This interface section also contains the control logic which initiates the test, and takes over control when the test is completed. The next section consists of the memory boards, with one memory board for the test vector data, and one memory board for the resultant data. The memory boards also contain the address-decoding. The next section discussed is the programmable clocks and test-phase control circuitry. The final hardware section includes the pipeline registers and device-under-test (DUT) interface.

The system configuration diagram shown in Figure 1 relates very closely to the physical, as well as the logical, configuration of the system. As shown, the GPIO interface cable connects directly to the Intel SDK-86 single-board computer, which is a separate physical

Also, a computer (the Hewlett Packard HP9920A) with a high-level language is used for entering the test vectors and processing the resultant data. Pipelining is used during the test phase so that the speed of the system is not completely dependent on the access time of the memory elements used for buffer memory. The maximum frequency of the functional tester is determined by the memory access time of the RAM plus the data setup and propagation delay of the output pipeline registers. High speed CMOS RAM is used with a memory access time of approximately 55-nanoseconds, and high speed CMOS pipeline registers are used with a combined data setup and propagation delay of approximately 15-nanoseconds, indicating the maximum possible operating speed for the functional tester to be approximately 14MHz.

output enable from the output pipeline registers, and the other clock used as the resultant data input strobe to the input pipeline registers. The other four programmable clocks are available to the user for functions executed during the test. The random-logic control circuity indicates the completion of the test, at which time the resultant data, which had been stored in buffer memory, is up-loaded to the HP9920A for processing. The PASCAL program on the HP9920A places this resultant data into a file, after sorting out all of the non-relevant data. A hardcopy may be obtained of the test vectors and resultant data on a printer.

This design approach to a functional tester for laboratory use differs from --and improves upon-- previous methods, in that random-logic *control circuitry* is used during the test phase to provide increased operating speeds over systems which use microprocessor-control for the complete test. Systems using a microprocessor to control the complete test can operate only as fast as the execution of a memory fetch and output write followed by an input read and memory write. A maximum test frequency of approximately 125KHz for an Intel 8086 operating at 5MHz is achievable, and this would allow only 16-bit test vectors. Test vectors of 64-bits would require four of the previous cycles for each test cycle, indicating a maximum test frequency of approximately 30KHz for the 8086 operating at 5MHz.

following two pages, in Figure 3: Memory Board 1 and Figure 4: Memory Board 2, for easier reference. On MEMORY BOARD 1, the write signal to the RAM is inhibited during test-phase by the control signal "TSTMD". On MEMORY BOARD 2, the read signal is inhibited during test-phase by TSTMD; therefore, during test-mode MEMORY BOARD 1 is read-only memory, and MEMORY BOARD 2 is write-only memory. The data from MEMORY BOARD 1 is valid as soon as the address is valid, plus the RAM latency time. New data is written into MEMORY BOARD 2 as soon as the new address is valid. The timing sequence of the read and write cycles for the two memory boards is included in the system timing analysis presented in the introduction of this chapter.

Since the 8086 address space is not greatly utilized, more memory boards could easily be added to increase the depth or the width of the test vectors. To increase the depth, memory boards should be added in pairs, such that going from 2Kx64bits to 4Kx64bits would include the output data as well as the resultant data. Increasing the width of the test vectors could be done for the output data only, input data only or both as required. In other words, the test vector memory buffer area could be increased from 2Kx64bits to 2Kx128bits by adding one memory board, with the resultant data memory buffer area remaining at 2Kx64bits. Another possible addition to the memory boards circuit would be a 2Kx64bit memory board to contain the direction bits, which would replace the

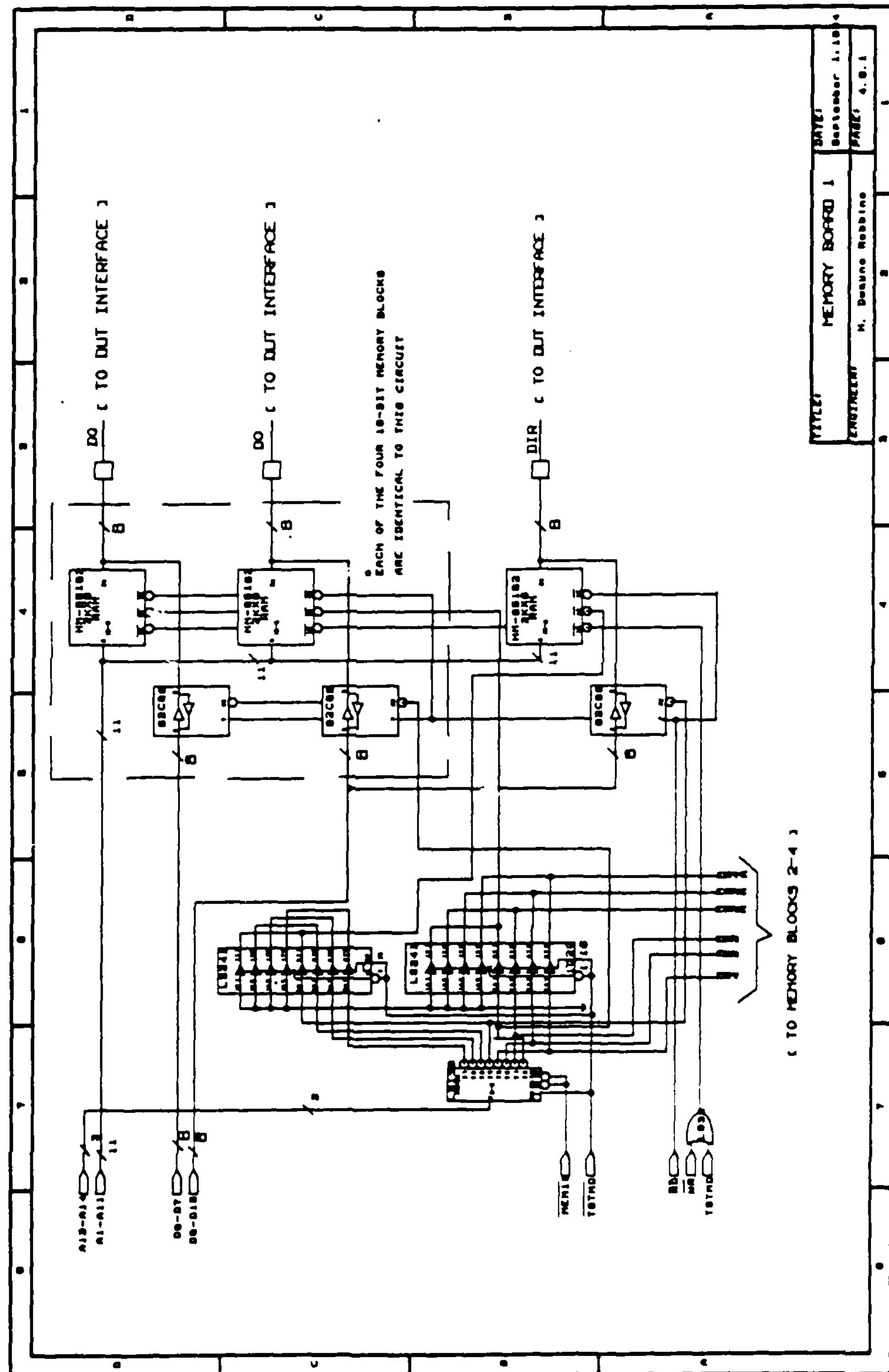



Figure 3: Memory Board 1

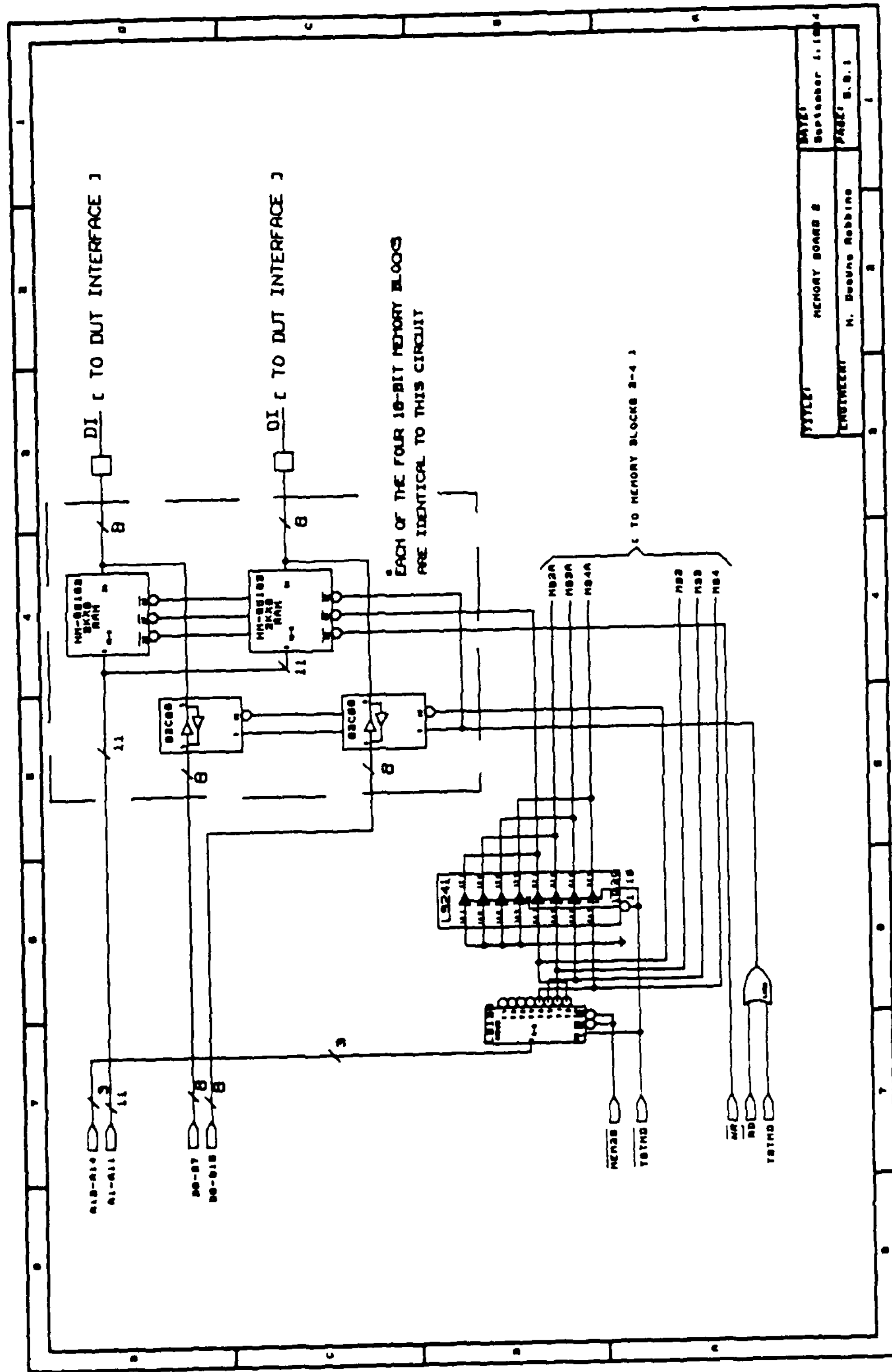



Figure 4: Memory Board 2

2Kx8bits RAM on MEMORY BOARD 1. This would allow each bit to be programmed as either an input or an output during each cycle of the test, rather than the direction of 8 bits being programmed together, as is presently done. This would necessitate changing the DUT INTERFACE board to use different elements than the 8-bit buffers presently used. A device such as an AND gate with tri-state outputs could be used to control each of the 64 output bits. This would require 16 of the AND gate chips, assuming four 2-input gates per chip; however, sixteen of these 14-pin chips would not require much more room than eight of the 20-pin chips presently used.

## Programmable Clocks

The PROGRAMMABLE CLOCKS board provides six programmable clocks, with four of the programmable clocks (PCLK1 - PCLK4) for use by the user as desired, and the other two programmable clocks used for control signals during test phase by the DUT INTERFACE board. All six of the programmable clocks are connected via shielded coaxial cable to the DUT INTERFACE board. The purpose of the two programmable control signals (PDOUTEN and STBIN) is described in the discussion of the DUT INTERFACE board operation. The PROGRAMMABLE CLOCKS board contains a 20MHz oscillator which is used to produce the six programmable clocks, and is also connected via shielded coaxial cable to the TEST-PHASE CONTROL board for use in producing the programmable frequency master test clock.

Although the six programmable clocks have different purposes, they are all generated in the same manner; therefore, this discussion will describe only one of the six. The schematics for the programmable clocks is presented in Appendix B. The block diagram is shown in Figure 5: Programmable Clocks, on the following page to aid in this discussion. As indicated by the block diagram, the 20MHz oscillator is used to drive two 74LS161 4-bit binary counters. Only seven of the eight bits of the counters are used, which allows a count range of 0-127 in 50 nanosecond steps. The two counters are reset by a control signal, BP, which is generated on the TEST-PHASE CONTROL board at the beginning of each cycle of the master test clock; therefore, the two counters always indicate the count in 50 nanosecond increments after the low-to-high transition of the master test clock. This 7-bit count is connected to inputs B0-B6 of an AM25LS2521 16-bit comparator. Although not shown on the block diagram, the high-order bit of the "A" and "B" inputs (A7 and B7) are connected together since they are not used. The 7-bit count is compared by the comparator to the 7-bit output of one of two latches. Seven of the output bits (Q0-Q6) of both latches are connected to the inputs A0-A6 of the comparator, with only one latch enabled at a time. One latch contains the count for the low-to-high transition of the programmable clock, and the other latch contains the count for the high-to-low transition of the programmable clock. These two latches are loaded during transfer phase from the Hewlett

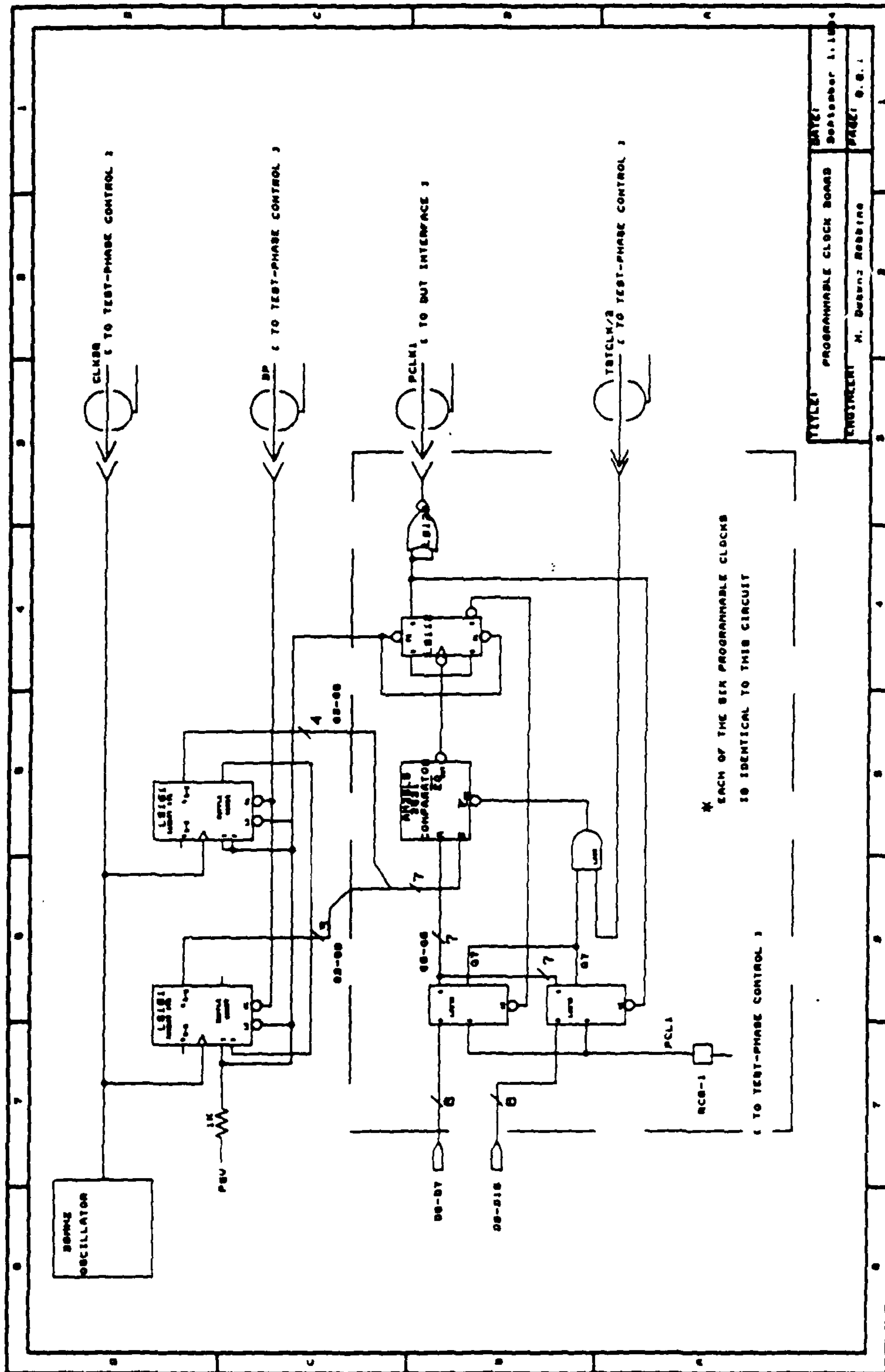



Figure 5: Programmable Clocks

Packard 9920A computer. The address decoding for the latches is done on the TEST-PHASE CONTROL board, and the strobe signals are passed over via ribbon cable. The high-order bit of the two latches is used to determine whether the programmable clock will occur each cycle, or every other cycle, of the master test clock. This is accomplished with an AND gate which has the high-order bit of the two latches as one input, and the master test clock divided by two as the other input. The output of the AND gate enables the comparator. Whether the programmable clock low-to-high or high-to-low transition occurs first is user programmable according to which latch contains the lower count. When the comparator detects a match between the 7-bit count and the output of the latch presently enabled, a pulse is produced at the output of the comparator which clocks a 74LS112 JK flip-flop. The 74LS112 is connected in toggle mode so that a clock input causes the flip-flop to toggle. This produces a transition of the programmable clock, and enables the other latch, which contains the count for the opposite transition of the clock. The output of the 74LS112 is connected to a 74LS128 50-ohm coaxial cable driver which drives the coaxial cable connected to the DUT INTERFACE board. Once a new count is loaded into the latches during transfer phase, all of the programmable clocks are free running at the new set-up, and are valid when the test-phase begins.

The operation of the programmable clocks was satisfactory for a

clock programmable in 50nanosecond increments; however, a clock programmable in smaller increments, such as 10nanoseconds or less, was desirable for a functional tester that operated at speeds up to 10MHz. A very-large-scale integrated (VLSI) circuit was considered which would use an internal clock of 100MHz or greater to produce a clock programmable in 10nanosecond, or smaller, increments. In order to be programmable over the entire cycle at speeds as low as 100KHz, the 10nanosecond programmable clock would require a count range of 0-1000, or a 10-bit counter; however since typical uses of these programmable clocks would be in the higher frequency ranges, an eight-bit count, which would allow programming of 0-2560nanoseconds (full range at 400KHz), would be adequate. The VLSI circuit would require data and control lines to input the low-to-high and high-to-low counts, and a strobe pulse input indicating the beginning of each cycle. A circuit similar to the ones on the PROGRAMMABLE CLOCKS board could be used, or a separate loadable counter could be used for each transition of the programmable clock. This accuracy for the programmable clock is required in order to run the functional tester at its desired 10MHz test frequency, and should be the first major modification performed on the existing design.

## Test-Phase Control

The TEST-PHASE CONTROL board provides three functions, the first is

used during transfer-phase, and the other two are used during test-phase. The first function is the address decoding for the programmable frequency master test clock latch located on the TEST-PHASE CONTROL board, and the programmable clock latches located on the PROGRAMMABLE CLOCKS board. The second function is the generation of the programmable frequency master clock and the other control signals used during test-phase. The third function is the generation of the address for the RAM located on the two memory boards. The schematics for the TEST-PHASE CONTROL board are located in Appendix B.

The first page of the TEST-PHASE CONTROL schematics located in Appendix B shows the address decoding which is used during transfer-phase to load the latches used for the programmable frequency master clock and the six programmable clocks. An SN74LS138 3-to-8-line decoder is used to generate the device select pulses. The outputs of the SN74LS138 are connected to SN74LS04 inverters since the strobe inputs of the SN74LS373 latches require high-true signals. The SN74LS138 produces device select pulses for I/O addresses A000H-A00EH, with each device select pulse representing 16-bit devices. For example, the master test clock latch address is A000H/A001H; the latches for the first programmable clock (PCLK1) are located at addresses A002H/A003H; and so forth for the other programmable clocks.

The second page of the TEST-PHASE CONTROL board schematics located in Appendix B shows the address generation during test-phase for the RAM located on the two memory boards. Three SN74LS163 4-bit binary counters driven by the programmable frequency master-test-clock are used to generate the address lines. The address counters are enabled by the "TSTSTRT" control signal generated at the 8086 output port located on the BACKPLANE INTERFACE board. Two AM25LS2521 comparators are used to determine when the first eleven of the counter's twelve outputs are high, indicating that all 2KBytes of the RAM have been accessed, and stop the counters. The signal used to stop the counters is also used as the end-of-test indicator (TSTEND) to the 8086. The outputs of the address counters are connected to the address bus through two SN74LS241 octal three-state buffers, so that the address counters are only connected to the address bus when the functional tester is in test mode.

The third page of the TEST-PHASE CONTROL board schematics located in Appendix B shows the circuitry used to produce the control signals used during test-phase. This includes the programmable-frequency master-test-clock (TSTCLK), the begin-phase pulse (BP), the master-test-clock divided by two (TSTCLK/2), the output data latch strobe (STBOUT), the RAM write signal (WR), and the RAM read signal (RD). The purpose of the begin-phase (BP) and test clock divided by two (TSTCLK/2) signals are included in the discussion of the PROGRAMMABLE CLOCKS board. The purpose of the output latch strobe

signal is included in the discussion of the DUT INTERFACE board. The purpose of the RAM read (RD) and write (WR) signals are included in the discussion of MEMORY BOARD 1 and MEMORY BOARD 2. This discussion will only present how these signals are generated.

Two SN74LS163 4-bit binary counters are used to generate the programmable frequency master test clock. The 20MHz clock generated on the PROGRAMMABLE CLOCKS board is used to clock the two counters. The two counters are loaded with the data contained in the test clock SN74LS373 latch at the beginning of each cycle. The counters count up from that value until they reach a count of 255, or all ones on their outputs. This causes the ripple carry output of the high-order counter to make a low-to-high transition, and remain high for one clock period (50nanoseconds). This ripple carry output is used for the master test clock, and re-loads the count data into the two counters. The formula for the test clock frequency based on the count value loaded into the latch is given by: F = 20MHz/(255-COUNT+1). This formula is transparent to the user; however, since the high-level language on the Hewlett Packard 9920A computer handles calculating this value, and ensures that a test frequency is not picked which cannot be generated.

The master test clock is used to drive the test-phase address counters, and to generate the remaining test-phase control signals. The test clock divided by two (TSTCLK/2) signal was generated by

connecting the master test clock to the clock input of an SN74LS74 which is configured to toggle as shown on the schematics. The begin-phase pulse (BP) was generated by connecting the master test clock to the clock input of an SN74LS74 with its data input ("D") connected high. Each time the flip-flop received the clock its "Q" output would go high and its "Q'" output would go low. The output "Q'" was connected to the clear input "CL" such that as soon as "Q'" went low, the flip-flop was cleared, causing "Q" to go back low, and "Q'" to go back high. The begin-phase signal was taken from the "Q'" output, making it a low-true pulse, which occured at the start of each cycle of the master test clock, and lasted for the duration of the SN74LS74 high-to-low propagation delay. The SN74LS74 high-to-low propagation delay was specified as typically 13nanoseconds, with a maximum of 25nanoseconds, but was measured as 20nanoseconds. This time was satisfactory since the begin-phase signal was connected to the asynchronous clear of the SN74LS161 counters on the PROGRAMMABLE CLOCKS board. The output data strobe signal (STBOUT) was generated in the same fashion as the begin-phase signal, except the "Q" output of the SN74LS74 was used to allow for a high-true signal, and the clock source was different. The low-to-high transition of the inverse of the master test clock, which occurs 50nanoseconds after the low-to-high transition of the uninverted master test clock, was used to clock the strobe out flip-flop to cause the signal to appear later in the cycle, which allowed the test-phase address counters to stabilize, and the data

out of the RAM on MEMORY BOARD 1 to become valid. The duration of the output data strobe signal was the same as for the begin-phase signal, which was the propagation delay of the SN74LS74, and was adequate for the strobe to the high-speed CMOS latches used on the DUT INTERFACE board. The write signal to the RAM on the memory boards was generated using the same clock input as the output data strobe signal, which was the low-to-high transition of the inverse of the master test clock. The duration of the write signal was generated by the gates inserted between the "Q'" output and the clear input "CL" of the SN74LS74. The required minimum duration of the write pulse to the HM-65162 RAM according to the specifications was 35nanoseconds. The propagation delay of the SN74LS74 of approximately 20nanoseconds, plus the propagation delay of the SN74LS32 of appoximately 15nanoseconds, plus the propagation delay of the SN74LS04 of approximately 10nanoseconds produced a write pulse with a duration of 45nanoseconds. The "Q'" output was used for the write signal to produce a low-true signal. The read signal was tied to ground during test-mode, which enabled the output of the RAM on MEMORY BOARD 1 at all times during the test.

The six programmable clocks are synchronized to the master-test-clock by use of the begin-phase (BP) control signal as indicated above. After this circuit was already implemented, the begin-phase control signal position was discovered to be in error. Since the functional tester output data is available to the

device-under-test as a function of the output data strobe (*STBOUT*) signal, all of the programmable clocks should be synchronized to the STBOUT signal. This means that future modifications to the system should replace the BP signal with the inverse of the STBOUT signal. Also, the input to the flip-flop which generates TSTCLK/2 should be the STBOUT signal instead of the master test clock. This means that from a user's point of view, the STBOUT signal is the master clock, even though a different frequency-programmable master test clock is really being used to synchronize the entire system.

As indicated, the control signals produced on this board were adequate for their purposes; however, they were impossible to adjust by very small amounts (i.e. 5-10nanoseconds) to produce a more accurate system. The VLSI circuit which would generate a programmable pulse based on an input strobe signal, as mentioned in the discussion of the PROGRAMMABLE CLOCKS board, would be very well suited to more accurately generate these control signals, and should be used when the programmable clocks are modified.

## Device-Under-Test Interface

The DEVICE-UNDER-TEST INTERFACE (DUT) board is located physically seperate from the module containing the majority of the boards in the system so that it can be placed near the probe card when integrated circuits on silicon wafers are tested. The interface

indicated by the multiplier into the data file. The UNASSEMBLE function may iterate if multipliers are located within multiplier strings. The CREATE module keeps track of the number of values input for each data group, and issues a warning if too many values are nput for any one data group, and re-enters the values for that group.

The purpose of the CLOCKS module is to enter the test frequency and positions of the programmable clocks using graphics on the HP9920A monitor to allow the user a better understanding of how to program the clocks. The CLOCKS module first inputs the test frequency. The CLOCKS module then calls a function called PCLOCKS six times, once for each of the six programmable clocks. The PCLOCK module draws a 20MHz base-frequency clock; the tester-master-clock (at the frequency previously input); and a programmable clock with default low-to-high and high-to-low transitions. The user is allowed to independently move the low-to-high and high-to-low transition edges in 50nanosecond increments until the desired programmable clock waveform is achieved. The user then indicates acceptance of the clock, and the next programmable clock is drawn. After all six of the programmable clocks have been input, the module CLOCKS calls a function named DRAW_ALL which draws the 20MHz base oscillator, the master-test-clock (at its programmed frequency), and all six programmable clocks (at their programmed values). The user is then given the option to accept the waveforms or reprogram all of the

may have an appended "D", or no appended letter (ex: 9086D or 9086). The module allows sequential data to be entered by giving only the first and last values. For example, if a test requires an address output which varies from 0 to 2047, the data could be entered as "[00H..7FFH]" which would enter 2,048 values. The module allows repeated data to be entered with a multiplier. For example, to enter a count from 0 to 15 and then from 40 to 49, fifty times, the following could be entered: "(50([00H..0FH],[40..49])" which would enter 1,300 values. Repetitions within repetitions are also allowed, such that entering fifty vectors which includes ten 0FH values followed by a count from 0 to 9 could be entered: "(50(10(0FH),[0..9])". The CREATE module inputs a value and checks the leftmost character of the value for a "(". If the left character is not a "(", the module checks for a leftmost character of "[". If the leftmost character is not either of these, the CREATE module calls a function called INTGD which accepts a binary, hexadecimal, or decimal string and returns an integer. This integer is written into the data file, and the program prompts for a new value to be input. If the leftmost character is a "[", the CREATE determines the start and end values, and calls INTGD to transform these values into integers. The values from the start integer to the end integer are then writen into the data file. If the leftmost character is a "(", the CREATE module calls a function called UNASSEMBLE. The function UNASSEMBLE determines the multiplier for the string and writes the integer values for the number of times

test repeatedly while varying the test frequency and programmable clocks. If the same programmable clocks were not to be used again, the program calls the CLOCKS module to input the new clock values. After the test vector data file is established, either by creating a new file or determining which old file to use, the MAIN program calls the TRANSFER module which transfers the test vectors and control data to the functional tester, and transfers the resultant data from the functional tester to the HP9920A computer. After the resultant data had been loaded, the MAIN program determines if a listing of the results is desired, and if so, whether the listing should go to the monitor or to the printer. If a listing is desired, the MAIN program transfers a listing to the desired device.

The purpose of the CREATE module is to allow "user-friendly" input of the test vectors. Since the type devices tested by the functional tester are expected to vary a great deal, no attempt was made to create a module which would compile a higher level language to create the test vectors. This means that the test vector data must be hand-assembled --or "microcoded"-- by the user. The CREATE module does ease the efforts required to input the test vectors. To describe the operation of the CREATE module, a description of the abilities of the module will first be presented. The CREATE module allows data to be input in binary, hexadecimal, or decimal format. Binary data must have an appended "B" (ex: 01011011B); hexadecimal decimal data must have an appended "H" (ex:0F0AEH); and decimal data

functional module without affecting the other modules. The first function module is the CREATE function which allows input of the test vectors to be used in the test. The second function module is the CLOCKS function which allows graphical input of the positions of the programmable clocks to be used in the test. The third functional module is the TRANSFER function which transfers the test data to the functional tester, and transfers the resultant data from the functional tester to the HP9920A. The PASCAL program and accompanying flowcharts are presented in Appendix D.

The MAIN module is the driver for the other functional modules. The program first determines whether the test vector data file already exists. If the test vector data file did not exist, the program inputs information to determine how many test vector words will be input, and then calls the CREATE module. After the test vectors are entered, the program again calls the CREATE module to input the direction words. The MAIN program sets a boolean variable "DIR_WRD_IN" true or false before calling CREATE to differentiate between test vector inputs and direction word inputs. After inputting the test vectors and direction words, the MAIN program calls the CLOCKS module to input the values for initializing the functional tester programmable clocks. All of these values are placed in one test data file. If the test vector data file already existed, the program determines whether the same programmable clocks are to be used again. This is included to allow running the same

have to allow for memory partitioning on the memory boards if all the memory was filled every time. After transfering the test vectors and direction control words, the control program inputs seven 16-bit words used for the functional tester programmable clocks. Once all the values are input, the control program changes the functional tester to test mode and initiates the test by outputting the correct bits to Output Port 0. After initiating the test, the control program enters a loop reading Input Port 0 and waiting for an indication that the test has ended. At the end of the test, the control program sets the functional tester mode to transfer, and initializes the data segment and index registers to point to the start of the RAM on MEMORY BOARD 2, which begins at address 0E000H. The control program then outputs 8,192 (8K) 16-bit resultant data words to the HP9920A computer. The high-level program on the HP9920A takes care of selecting the true resultant data from the data words which are transferred. After transferring all of the resultant data words, the assembly language control program jumps to the start of the program beginning (physical address 0100H) to start the process over again for the next test.

## PASCAL Program

The PASCAL program on the Hewlett Packard HP9920A computer consists of three major functions which are placed in three seperate modules, along with a main driver module, which allows changes to any one

automatically jump to the utility program upon power-up. The SDK-86 Monitor program performs several initialization routines upon power-up, and then enters a loop waiting for keypad input. Changing the Monitor program to jump to the utility program rather than waiting for keypad input would have required several patches which would be very difficult without an 8086 Assembler. Replacing the SDK-86 Monitor PROM's with EPROM's containing the utility program would have been simple, but would not have allowed use of the SDK-86 Monitor program, which was very useful in trouble-shooting. Based on these considerations, the alternative, which involved manually starting the execution of the utility program at address 0FE000H was reasonable.

The second assembly language program, and accompanying flowchart, is presented in Appendix C: Functional Tester Control Program. The program begins by initializing the PIA's, and setting the functional tester to transfer mode. Transfer mode is set by outputting the correct bits to Output Port 0. The control program loads the data segment and index registers to point to the start of the RAM on MEMORY BOARD 1, starting at address 0F000H. The program then inputs 8,192 (8K) 16-bit test vector data words, and 2,048 (2K) 8-bit direction data words from the HP9920A computer. The high-level program on the HP9920A computer inserts "dummy" data words if less than 2Kx64bits of test vector data is used. This was done to simplify the assembly language program, since the program would not

program much simpler to implement. The second assembly language program written is the program which controlls the functional tester.

The downloader utility program and flowchart is presented in Appendix C: Utility Program. The program first loads the data segment and index registers to point to the destination in memory for the program to be downloaded, which is physical address 0100H. Address 0100H is the first available space in the SDK-86 on-board RAM. The program then initializes the peripheral interface adaptors (PIA's) and checks for a word written in. The first word read in from the HP9920A computer is the transfer word count used to determine how many words will be transferred. The program then continues in a loop: checking the status for a new word written in, inputting and storing the word, and decrementing the count, until all the words have been read in and the count equals zero. The program then jumps to physical address 0100H and starts executing the new program. This mode of operation is desirable since it allows easy modifications to the functional tester control program, as previously indicated, without the necessity of burning-in new EPROM's. Other programs, such as memory test programs and other maintenance programs, may just as easily be downloaded from the HP9920A computer and used in trouble-shooting problems with the functional tester. The only inconvenience with the method which this utility program has implemented is that the SDK-86 does not

functional tester could include compilers for specific circits which are tested often, such that the test vectors could be generated at the assembly language level rather than microcode. Another desirable modification would be the ability to automatically analyze the results produced by the functional tester, instead of analyzing by hand. This could be done for a specific element which is tested often by using the "known good" method in which the test is run on a known good element, and the results of this test are compared to the tests of all the other elements of the same type to locate discrepancies. Another method for automatically analyzing the results, which could also be used to produce the test vectors, would be an emulator for a specific circuit which would be run on a mainframe communicating with the HP9920A.

## Assembly Language Programs

Two 8086 assembly language programs were written to operate the Intel SDK-86 used as the microprocessor control of the functional tester. The first program is a utility program to download, and start executing, another program from the Hewlett Packard HP9920A computer. This program is burned into EPROM and installed in the SDK-86. The utility program is executed when the SDK-86 is powered-up, and downloads the program which actually controlls the functional tester from the HP9920A computer. This method is employed to make modifications to the functional tester control

# CHAPTER III

## SOFTWARE DESIGN

Two areas of software design were required to produce an operable functional tester. The first was the 8086 assembly language programs required to operate the Intel SDK-86 which transferred data to/from the Hewlett Packard HP9920A computer and controlled the functional tester. Although written in 8086 assembly language, no 8086 Assembler was available, and the programs were hand-assembled. This accounts for the lack of Assembler statements in the assembly language programs. Although not specifically referenced in the following discussion of the assembly language programs, all of the references listed under the REFERENCES section which deal with 8086 programming and the Intel SDK-86 manuals were used in writing the 8086 assembly language programs. The second area of software design was the high-level language programs written in PASCAL on the HP9920A computer. The purpose of the PASCAL programs was to create the test vector data file, transfer the test vectors to the functional tester, input the resultant data from the functional tester, and produce hardcopies of the results. Since the circuits to be tested on the functional tester were expected to vary a great deal, no attempt was made to create a compiler to create the test vectors, and the PASCAL program required the user to hand-compile, or "microcode", the test data. Future modifications to the

but a 64-pin zero-insertion socket could not be found. These two sockets allow connection of almost any standard dual-in-line packaged integrated circuit. A ribbon cable with a header to fit the zero-insertion socket can be used to connect to a probe card for use in testing integrated circuits on silicon wafers.

would allow for the sixty-four data lines and eight direction lines from MEMORY BOARD 1 in one ribbon cable. Another ribbon cable would carry the sixty-four data lines from MEMORY BOARD 2, and could include the power signals and the TSTMD signal. The interface from the DUT INTERFACE board to the DUT board was a card-edge connector; however, due to problems with this connection, the interface was made permanent. This is not satisfactory, since the DUT card must be changed to go from testing packaged integrated circuits to testing integrated circuits on a silicon wafer, and the DUT board may be changed when changing the device being tested, since some device dependent circuitry may be included on the DUT board. A different connection is required for this interface, which allows flexibility, but also provides acceptable connection during high speed tests. A daughter-board arrangement would appear to be the optimal connection.

## DEVICE-UNDER-TEST BOARD

The DEVICE-UNDER-TEST (DUT) board presently used with the functional tester is shown in Appendix B. As can be seen, this board is simply a patch-board to allow routing any of the sixty-four data lines, four programmable clocks, P5V, or GND to any pin of the device-under-test. The device-under-test connection includes a 20-pin zero-insertion socket and a 40-pin zero insertion socket. The board was designed for 20-pin and 64-pin zero-insertion sockets,

high-level language program on the Hewlett-Packard 9920A computer, leaving only the resultant data read in from the device-under-test. The time at which the data is written into the input latches from the device-under-test is determined by "STBIN", which is a user-programmable control signal generated on the PROGRAMMABLE CLOCKS board. This allows a user to determine the exact time that data is read in from the device-under-test in order to test such things as propagation delay or read access time of the device being tested.

Since the DUT INTERFACE board is not housed in the module, the power (P5V and GND) and the TSTMD control signal are input through a ribbon cable from MEMORY BOARD 1. Besides powering the DUT INTERFACE board, these power signals are sent to the DUT board. A needed extra for the functional tester is two programmable voltage sources to connect to the device-under-test, and possibly to the pipeline register latches. This would allow testing the A.C. characteristics of a device at different voltage levels, and would allow testing devices which require a different voltage level than P5V and GND.

Besides programmable voltage supplies, the physical configuration of this board needs to be modified. Due [illegible] availability, the ribbon cables used from the two memory boards were 14- and 16-pin ribbon cables. These should be changed to 72-pin ribbon cables. This

PREVIOUS PAGE
IS BLANK

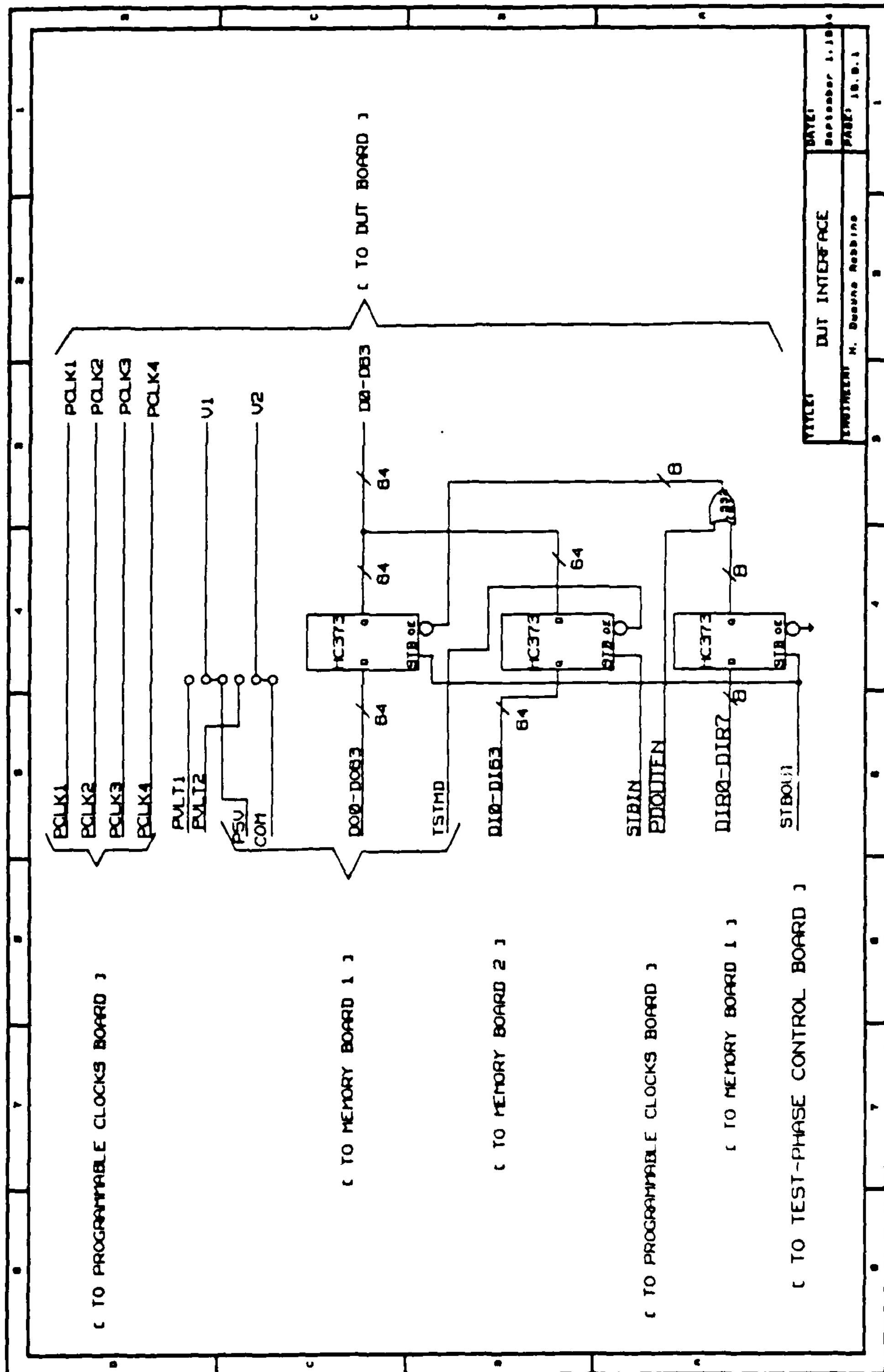



Figure 6: DUT Interface

between the DUT INTERFACE board and the rest of the system consists of ribbon cables to the two memory boards and shielded coax cables to the PROGRAMMABLE CLOCKS board. The interface to the DUT board is a card-edge connector.

The schematics for the DUT INTERFACE board are presented in Appendix B. The block diagram is also presented in Figure 6: DUT Interface, on the following page to aid in discussion. As indicated by the block diagram, the four user programmable clocks from the PROGRAMMABLE CLOCKS board are not used on this board, but are passed through to the DUT board. Sixty-four test data lines and eight direction lines are received from MEMORY BOARD 1 by high speed, CMOS latches. The data is clocked into these latches by the "STBOUT" signal generated on the TEST-PHASE CONTROL board. As indicated by the timing diagram previously discussed, this signal is generated as soon as the data from MEMORY BOARD 1 is valid each cycle. The output of the latch for the directional bits is always enabled. These directional bits are used in conjunction with the programmable-control signal "PDOUTEN" to enable the output of the data latches to the device-under-test. If the direction bit is high for a particular group of eight data lines, then those data lines are inputs from the device-under-test to the functional tester for that cycle. If the direction bit is low for a group of eight data lines, then those data lines are outputs from the functional tester to the device-under-test for that cycle; however, the data is not

clocks. After the user accepts the waveforms, the graphical data is used to calculate the values to send to the functional tester so that the desired waveforms are obtained, and these values are written to the data file. The CLOCKS module then determines if a plot of the clock waveforms is desired, and if so, calls the DRAW_ALL module after initializing the plotter.

The purpose of the TRANSFER module is to transfer the test vectors and control data from the HP9920A computer test data file to the functional tester, and to input the resultant data from the functional tester to the HP9920A computer resultant data file. The TRANSFER module first initializes the GPIO channel and opens the test vector data file. The TRANSFER module then inputs the number of data groups used and the number of data words per group from the data file. These two numbers are used to indicate when to input a test vector from the data file to transfer, and when to transfer a "dummy" value. The TRANSFER module always transfers four blocks of 2048 (2K) 16-bit test vector data words, and then 2048 (2K) 16-bit direction words. If the group count indicated that a test vector data group was being transferred that was not used, or if the word count indicated that all of the test vector data words for the particular group had been transferred, dummy values of OFFFFH are transferred. After transferring the 10K 16-bit words for test vectors and direction control, the TRANSFER module transfers the clock programming values. When transfers of all test data is

completed, the TRANSFER module creates a new file for the resultant data and inputs 2K 16-bit resultant data words. The resultant data mask, which has been input earlier in the program, is obtained from the test data file, and used to filter out unused groups. The words per group value, which has been previously input, is used to filter out the unused words at the end of each group. The direction words are used from the test data file to indicate if a value is non-valid due to being in output mode for that cycle. If that is the case, an "XX" is written in the resultant file for that value. After inputting all of the resultant data words, the TRANSFER module closes the resultant file.

# CHAPTER IV

## Operation

The operational concepts of the functional tester, along with the particulars of the hardware and software operation, have been discussed in the previous chapters. This chapter will reiterate the previous discussions concerning the operation of the functional tester in order to bring together the total requirements necessary to complete a test of a device using the functional tester.

The functional tester requires modifications which have been discussed in previous chapters and will be discussed in greater detail in the following chapters to reach the desired level of testing capabilities; however, as the functional tester now exists, the following capabilities are available:

- 100KHz-5MHz Test Frequency
- 64 Input/Output Data Lines
- Direction Control of Data Lines in Groups of Eight
- Four User-Programmable Clocks
- User-Programmable Data Output Enable and Data Input Strobe Pulses
- "User-Friendly" Input of Test Vectors

- Sorted Results in Hardcopy (Printer) or
  Softcopy (Floppy or Hard Disk)

The following steps are required to operate the functional tester:

1) Make Electrical Connections
2) Functional Tester Power-Up Initialization
3) Enter Test Vectors
4) Execute the Test
5) Analyze the Results

The electrical connections could be performed after the power-up initialization in order to allow different types of tests of the same device, or in testing different devices, without going through the power-up initialization every time; however, care must be taken to avoid shorting lines which could cause component failures in the functional tester. All of the sixty-four data lines and the four programmable clocks are tri-stated when the tester is not in test mode, but the P5V power supply is not. When the programmable voltage supplies are developed for the functional tester, they should be tri-stated when the tester is not in test mode, eliminating this problem.

Sixty-four data lines, which can be programmed in groups of eight as input or output lines during each cycle of the test, and four user-programmable clocks are provided by the functional tester, along with P5V and GND, as shown in the DUT INTERFACE schematic in Appendix B, Functional Tester Schematics. Figure 7: 2114 RAM Test,

on the following page, shows the connection of a 2114 1024x4bit Static RAM to the functional tester. This device will be used in this chapter to illustrate the operation of the functional tester. The four data lines of the 2114 are connected to a distinct eight-data-line group of the functional tester, since these lines will be programmed as input to and output from the function tester at different times while the other 2114 lines will always be outputs from the functional tester. The four data lines of the 2114 are connected to D0-D3 of the functional tester. The ten address lines of the 2114 are connected to D17-D26 of the functional tester. The write enable of the 2114 is connected to D31 (which is the most-significant bit of the upper group of the address data lines). A programmable clock could have been used for the write enable to measure the write cycle time of the 2114; however, this was not done for this test since only the functionality of the device was being tested. The chip enable of the 2114 is connected to ground. The P5V and GND were connected to the power supply inputs of the 2114. This shows a typical case of the electrical connections necessary to test a device using the functional tester.

Power-up initialization of the functional tester involves two steps. First, power must be applied to the functional tester, and the test vector downloader utility program on the Intel SDK-86 must begin execution. This program is executed by pressing the "GO" key on the SDK-86, and entering the starting address of the utility program,

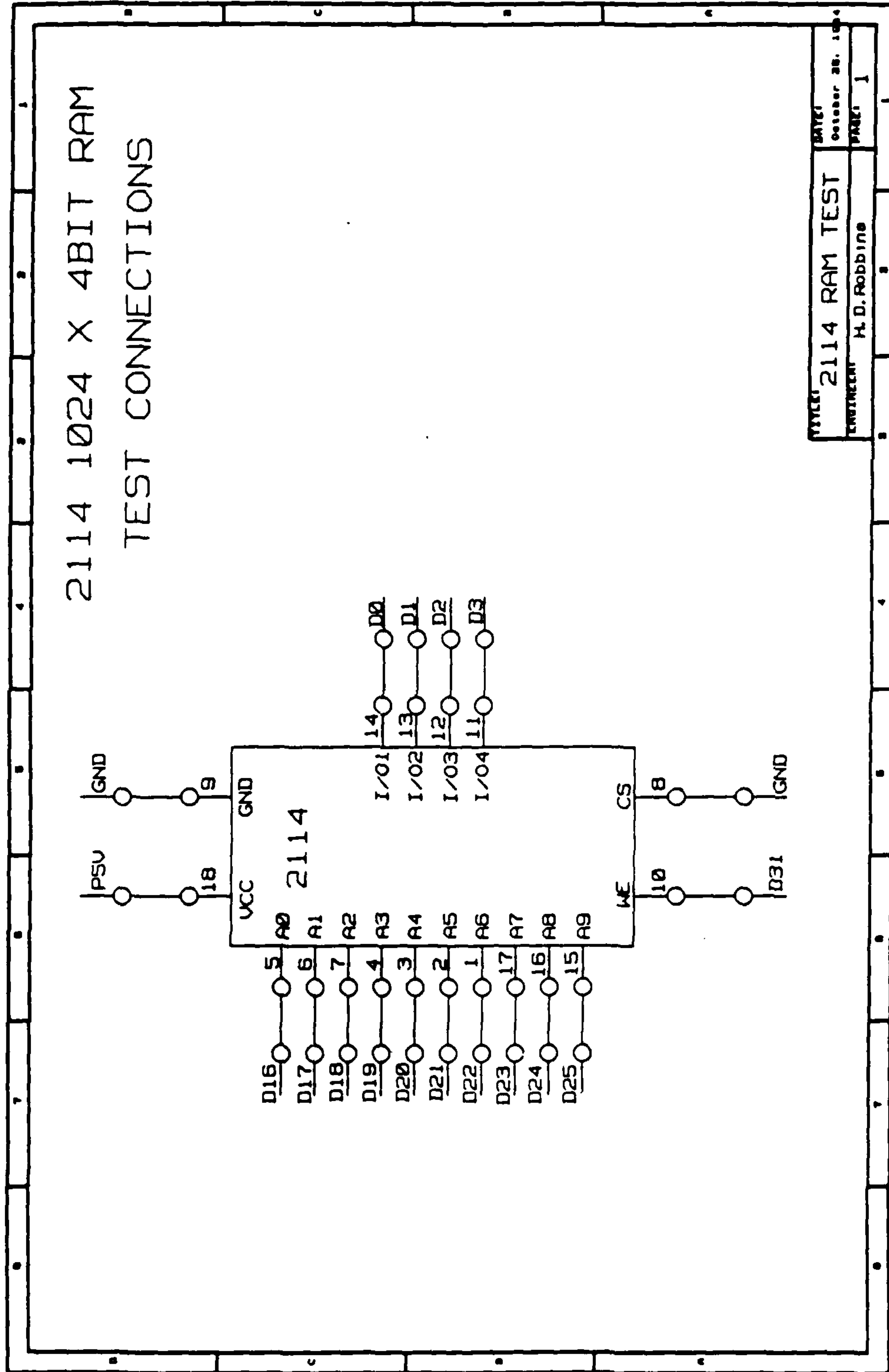



Figure 7: 2114 RAM Test

which is OFE000H. The utility program downloads the actual function tester control program from the Hewlett Packard HP9920A computer, and begins execution of that program. The second step required in power-up initialization of the functional tester involves transferring the functional tester control program to the SDK-86. To accomplish this, the HP9920A computer must be "booted-up" under the PASCAL operating system. This is accomplished by energizing the HP9920A, and selecting PASCAL as the desired operating system from the menu which appears. A program named "TRANSFER" must be executed on the HP9920A computer which transfers the functional tester 8086 machine language program located in a file on the HP9920A called "FT_CONTROL". When this is accomplished, the functional tester is ready to operate.

Entering the test vectors and executing the test are performed by a PASCAL program called "MSU_FT", which executes on the HP9920A. The program first asks the user whether the test vector data file already exists, and if it does, whether the same programmable clocks are to be re-used. This feature simplifies running the same test several times while varying the programmable clocks and test frequency in order to determine the a.c. characteristics of a device. If the test vector data file does not exist, the program requests the name to be used for the test vectors and results data files to be created. The program appends a ".DAT" to the name entered and creates a file with this name for the test vectors. An

".RES" is appended to the name entered, and a file with this name is created for the resultant data. The program next requests the number of data groups (where a group is eight data lines), with the maximum being eight data groups. For the 2114 test, three data groups were used; therefore a three was entered. The program next requests the number of data words which will input for each data group, with a maximum of 2048 data words per data group. All 1024 locations of the 2114 were to be written into, and then all 1024 of the locations were to be read back, which required 2048 addresses to be generated; therefore 2048 was entered for the number of data words per group. The same number of data words must be entered for every data group, which is the reason that 2048 data words were entered for the data lines connected to the input/output lines of the 2114 even though only 1024 of these were used. Also, inputting the test vectors is done sixteen bits at a time, which is the reason the second eight-data-line group was not used in testing the 2114, which allowed entering the 2114 data vectors separately from the 2114 address test vectors. After inputting the number of data words per data group, the program requests a result mask which is used in sorting the results. Since data groups may be inputs or outputs, determining if a data group was ever used as an input requires testing the direction bit for that data group for the entire length of the test. This is very time-consuming, and to save time a mask is entered with a "1" in a positon for a data group if that data group was ever used as an input. This allows the program to check

only the direction bits for groups which were used as input. The test of the 2114 required a result mask of "00000001" since only the first data group was ever used as input. After entering this result mask, the program requests the test vectors and then the direction bytes. The direction bytes are entered in the same format as the test vectors, with the exception that the direction bytes are only eight bits long, and constitute only one group. The test vectors may be input in binary, hexadecimal, or decimal format. Binary data must have an appended "B" (ex: 01011011B); hexadecimal decimal data must have an appended "H" (ex:0F0AEH); and decimal data may have an appended "D", or no appended letter (ex: 9086D or 9086). Sequential data may be entered by giving only the first and last values. For example, if a test required an address output which varied from 0 to 2047, the data could be entered as "[00H..7FFH]" which would enter 2,048 values. Repeated data may be entered with a multiplier; for example, to enter a count from 0 to 15 and then from 40 to 49, fifty times, the following could be entered: "(50([00H..0FH],[40..49])" which would enter 1,300 values. Repetitions within repetitions are also allowed, such that entering fifty vectors which includes ten 0FH values followed by a count from 0 to 9 could be entered: "(50(10(0FH),[0..9])". The "MSU_FT" program keeps track of the number of values provided for each data group, issues a warning if too many values are input for any one data group, and re-enters the values for that group. After entering the test vectors and direction bytes, the program requests the test frequency be entered,

which may be a value between 100KHz and 10MHz. Entering values above 5MHz are allowed, but will result in erroneous resultant data until modificatons are made to the functional tester. For the 2114 test, a value of 1MHz was used. Next, the data output enable pulse, data input strobe pulse, and four programmable clocks are programmed by using the arrow keys to modify a graphical repesentation of the clock on the screen until the desired waveform is achieved, and then presing the "RETURN" key. After entering the programmable clocks, the program transfers the test vector data file, starts the test, reads the results back to a result file, and asks whether the user wishes to view the results on the screen or have the results printed on the printer. The analysis of the results must be performed by the user.

The following represents the information entered in order to create the test vector data file used to test the 2114. This data was entered after executing the "MSU_FT" PASCAL program on the HP9920A computer.

```
DOES THE TEST VECTOR DATA FILE ALREADY EXIST?
>>N
ENTER THE NAME TO BE USED FOR THE FILES:
>>RAM2114
HOW MANY DATA GROUPS WILL BE USED?
>>4
```

HOW MANY DATA WORDS PER DATA GROUP WILL BE USED?

>>2048

ENTER AN 8-BIT BINARY NUMBER WITH A "1" FOR EACH DATA GROUP WHICH CONTAINS TEST RESULTS:

>>00000001

ENTER THE VALUES OF THE TEST VECTOR DATA:

DATA GROUP NUMBERS 1 & 2 WORDS LEFT = 2048:

>>(512(0AH,05H))

DATA GROUP NUMBERS 1 & 2 WORDS LEFT = 1024:

>>(1024(00H))

DATA GROUP NUMBERS 3 & 4 WORDS LEFT = 2048:

>>[00H..3FFH]

DATA GROUP NUMBERS 3 & 4 WORDS LEFT = 1024:

>>[8000H..83FFH]

PLEASE ENTER THE "DIRECTION" WORDS, WITH THE LEAST SIGNIFICANT BIT AS THE DIRECTION FOR DATA GROUP NUMBER 1:

DIRECTION WORDS LEFT = 2048:

>>(1024(00H))

DIRECTION WORDS LEFT = 1024:

>>(1024(01H))

PLEASE ENTER THE DESIRED TEST FREQUENCY IN HZ:

>>1000000

The program then requests the values necessary to achieve the desired waveforms for the programmable clocks as indicated previously. After inputting the programmable clocks' values, the

program transfers the test vector data file to the functional tester. At the completion of the test, the functional tester transfers the results back to the HP9920A computer which loads them in a file named "RAM2114.RES". This test was run on a 2114 with complete success; however, due to the length of the results (which consisted of 1024 [AH,5H] as the input, and the same as the result), the results of this test are not presented. Another test was run which consisted of twenty values, which were (0AH,05H,[00H..0FH],0AH,05H). The results of this test, along with the electrical connections and test results of tests run on a 7404 HEX INVERTER and a 7482 2-BIT BINARY FULL ADDER, are presented in Appendix E, TEST RESULTS.

# CHAPTER V

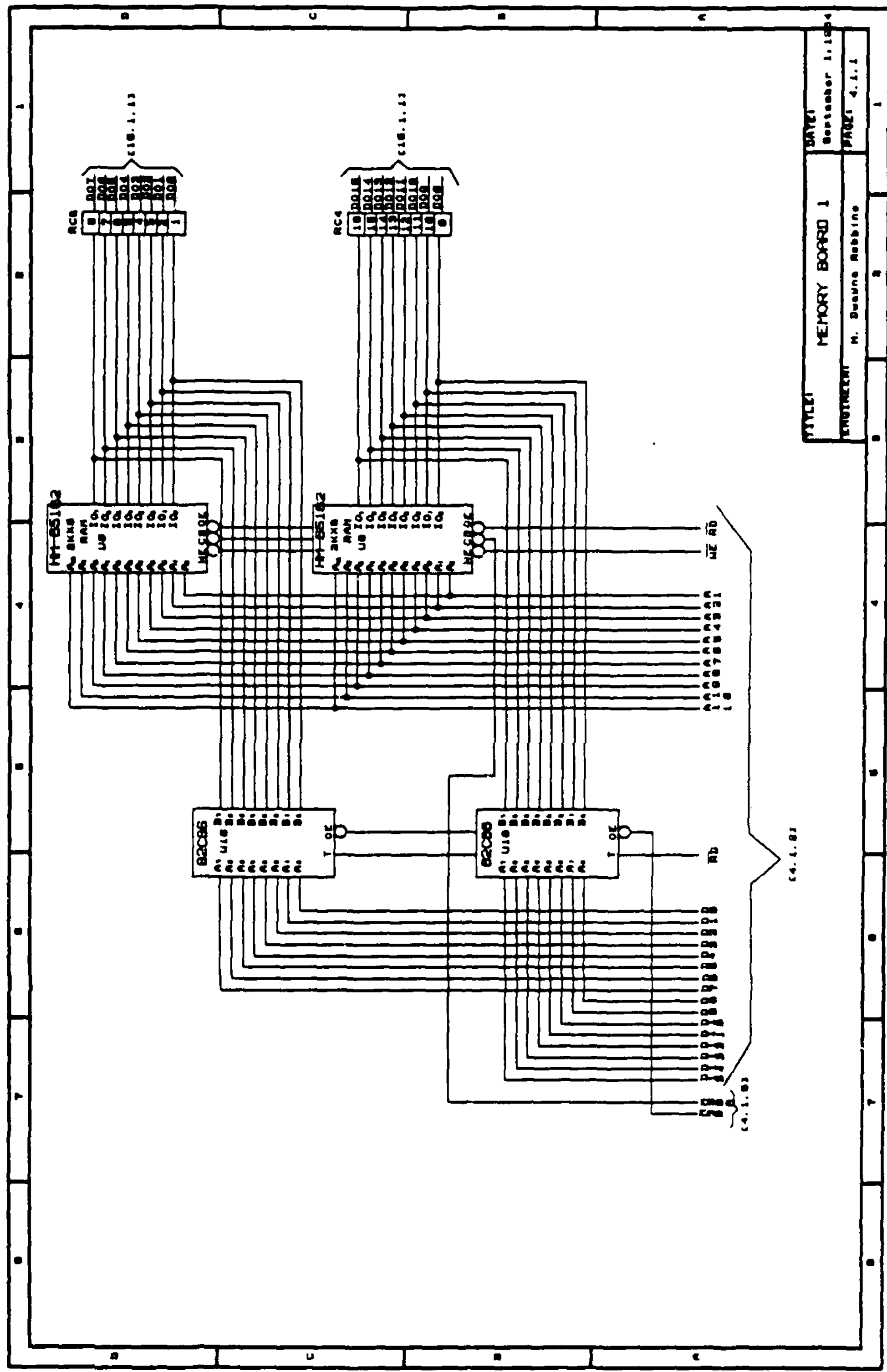
# EVALUATION AND DISCUSSION

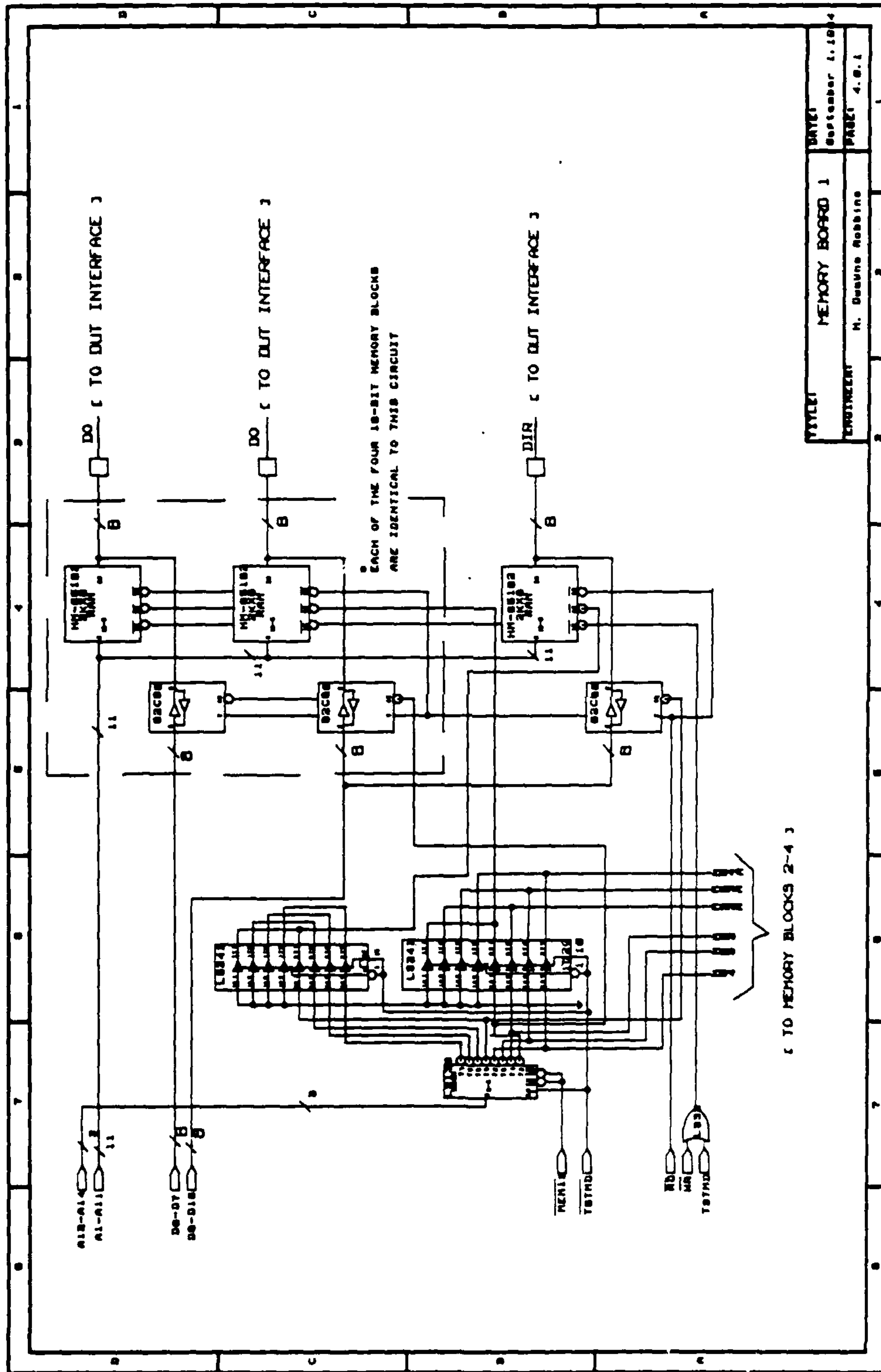
This thesis presented the design and implementation of a high speed, integrated circuit functional tester, primarily for use in a university setting. As indicated in Chapter I, major design work in very large scale integration (VLSI) is being performed in universities. One of the major limitations in this endeavor is the ability to adequately test the fabricated IC's. The functional tester presented here has the following capabilities:

- 100KHz - 5MHz Test Frequency
- 64 Input/Output Data Lines
- Direction Control of the Data Lines in Groups of Eight
- Four User-Programmable Clocks
- User-Programmable Data Output Enable and Data Input Strobe Pulses
- "User-Friendly" Input of Test Vectors
- Sorted Results in Hardcopy (Printer) or Softcopy (Floppy or Hard disk)

The hardware design and implementation included the interface between the functional tester and the Hewlett Packard computer; the memory boards to buffer the test vectors and resultant data; the test-phase control timing board; the programmable clocks board; and

the pipeline registers board interfacing the functional tester to the device under test. The software design and implementation included the 8086 assembly language programs to control the functional tester and the PASCAL programs on the Hewlett Packard HP9920A computer to transfer data to/from the functional tester and allow "user-friendly" input of the test vectors. The operational requirements of the functional tester were presented, along with test results from the test of a 2114 1024x4BIT RAM, a 7404 HEX INVERTER, and a 7482 2-BIT FULL ADDER. These results indicate that the functional tester operates according to the specifications indicated above, which are adequate to functionally test most of the VLSI designs being done in universities at present, since these very large scale integrated circuits can usually be seperated into smaller functional circuits similar to the circuits tested. The only drawback to the functional tester is the requirement that the user microcode all of the test vectors required to adequately test the device. This is not a major drawback for this application however, since most of the designs are new, and no assembler exists for these circuits. Testing parts of the circuits could be handled by generating standard tests for the various functions implemented by the VLSI circuit. For instance, the test of the RAM device indicates the ease in developing test patterns for memory devices, and this test, or a similar test of a memory device such as the "walking-ones" test, could be developed for the memory storage functions of other devices to be tested.


MEMORY BOARD 1
HM-65162
82C86


( TO DUT INTERFACE )
( TO DUT INTERFACE )
( TO DUT INTERFACE )
EACH OF THE FOUR 16-BIT MEMORY BLOCKS
ARE IDENTICAL TO THIS CIRCUIT
( TO MEMORY BLOCKS 2-4 )
TITLE:
MEMORY BOARD 1
ENGINEER:
M. Duane Robbins
DATE:
PAGE:

TO HP COMPUTER

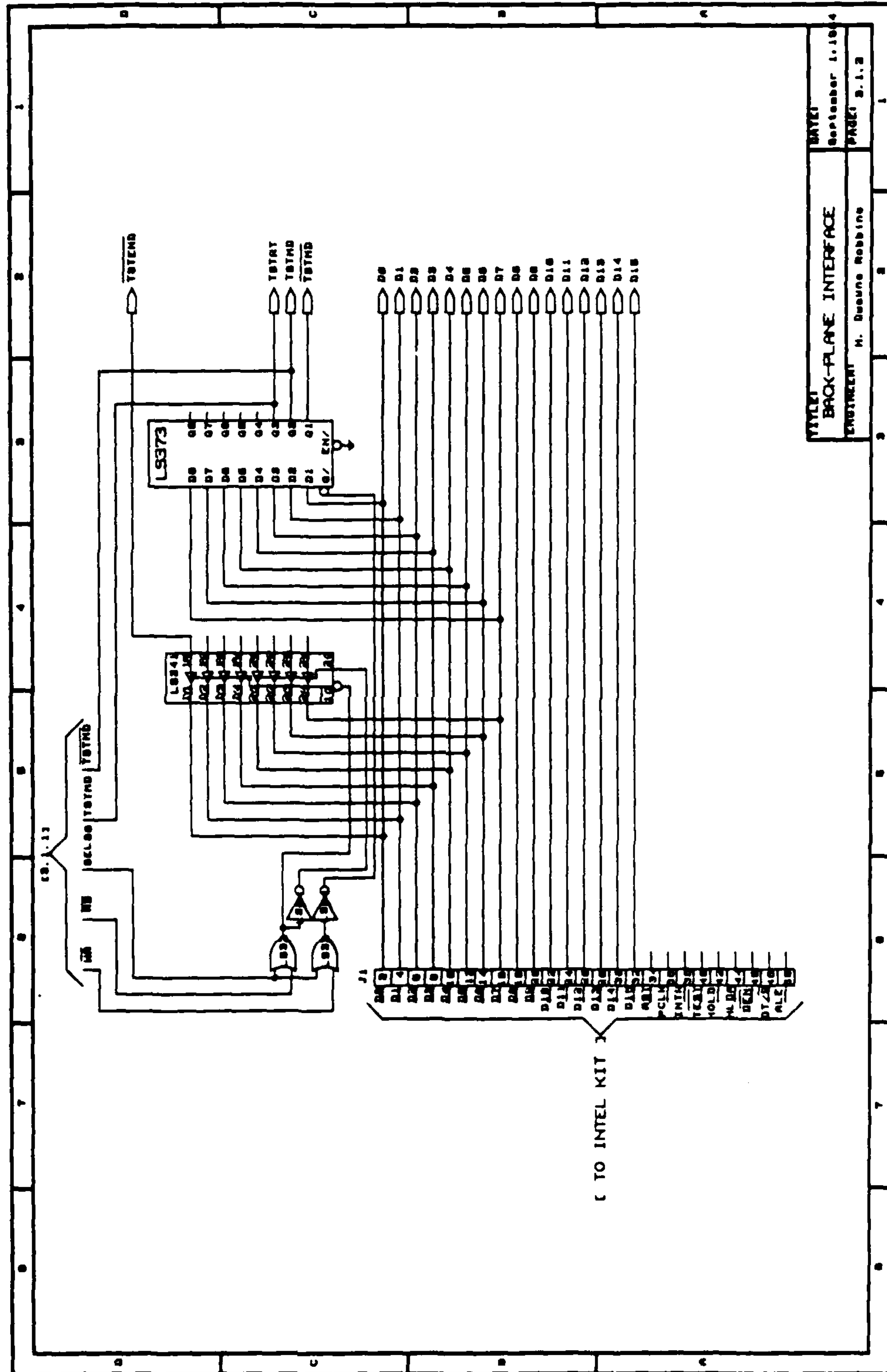
INTEL 8086 KIT

TITLE: GPIO INTERFACE

ENGINEER: H. Duwane Robbins

DATE: September 1,1984

PAGE: 3.3.1


BACK-PLANE INTERFACE
LS373
TO INTEL KIT

(TO INTEL KIT)

J3

BHE, A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A17, A18, A19, M/IO, RD, WR, INTA

LS241

A0, A1, A2, A3, A4, A5, A6, A7

A8, A9, A10, A11, A12, A13, A14, A15

A16, A17, M/IO, RD, WR, BHE

AM25LS 2521

MSEL2

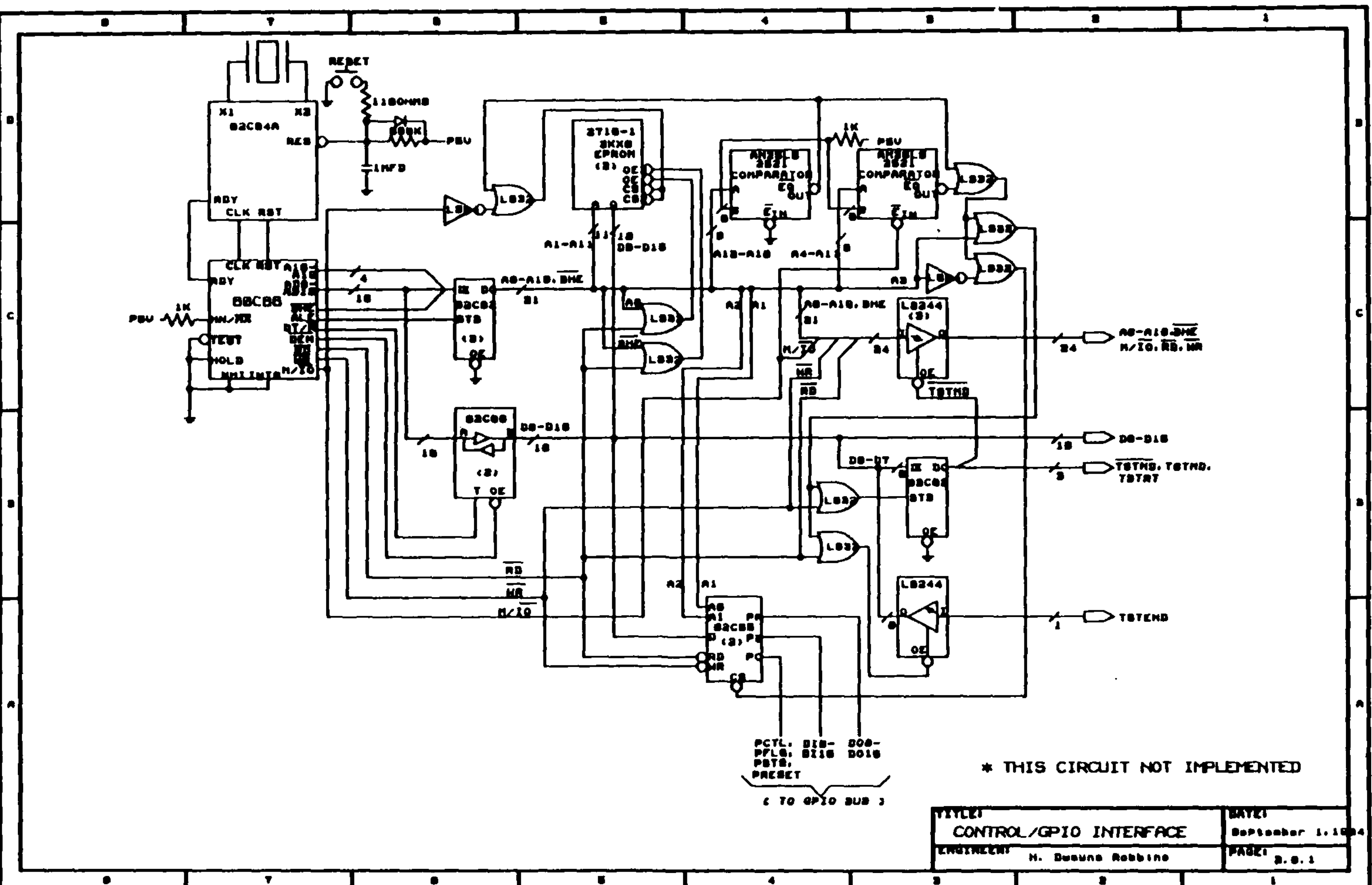
MSEL3

PSV 1K

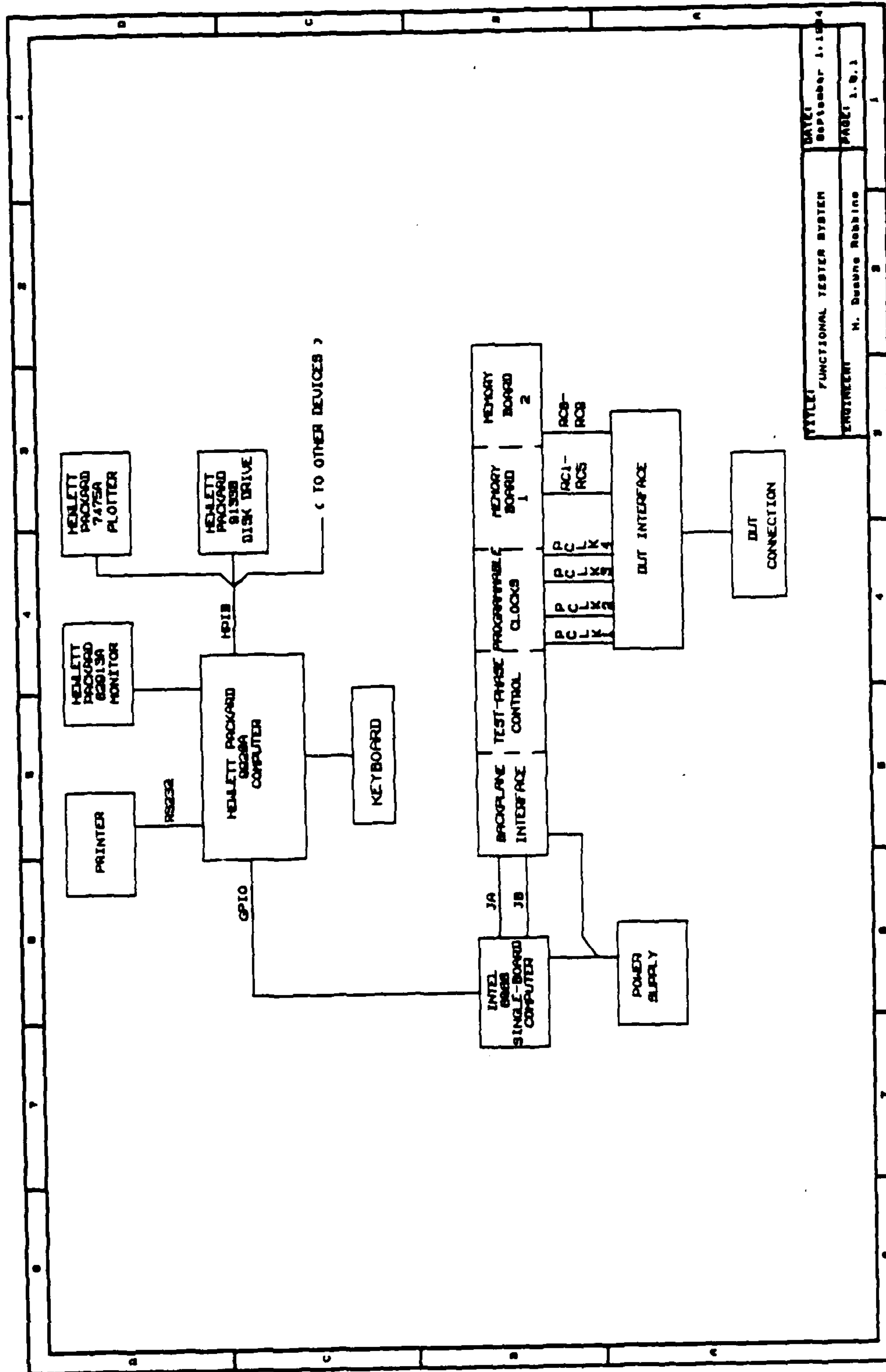
WR RD SELSS TSTMD TSTMD

(3.1.2)

| TITLE: BACK-PLANE INTERFACE | DATE: September 1, 1984 |
| --- | --- |
| ENGINEER: H. Dwayne Robbins | PAGE: 3.1.1 |



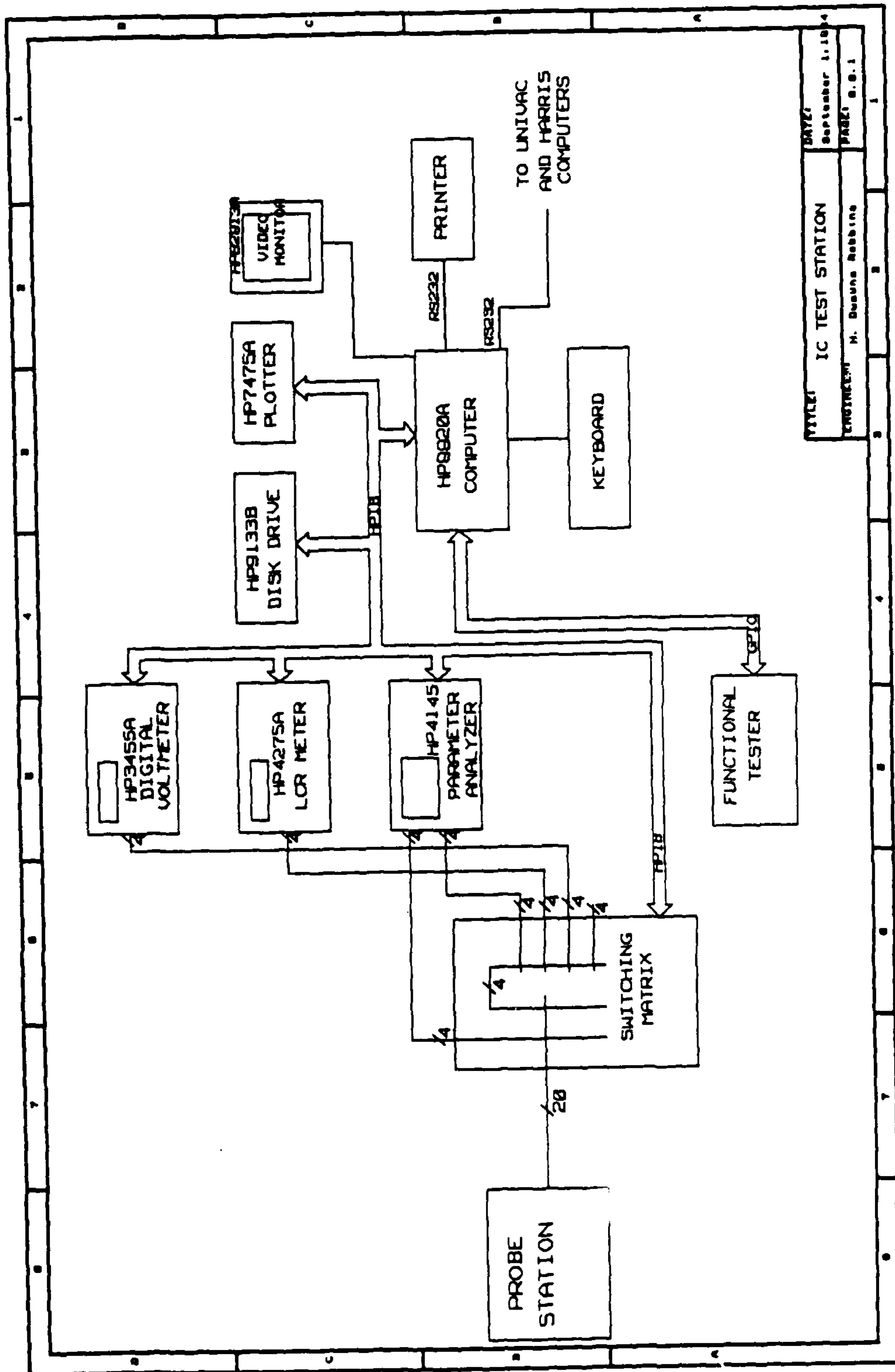
RESET
82C84A
8086
8282
8286
2716-1
EPROM
COMPARATOR
LS244
LS32
A0-A19, BHE
M/IO, RD, WR
D0-D15
TSTMD
TSTEND
PCTL, PFLG, PSTS, PRESET
DI0-DI15
DO0-DO15
( TO GPIO BUS )
* THIS CIRCUIT NOT IMPLEMENTED
TITLE:
CONTROL/GPIO INTERFACE
ENGINEER:
DATE:
PAGE:



HEWLETT PACKARD 7475A PLOTTER
HEWLETT PACKARD 9133B DISK DRIVE
( TO OTHER DEVICES )
HPIB
HEWLETT PACKARD 82913A MONITOR
HEWLETT PACKARD 9826A COMPUTER
KEYBOARD
PRINTER
RS232
GPIO
INTEL 8088 SINGLE-BOARD COMPUTER
POWER SUPPLY
JA
JB
BACKPLANE INTERFACE
TEST-PHASE CONTROL
PROGRAMMABLE CLOCKS
MEMORY BOARD 1
MEMORY BOARD 2
RC1-RC5
RC6-RC8
PCLK1
PCLK2
PCLK3
PCLK4
DUT INTERFACE
DUT CONNECTION
TITLE: FUNCTIONAL TESTER SYSTEM
ENGINEER: W. Duane Robbins
PAGE: 1.0.1

# APPENDIX B

## FUNCTIONAL TESTER SCHEMATICS


TITLE:
IC TEST STATION
DATE:
ENGINEER:
PAGE:
TO UNIVAC
AND HARRIS
COMPUTERS
PRINTER
VIDEO
MONITOR
RS232
RS232
HP7475A
PLOTTER
HP9133B
DISK DRIVE
HPIB
HP9820A
COMPUTER
KEYBOARD
GPIO
FUNCTIONAL
TESTER
HP3455A
DIGITAL
VOLTMETER
HP4275A
LCR METER
HP4145
PARAMETER
ANALYZER
HPIB
4
SWITCHING
MATRIX
20
PROBE
STATION

# APPENDIX A

# TEST STATION

Appendix B) with separate patch wires. The patch-plug connections have been tested at 5MHz with no loss of signals, and should provide adequate connections at 10MHz test frequencies.

The primary modification to the packaging of the functional tester involves replacing the wire-wrapped boards with printed-circuit boards. This would greatly enhance the operation and reliability of the functional tester, but should be implemented over a period of time as the other modifications are included.

Several of the modifications mentioned throughout this thesis are not reiterated in this chapter, but would serve to enhance the overall operation of the functional tester. The modifications presented in this chapter are considered to be the most important, and should be considered first.

performed using a high-level language, with the results easily accessible in a sorted file.

The programmable voltage sources and the single-board microprocessor controller could be implemented with additional development effort. The programmable voltage sources are required to test CMOS devices at their operating voltage, and to test devices at voltages other than their operating voltage in order to determine how the voltage differential affects timing characteristics. The design of the programmable voltage board would include latches to contain the binary representations of the desired output voltages; digital-to-analog converters to produce the desired voltages; and analog voltage regulators to regulate the voltage while supplying the necessary current. The single-board microprocessor controller would replace the Intel SDK-86 single-board computer, for cost and space improvements. The cost savings would affect future functional testers to be built, but the space improvements would greatly enhance the packaging of the existing functional tester.

The mechanical modifications to allow easier electrical connections when connecting a device to be tested to the functional tester would require only minor modifications to the patch-plug-type board which now exists. Rather than the patch wires being permanently connected on one end as is now the case, the board should include patch receptacles on both sides (refer to the DUT BOARD schematic in

problem, which is the timing delays of the TTL devices used to implement the test-phase control signals and the programmable clocks. Using ECL devices to implement these circuits was studied, but would require additional expenses for the devices and more precision in developing the prototype functional tester than time allowed. A VLSI design which will provide a programmable pulse with a resolution of 10nanoseconds or better, based on a reset strobe each cycle, has been considered and appears feasible. A device of this nature would allow generation of the test-phase control signals and the programmable clocks at the resolution necessary to allow tests up to 10MHz, or more, with adequate precision to measure the timing characteristics of the device under test.

The most obvious solution to computer analysis of test results involves the "known-good" method. For tests involving the same test for several identical devices, the test would be executed on a device which was known to be good (by hand analysis of the test results), and then the results of this test would be compared by the computer for the remaining devices. Another area of computer analysis would be statistical analysis and records of the failure modes resulting from the tests. These analysis aids were not developed, based in part on time constraints and on the desire to produce the most general-purpose functional tester possible. The addition of these analysis aids is considered a simple matter for specific circuits, since all of the test results analysis is

## Enhancements

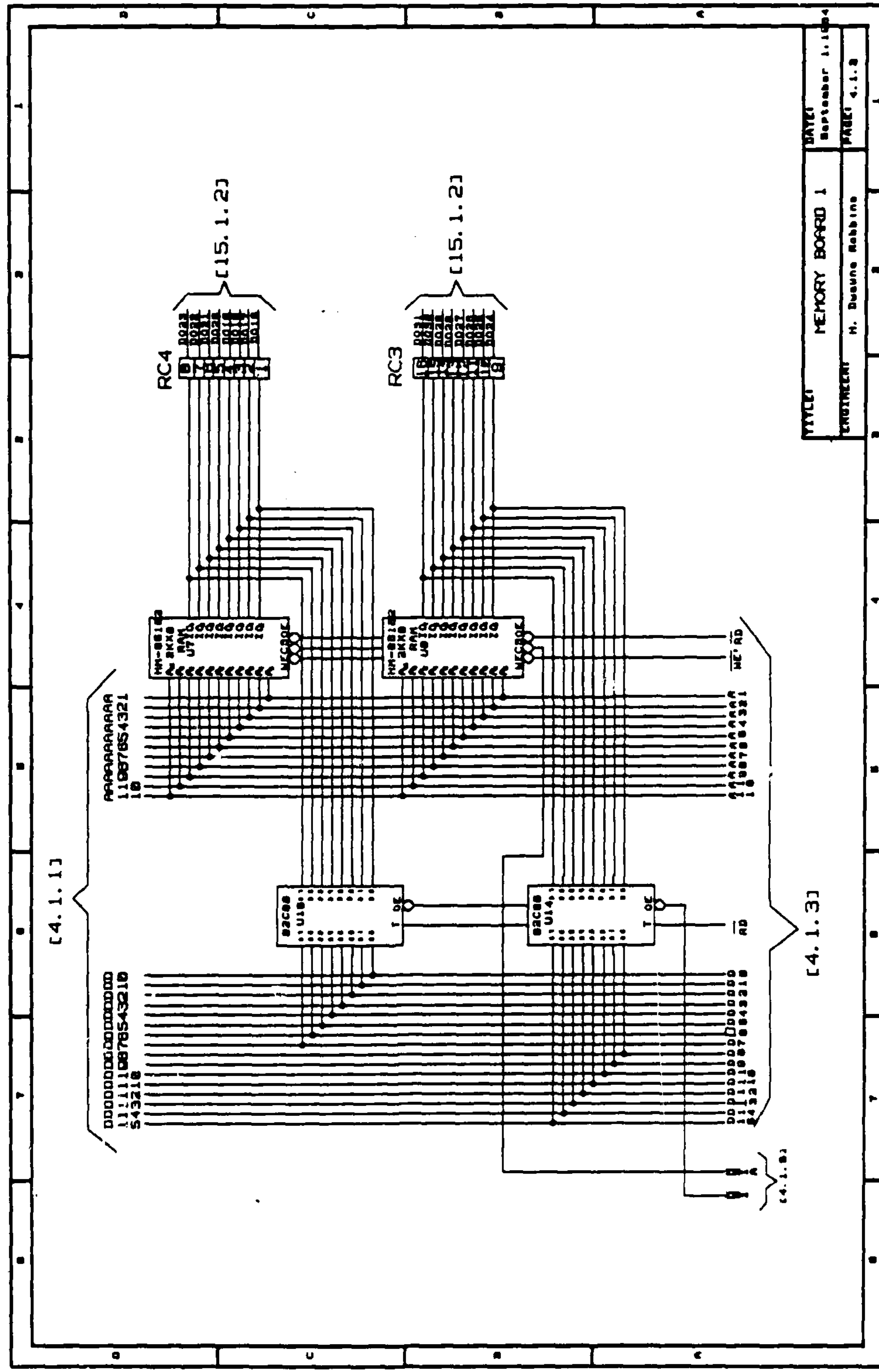
The abilities of the functional tester presented above are adequate to functionally test, at or above their operating speed, most of the circuits being developed in universities at present. The ability to functionally test the higher speed VLSI circuits, and the ability to test the timing characteristics of present VLSI circuits, requires the tester to perform beyond its present level of capability in some areas. These modifications can be functionally classified as:

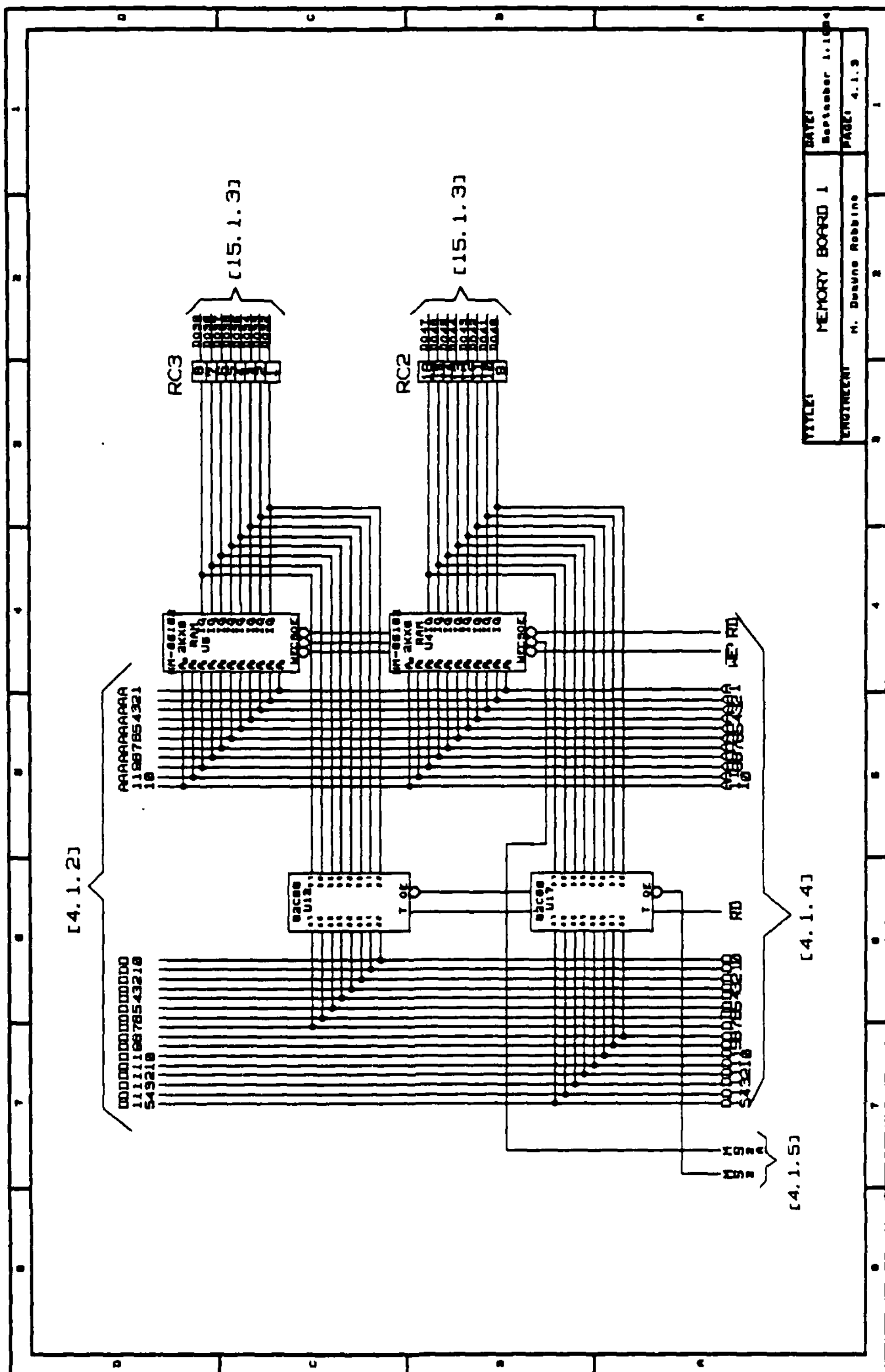
- Test Frequencies of at least 10MHz
- Programmable Clocks With 10nanosecond Resolution
- Automated Generation of Test Vectors
- Computer Analysis of Results

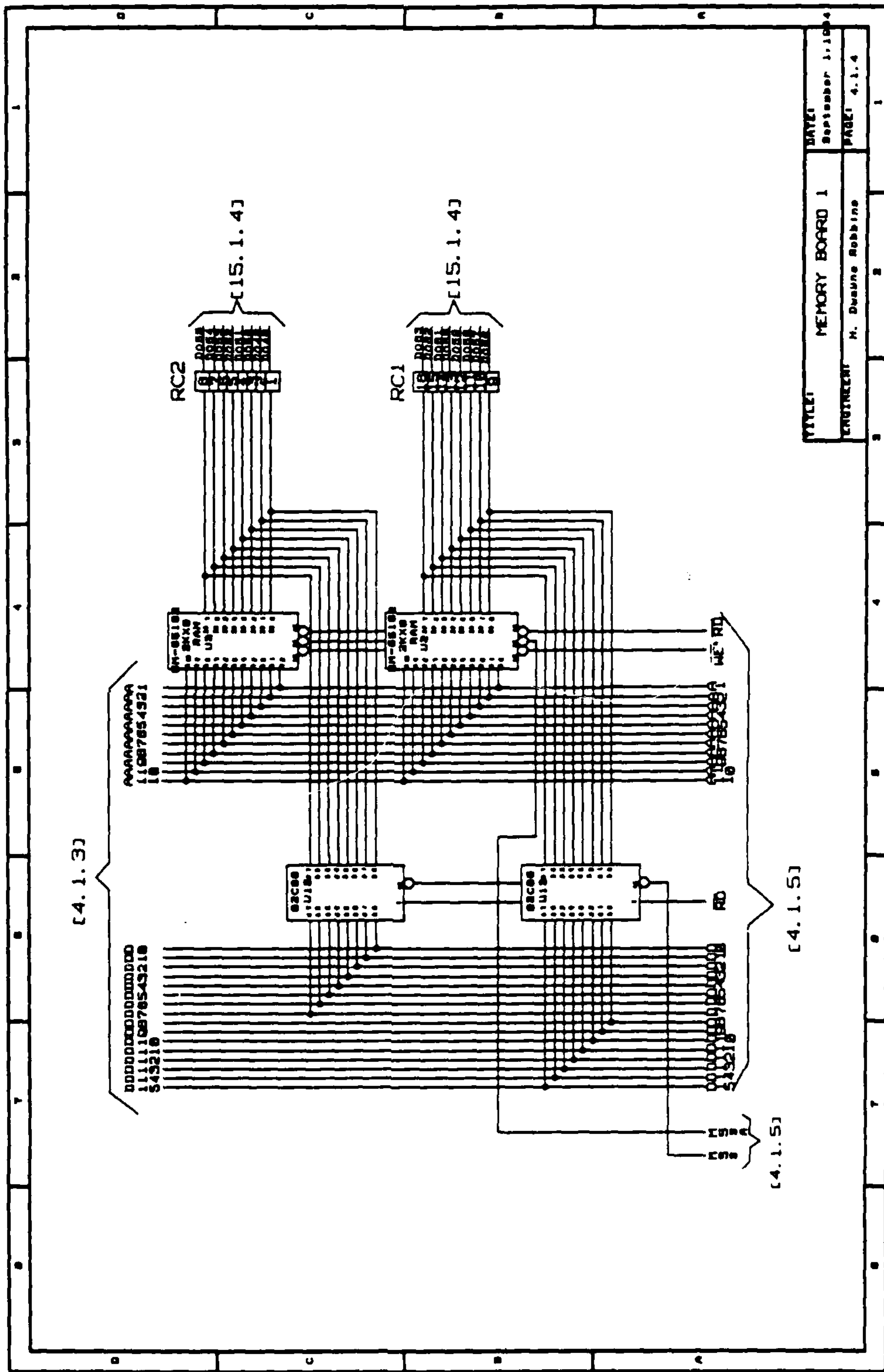
In addition to these requirements, other modifications which are needed to make the functional tester perform at the operational level desired and provide easier operation include:

- Two or More Programmable Voltage Sources
- Mechanical Modifications to Allow Easier Electrical Connections
- Single Microprocessor Control Board
- Mechanical Modifications to Better Package Tester

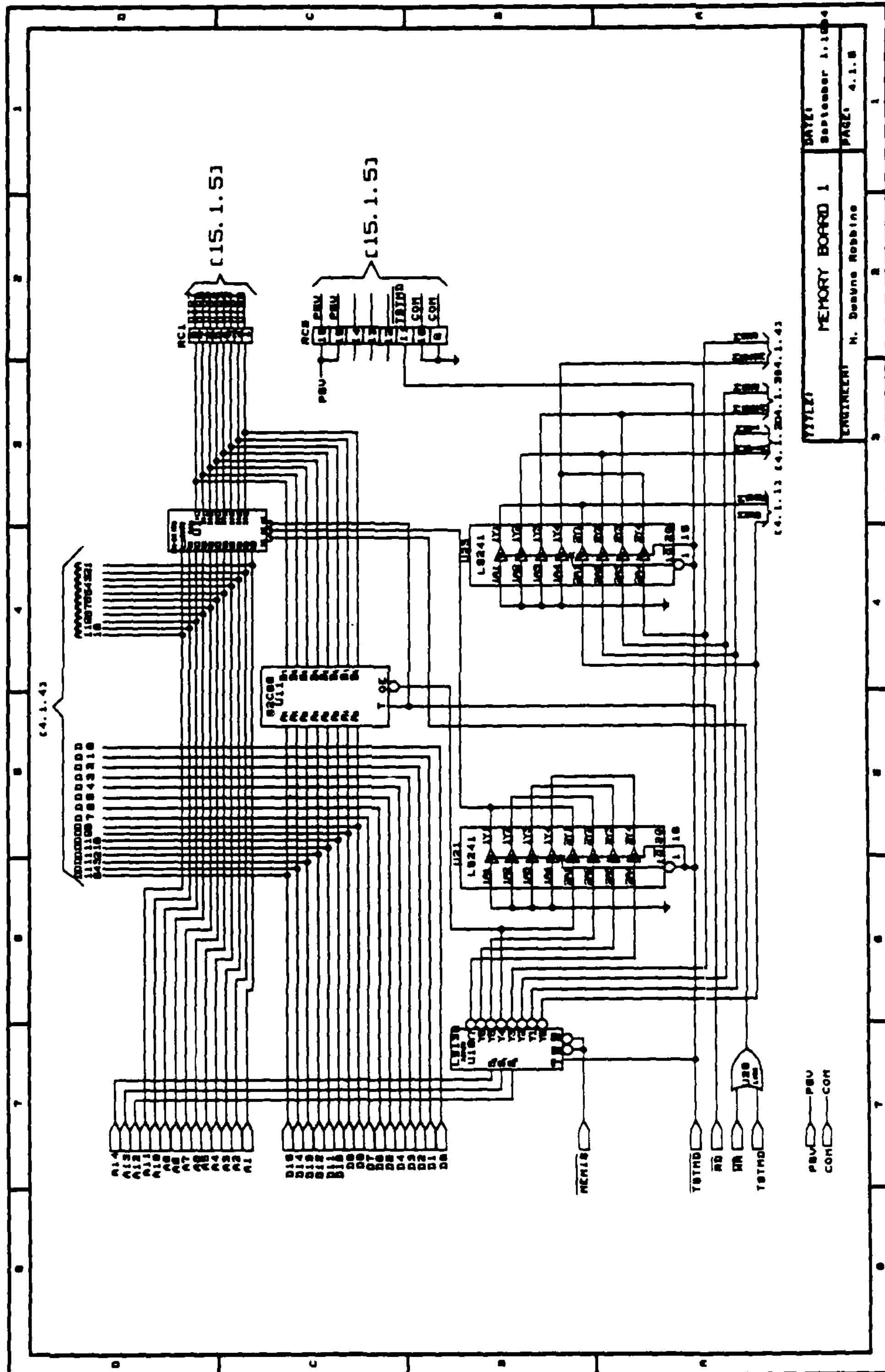
The limit on test frequency at 5MHz and the limit of the programmable clocks resolution at 50nanoseconds relates to the same


TITLE:
MEMORY BOARD 1
ENGINEER:
DATE:
PAGE:
RC4
RC3
[15.1.2]
[15.1.2]
[4.1.1]
[4.1.3]


[15.1.3]
[15.1.3]
RC3
RC2
[4.1.2]
[4.1.4]
[4.1.5]
MEMORY BOARD 1



TITLE:
MEMORY BOARD 1
ENGINEER:
M. Duane Robbins
PAGE:
4.1.4
RC2
RC1
[15.1.4]
[15.1.4]
[4.1.3]
[4.1.5]
[4.1.5]


TITLE:
MEMORY BOARD 1
DATE:
PAGE:
4.1.8
ENGINEER:
LS241
LS241
U11
U21
[15.1.5]
[15.1.5]
[4.1.4]
RC1
RC8
PBV
COM
TSTMD
MEMIS
RD
WR

A12-A14
A1-A11
D0-D7
D8-D15
MEM2S
TSTMD
WR
RD
TSTMD

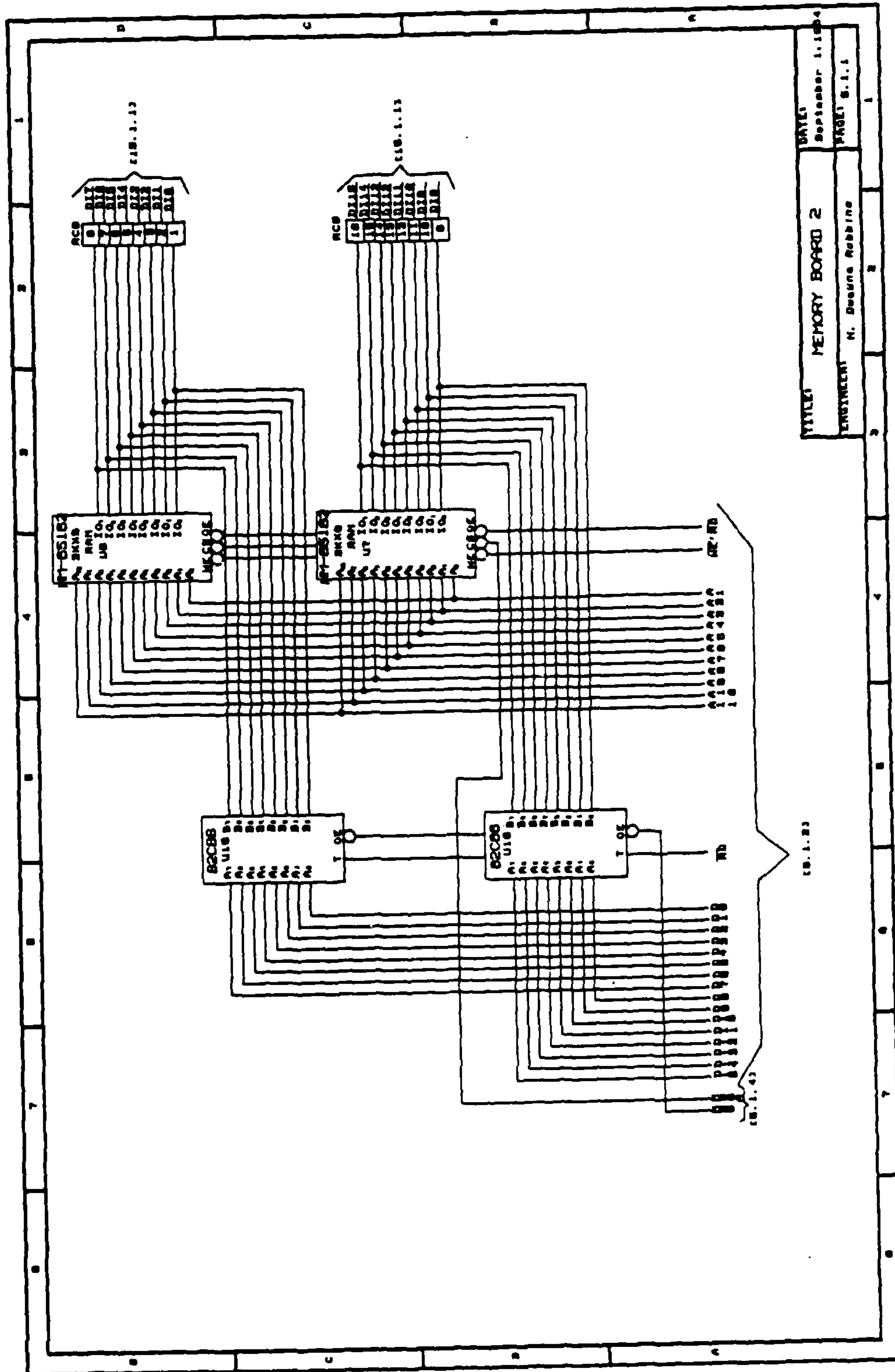
MM-65162 2KX8 RAM
82C86
LS138
LS241

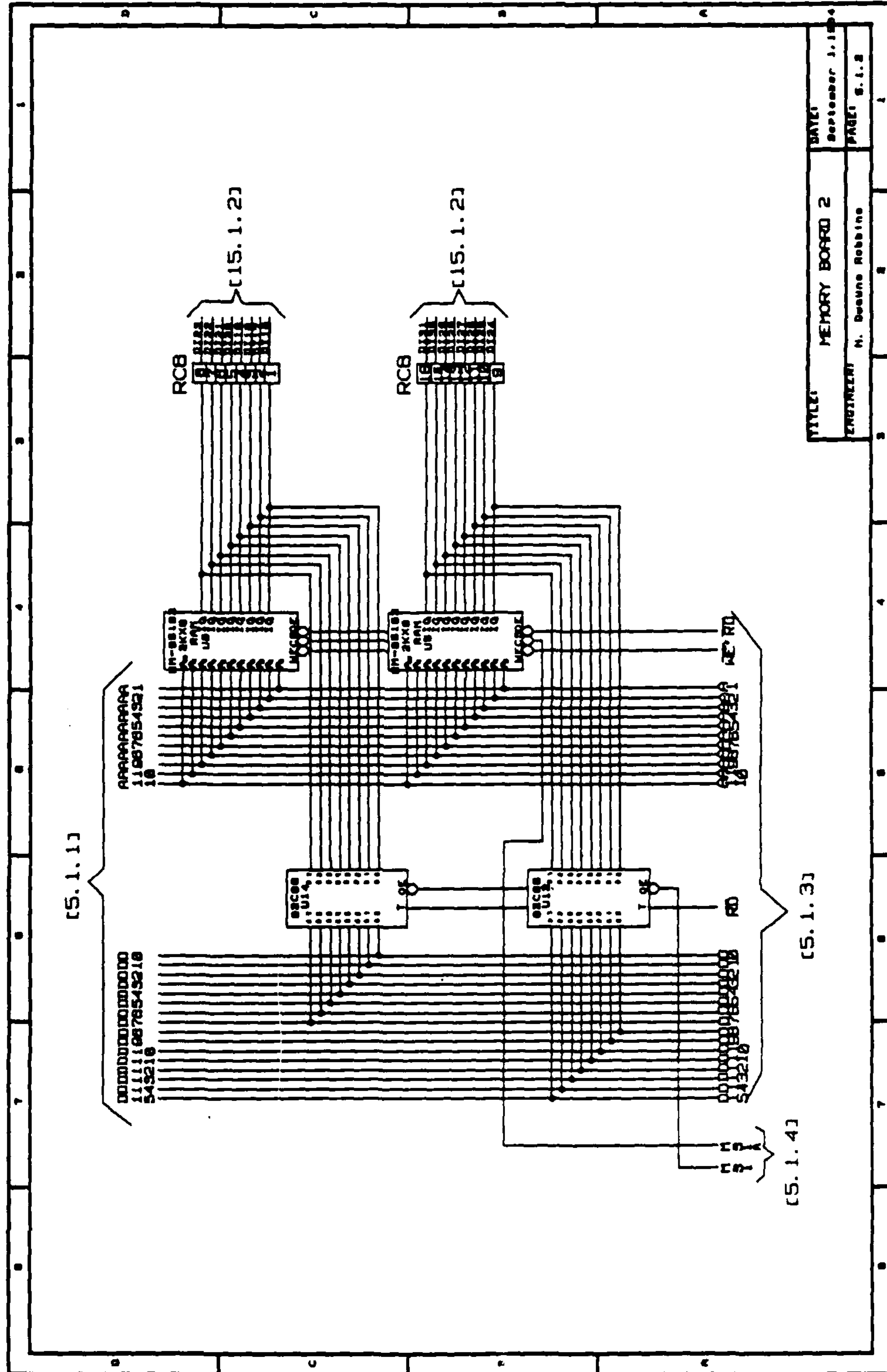
DI [ TO DUT INTERFACE ]
DI [ TO DUT INTERFACE ]

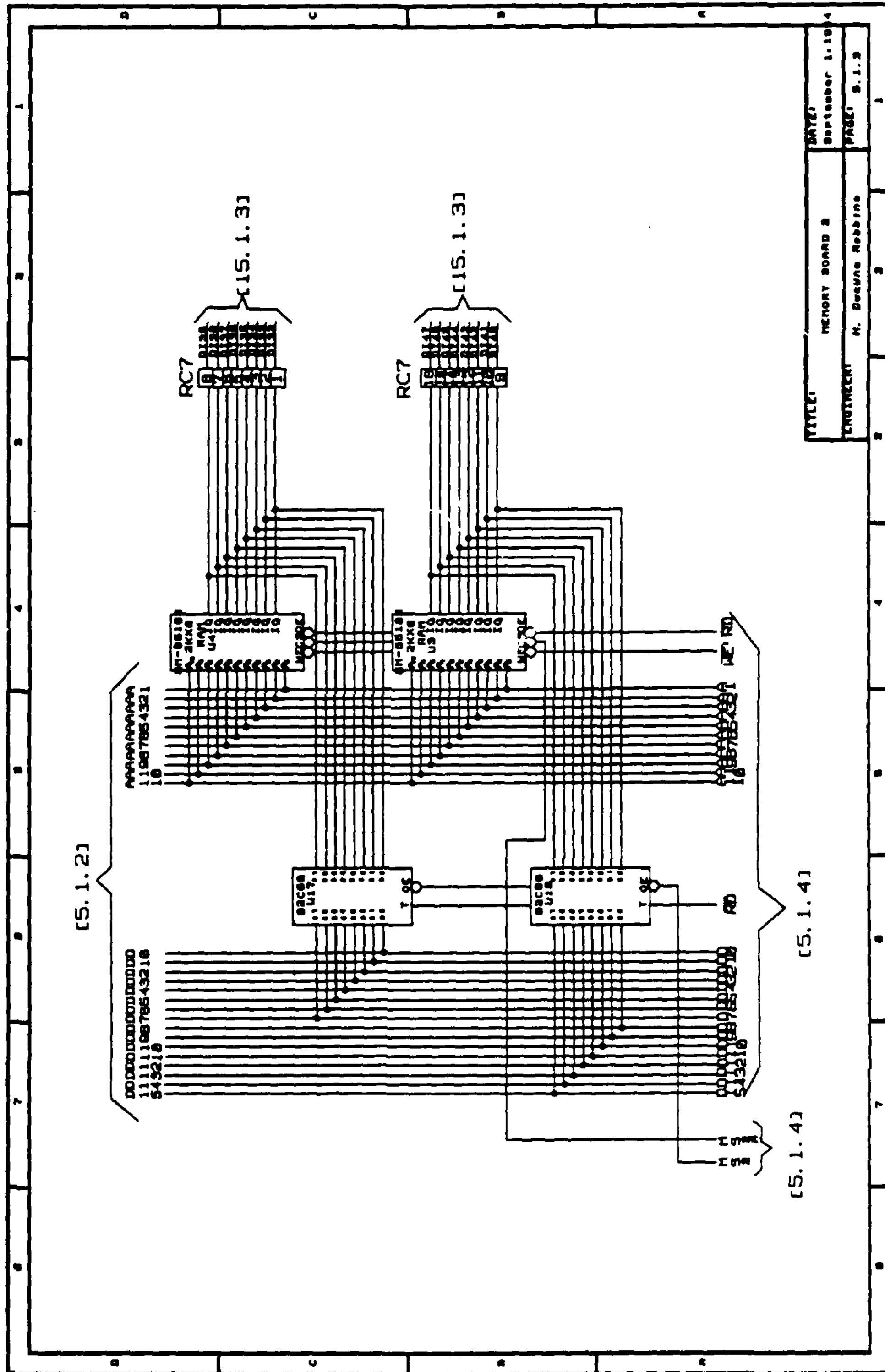
EACH OF THE FOUR 16-BIT MEMORY BLOCKS
ARE IDENTICAL TO THIS CIRCUIT

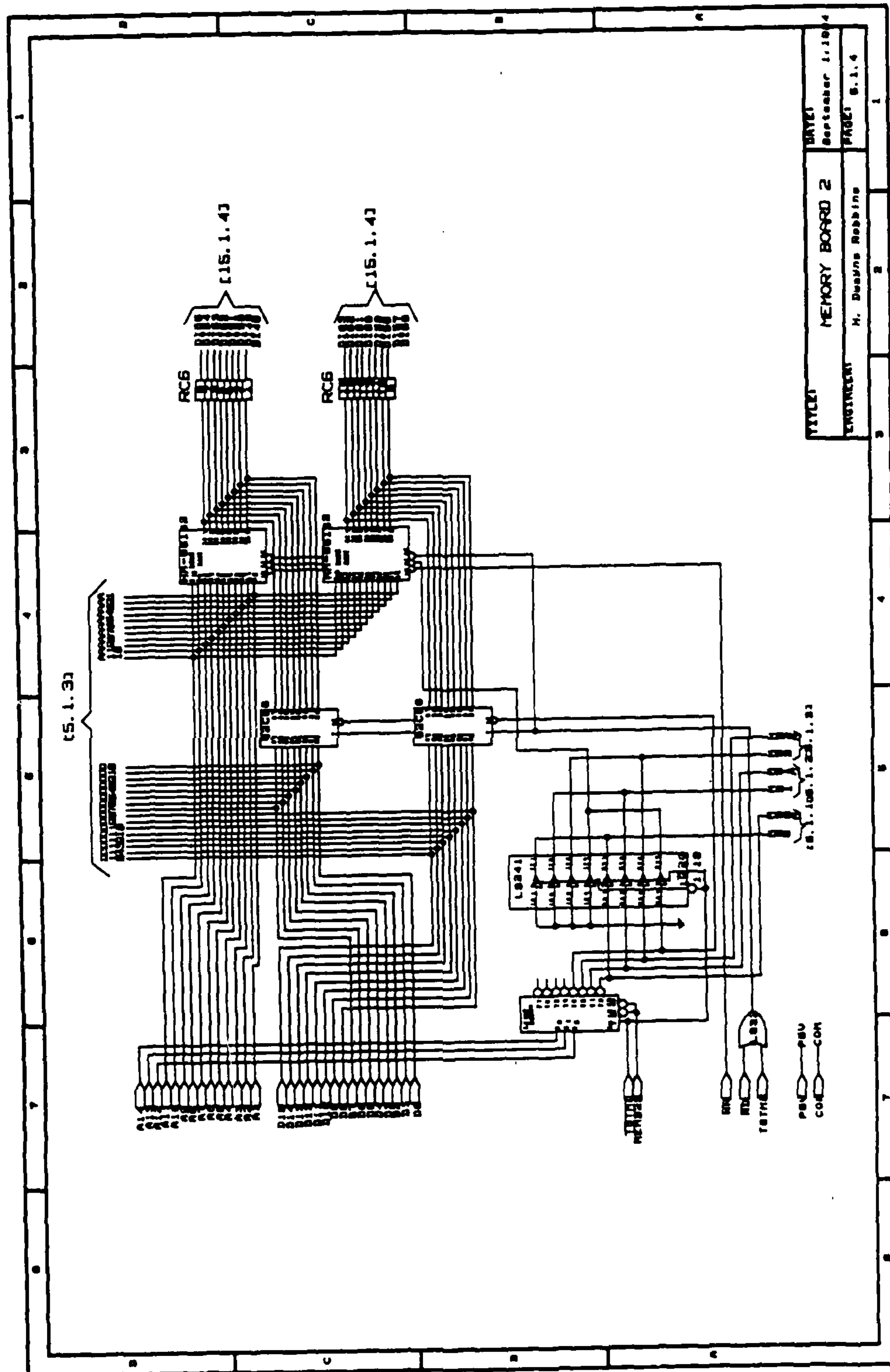
MS2A
MS3A
MS4A
MS2
MS3
MS4
[ TO MEMORY BLOCKS 2-4 ]

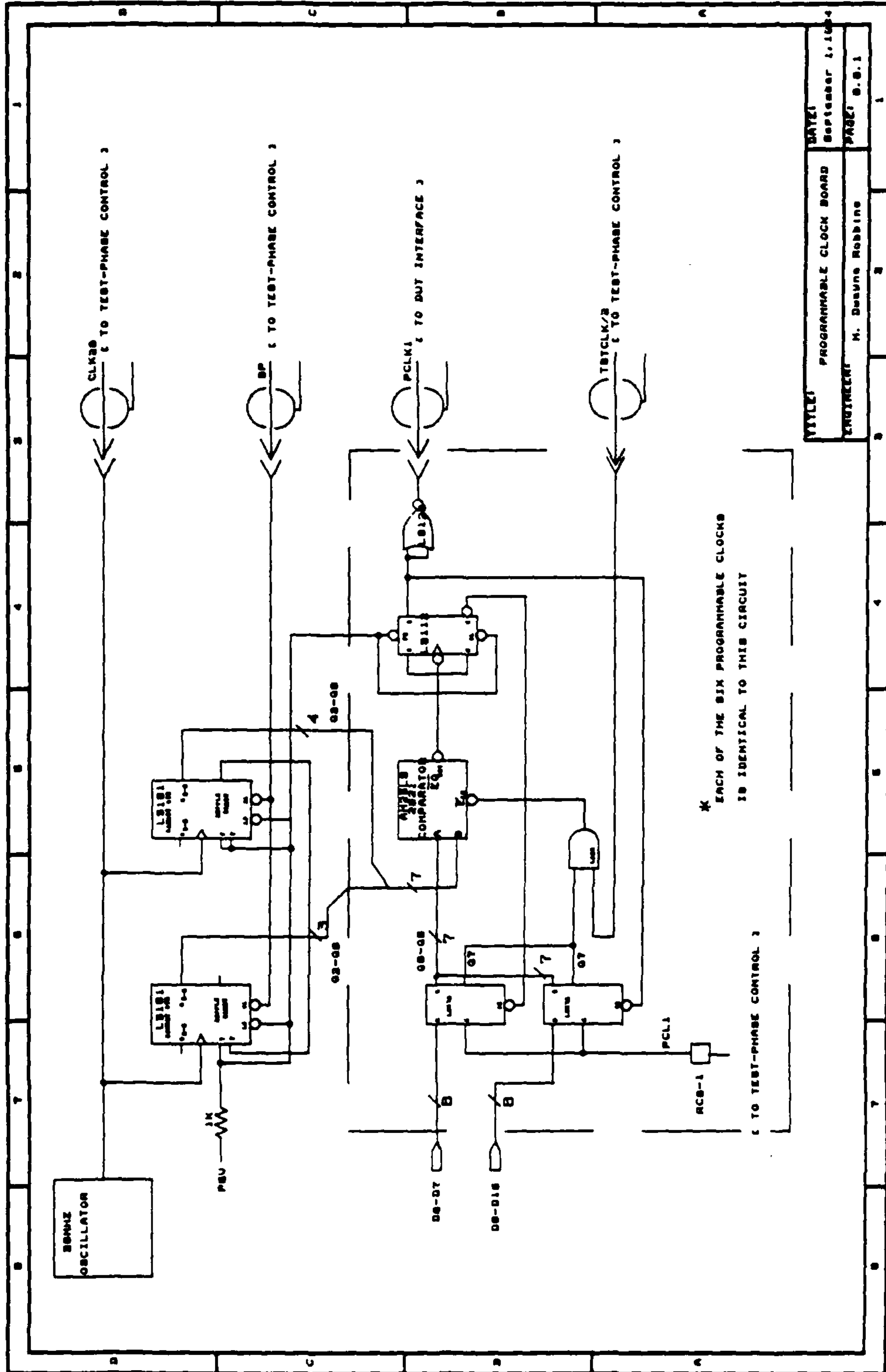
| TITLE: MEMORY BOARD 2 | DATE: September 1:1984 |
| --- | --- |
| ENGINEER: M. DuaWne Robbins | PAGE: 5.5.1 |

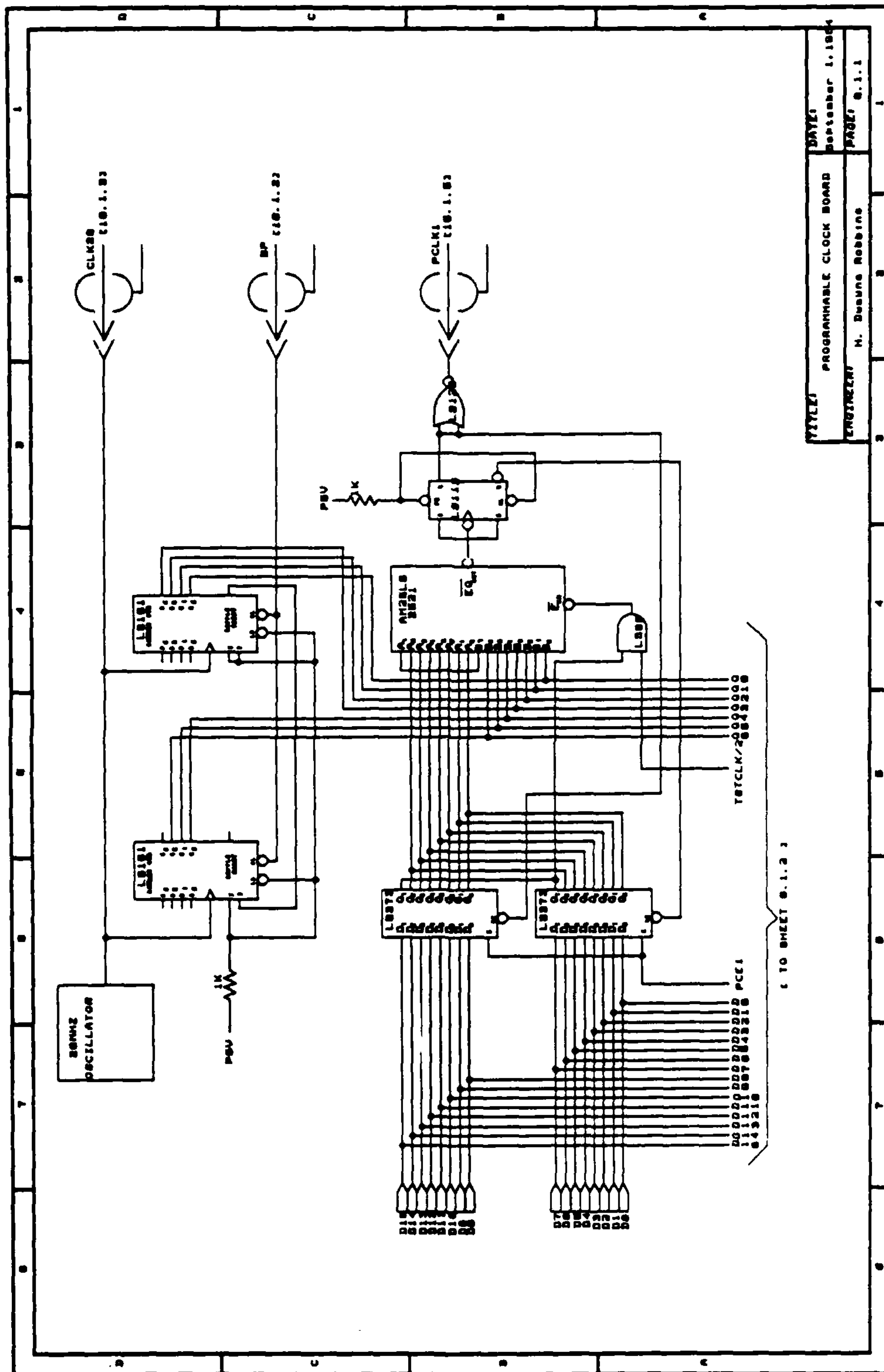

TITLE:
MEMORY BOARD 2
ENGINEER:
DATE:
PAGE:


[5.1.1]
RC8
[15.1.2]
RC8
[15.1.2]
U14
U13
RD
WE
[5.1.4]
[5.1.3]
TITLE: MEMORY BOARD 2
ENGINEER: N. Dwaune Robbins
DATE:
PAGE:

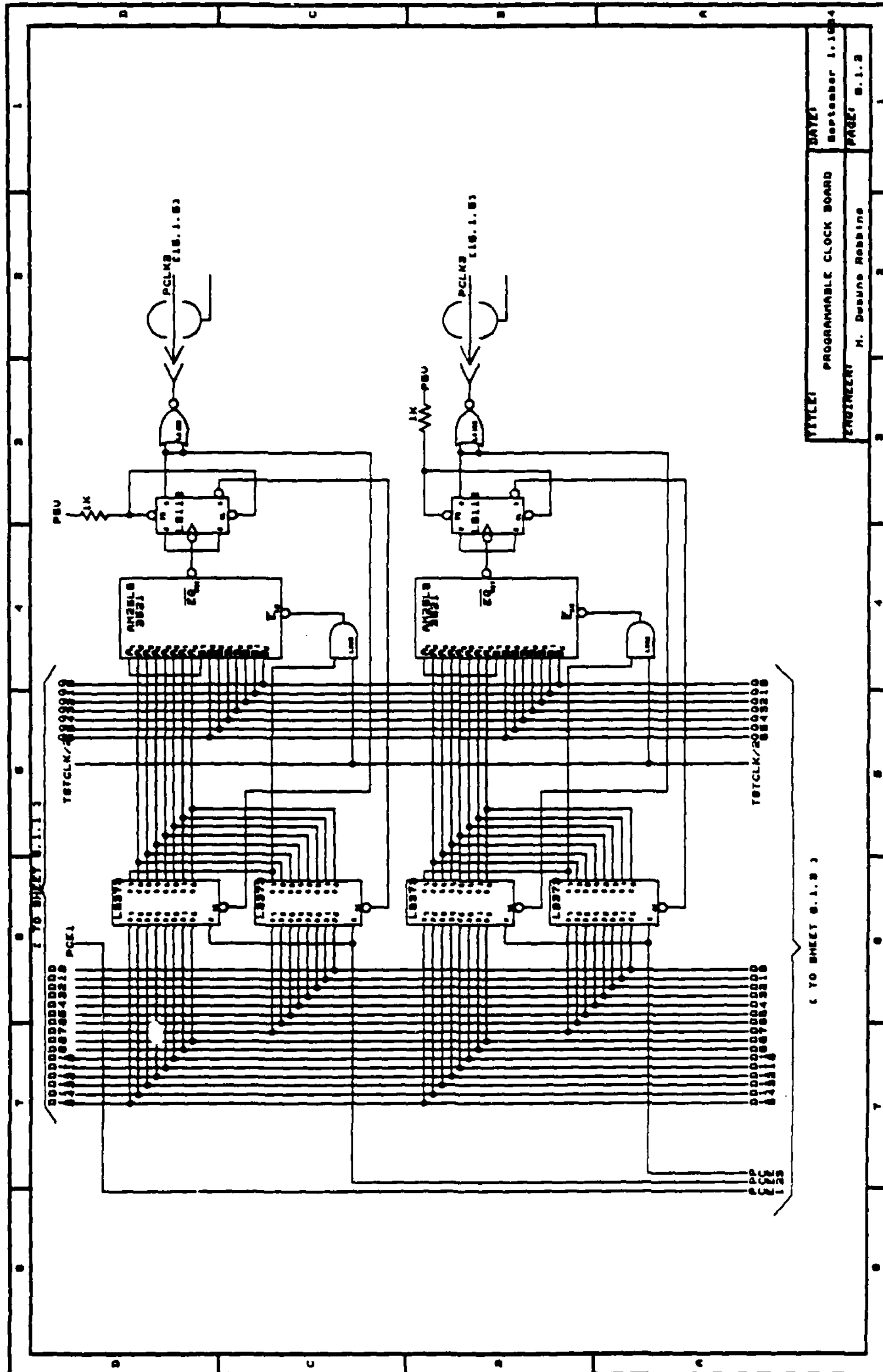

RC7
[15.1.3]
RC7
[15.1.3]
[5.1.2]
[5.1.4]
[5.1.4]
TITLE:
MEMORY BOARD 2
ENGINEER:
DATE:
PAGE:

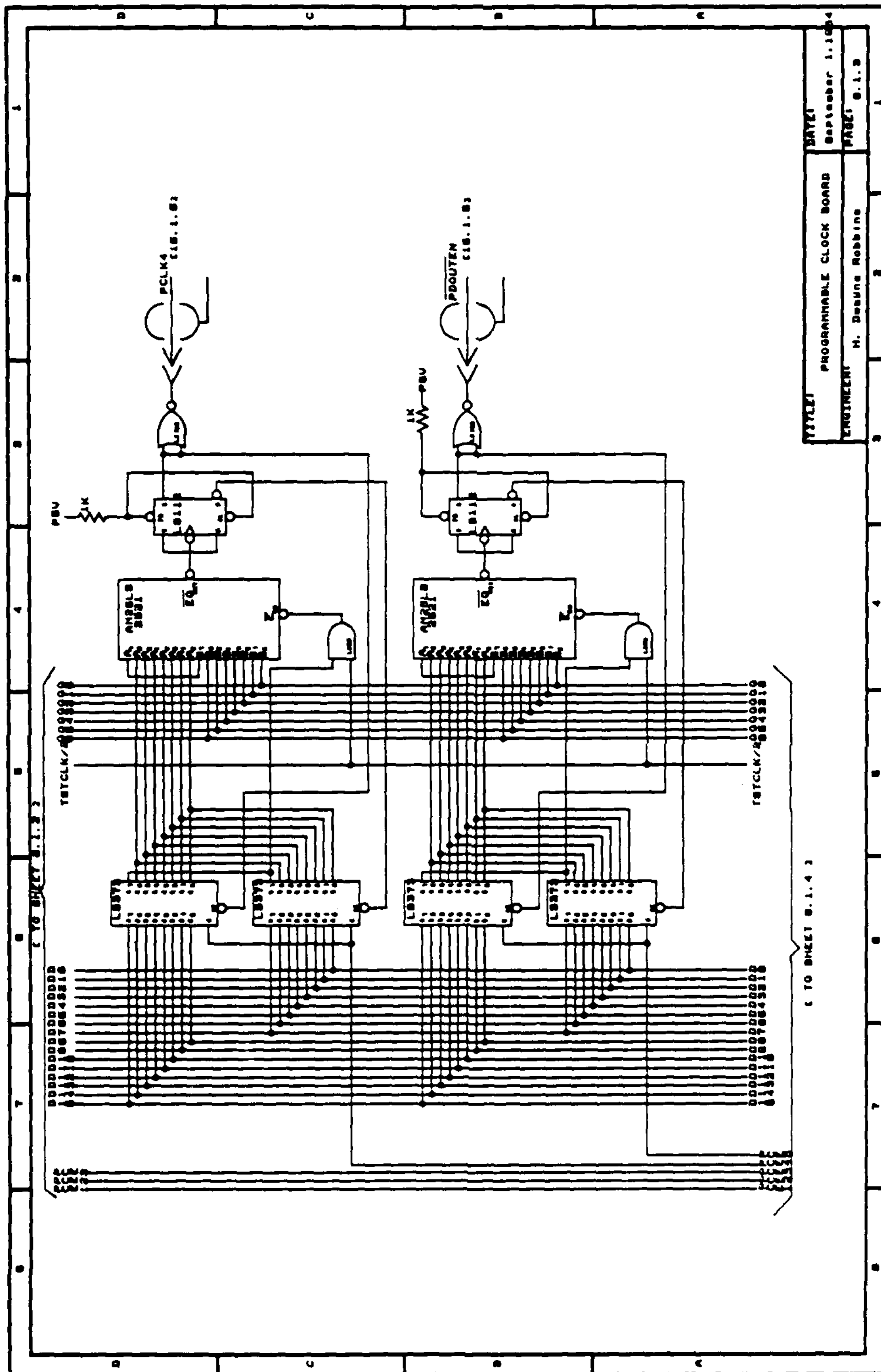

MEMORY BOARD 2
RC6
RC6
[15.1.4]
[15.1.4]
[5.1.3]

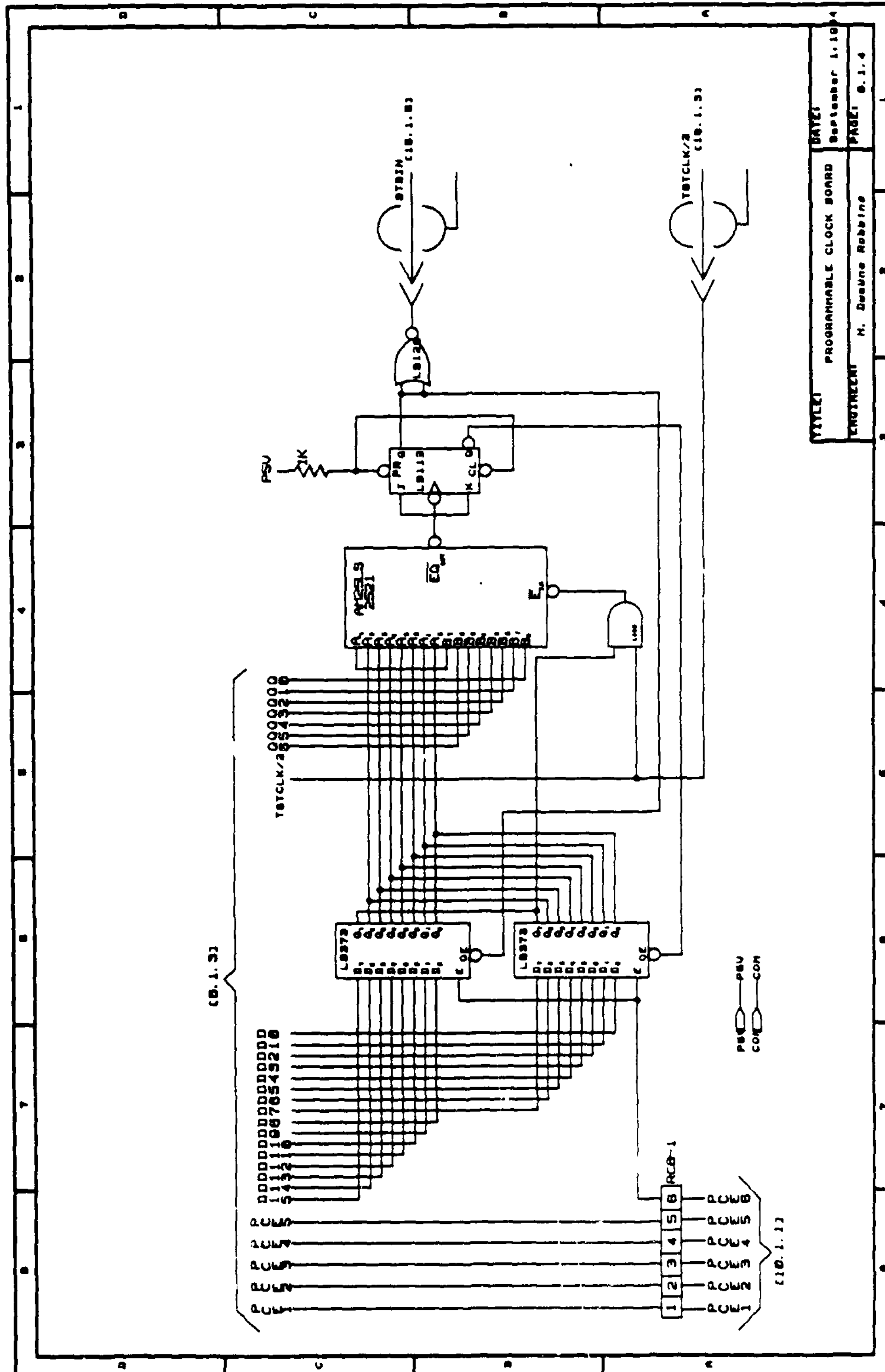

OSCILLATOR
CLK20
( TO TEST-PHASE CONTROL )
BP
( TO TEST-PHASE CONTROL )
PCLK1
( TO DUT INTERFACE )
TSTCLK/2
( TO TEST-PHASE CONTROL )
LS161
LS161
COMPARATOR
PCL1
( TO TEST-PHASE CONTROL )
*
EACH OF THE SIX PROGRAMMABLE CLOCKS
IS IDENTICAL TO THIS CIRCUIT
TITLE:
PROGRAMMABLE CLOCK BOARD
DATE:
ENGINEER:
M. Dwayne Robbins
PAGE:



TITLE:
PROGRAMMABLE CLOCK BOARD
ENGINEER:
M. Duane Robbins
DATE:
PAGE:
8.1.1
20MHZ OSCILLATOR
AM25LS
2521
CLK20
BP
PCLK1
P5V
1K
TSTCLK/2
PCE1
( TO SHEET 8.1.2 )


PROGRAMMABLE CLOCK BOARD
PCLK2
TSTCLK/2
AM25LS
2521
LS374
( TO SHEET 8.1.1 )
( TO SHEET 8.1.3 )


TITLE:
PROGRAMMABLE CLOCK BOARD
ENGINEER:
M. DeWayne Robbins
DATE:
September 1,1984
PAGE:
PCLK4
PDOUTEN
TSTCLK/
( TO SHEET 8.1.2 )
( TO SHEET 8.1.4 )


TITLE: PROGRAMMABLE CLOCK BOARD
DATE: September 1, 1984
ENGINEER: M. DuWayne Robbins
PAGE: 8.1.4
STBIN
TBTCLK/2
AM25LS 2521
LS112
LS373
LS373
RCB-1
PSV
COM

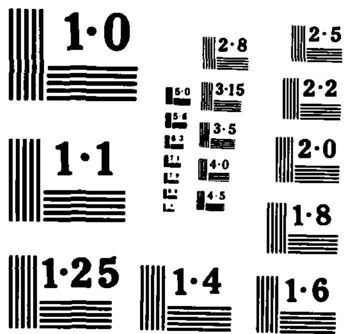
AD-A158 370 BULK CMOS VLSI TECHNOLOGY STUDIES PART 5 THE DESIGN AND IMPLEMENTATION OF..(U) MISSISSIPPI STATE UNIV MISSISSIPPI STATE DEPT OF ELECTRICAL E.. J D TROTTER ET AL. 17 JUN 85 2/2

UNCLASSIFIED F/G 14/2 NL

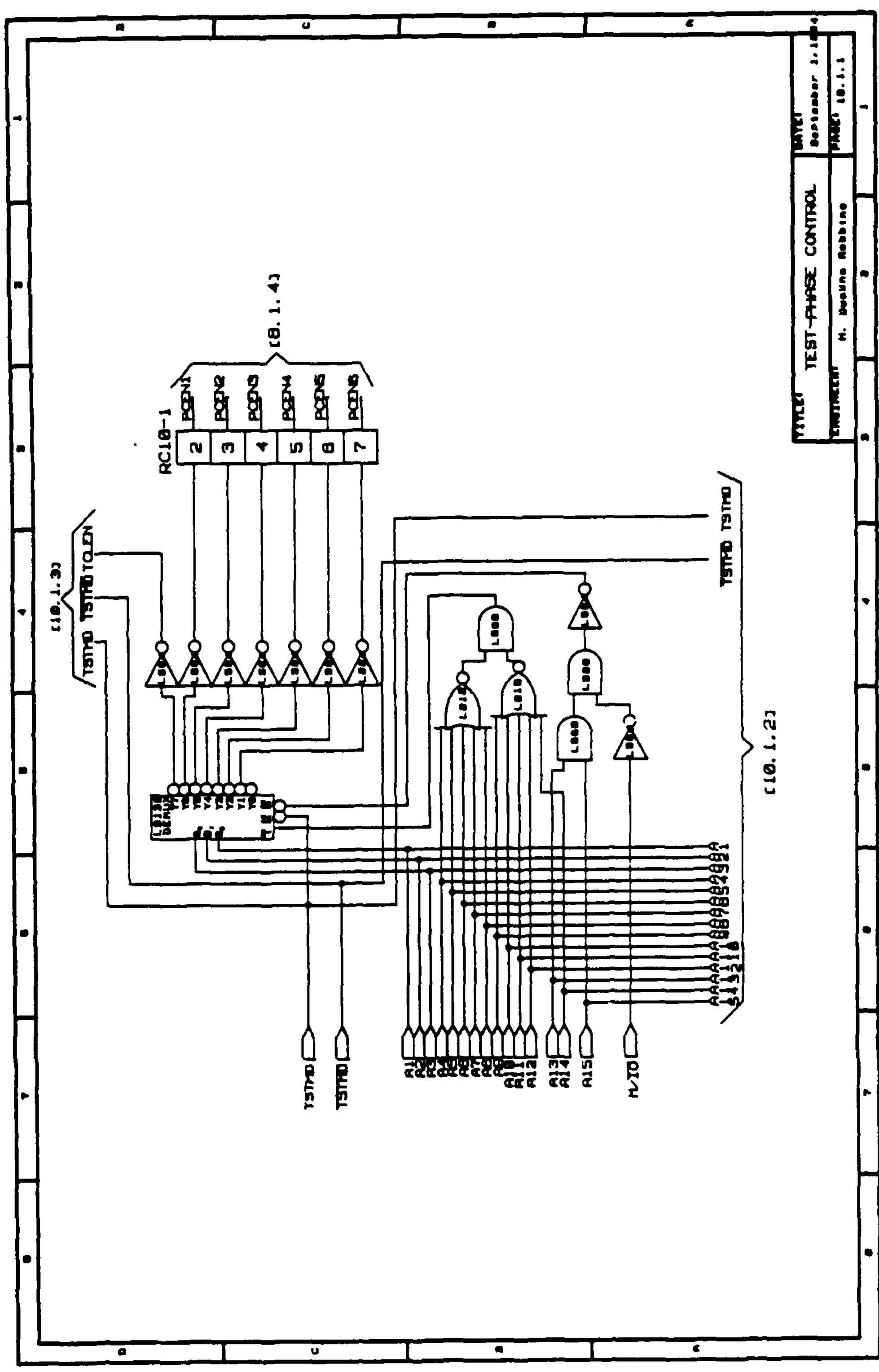
END

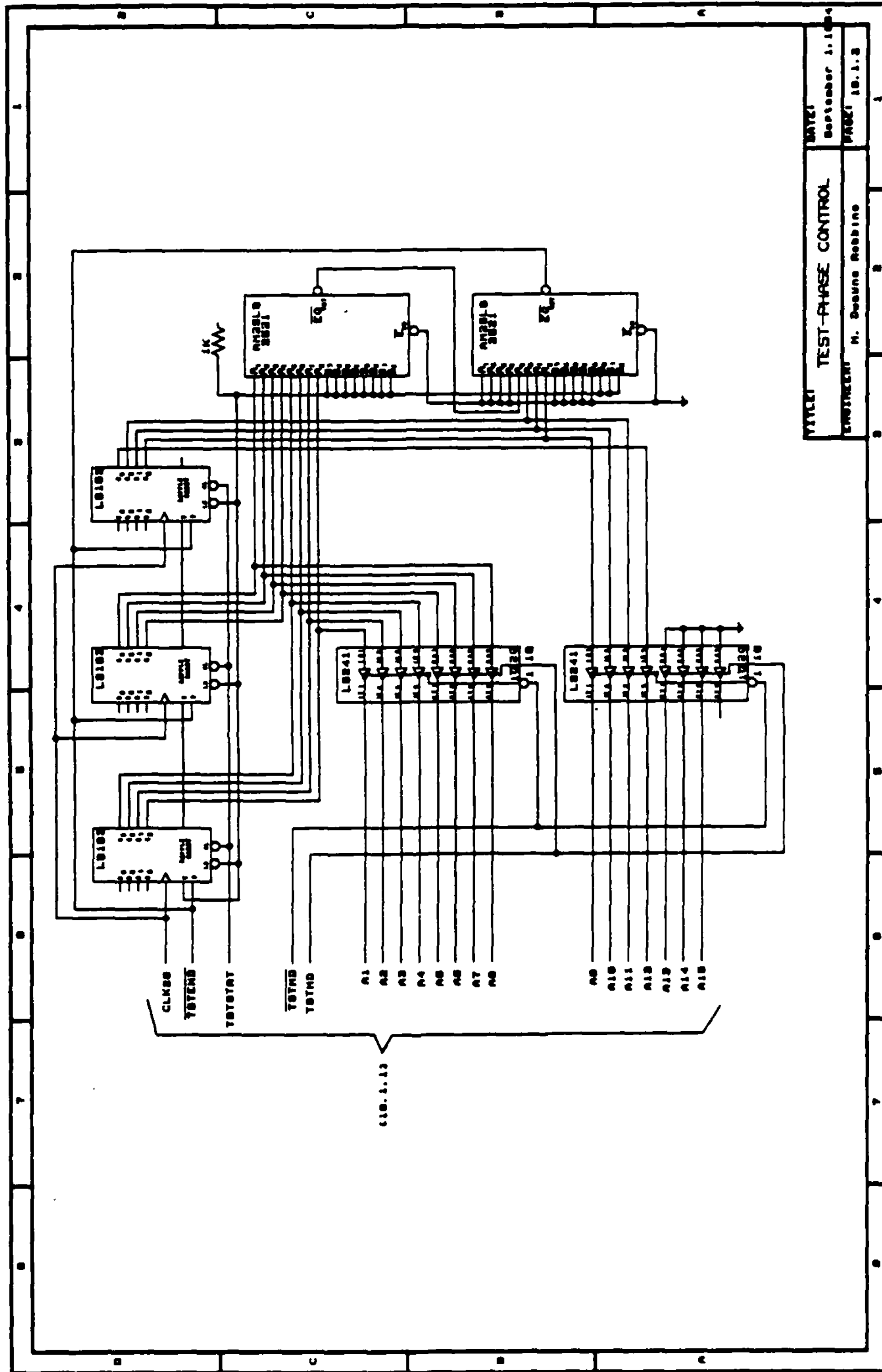
FILMED

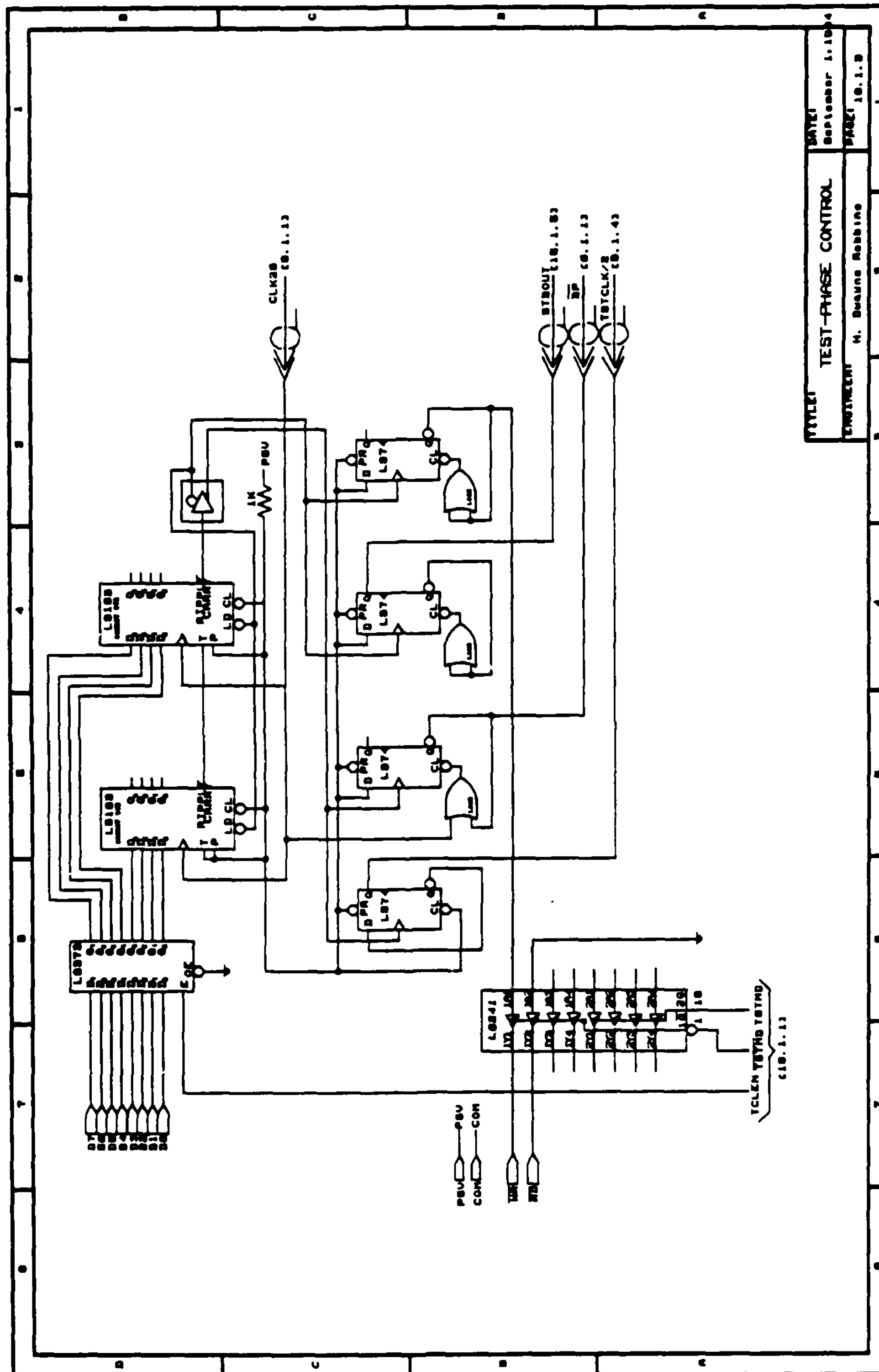
DTIC

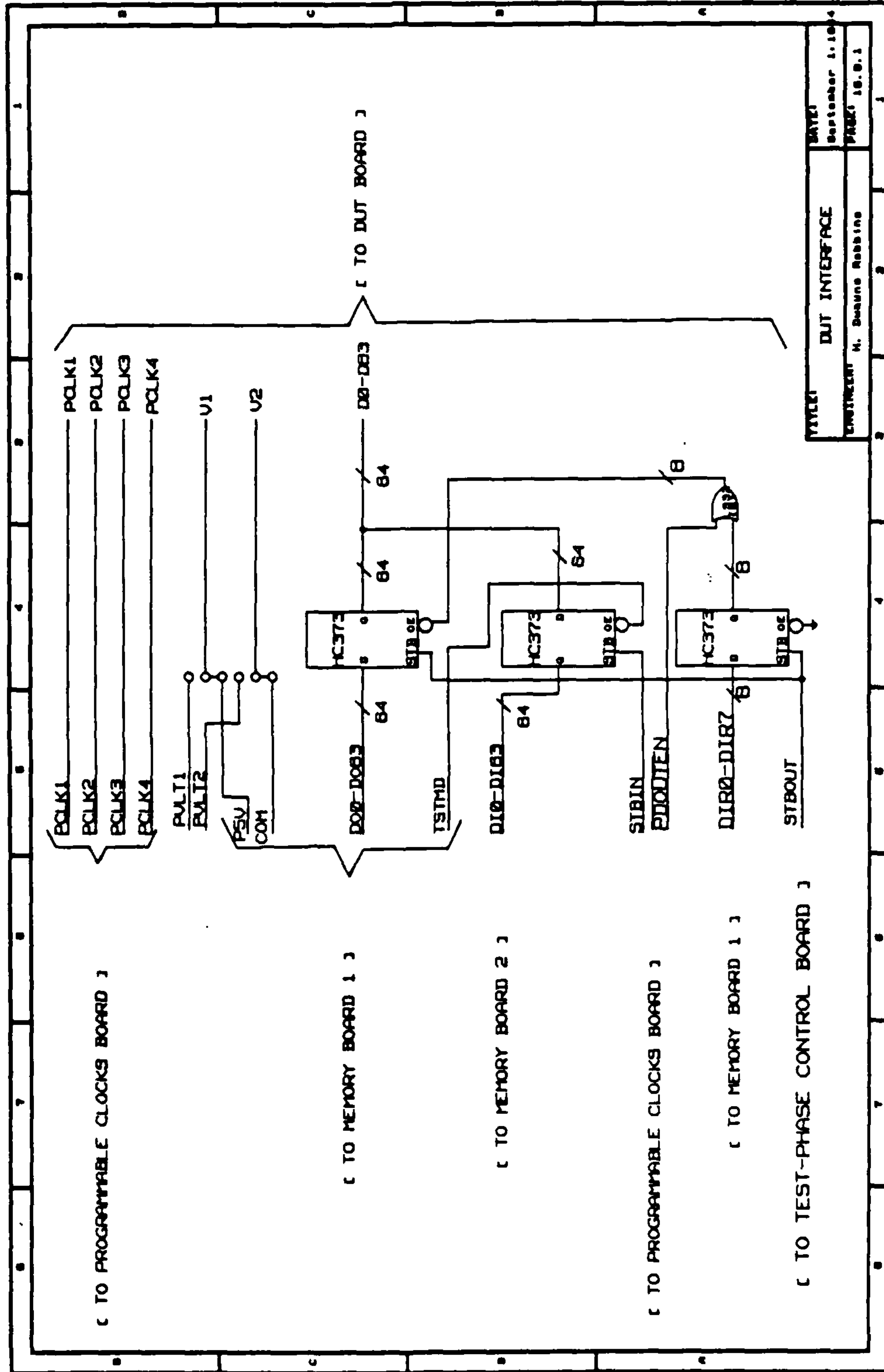


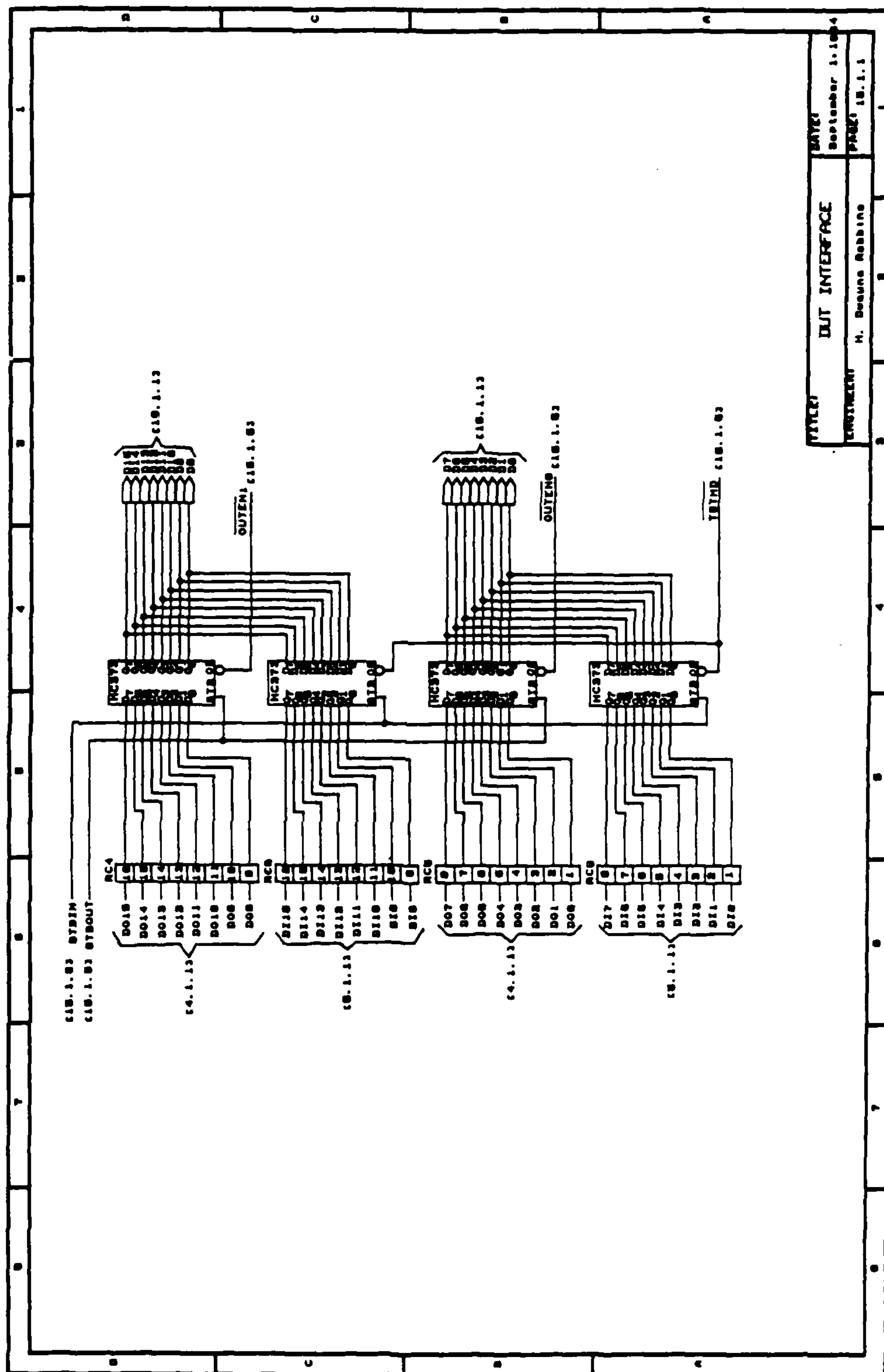

NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

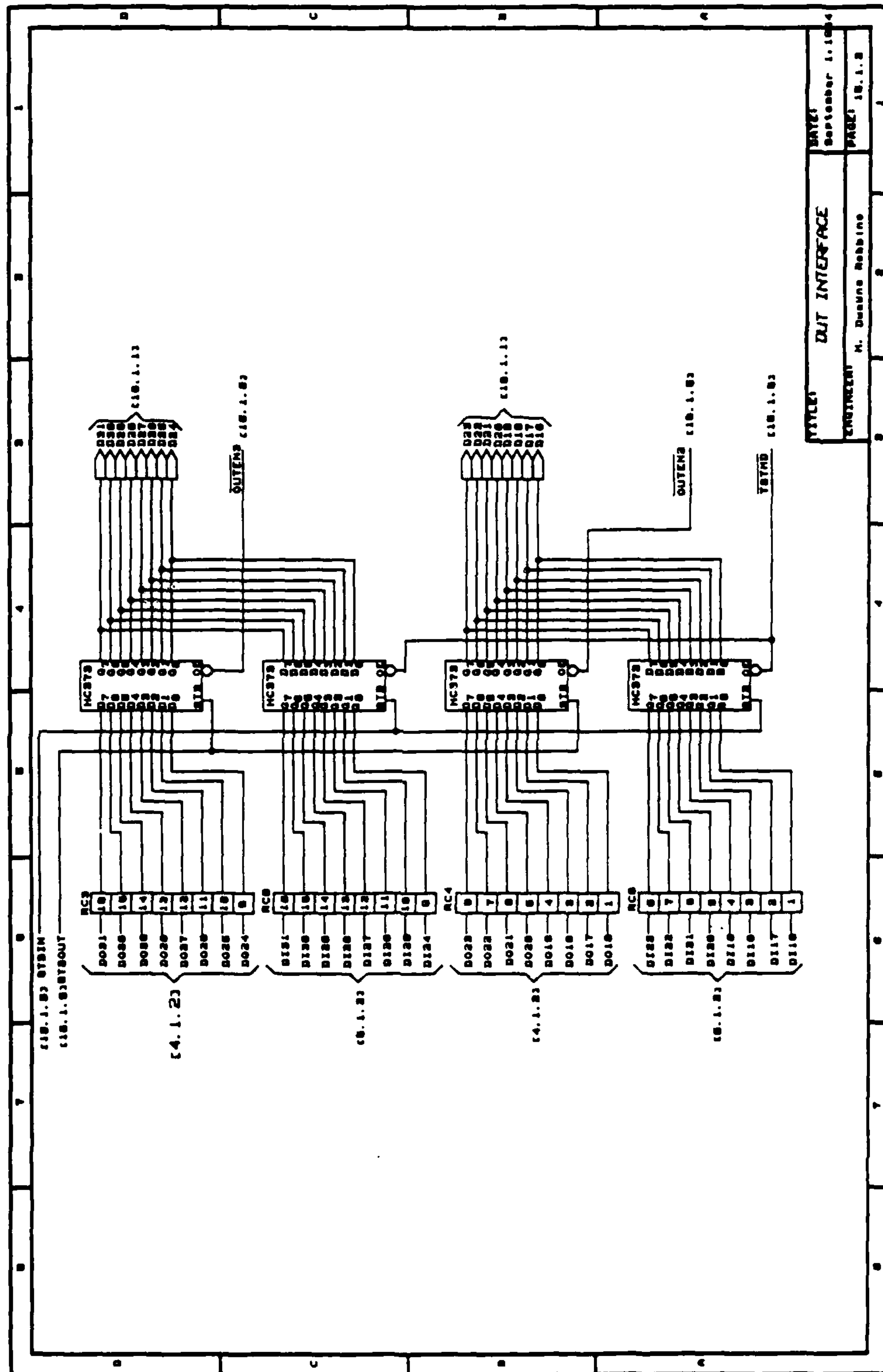

TEST-PHASE CONTROL
RC10-1
TSTMD
M/IO
A15

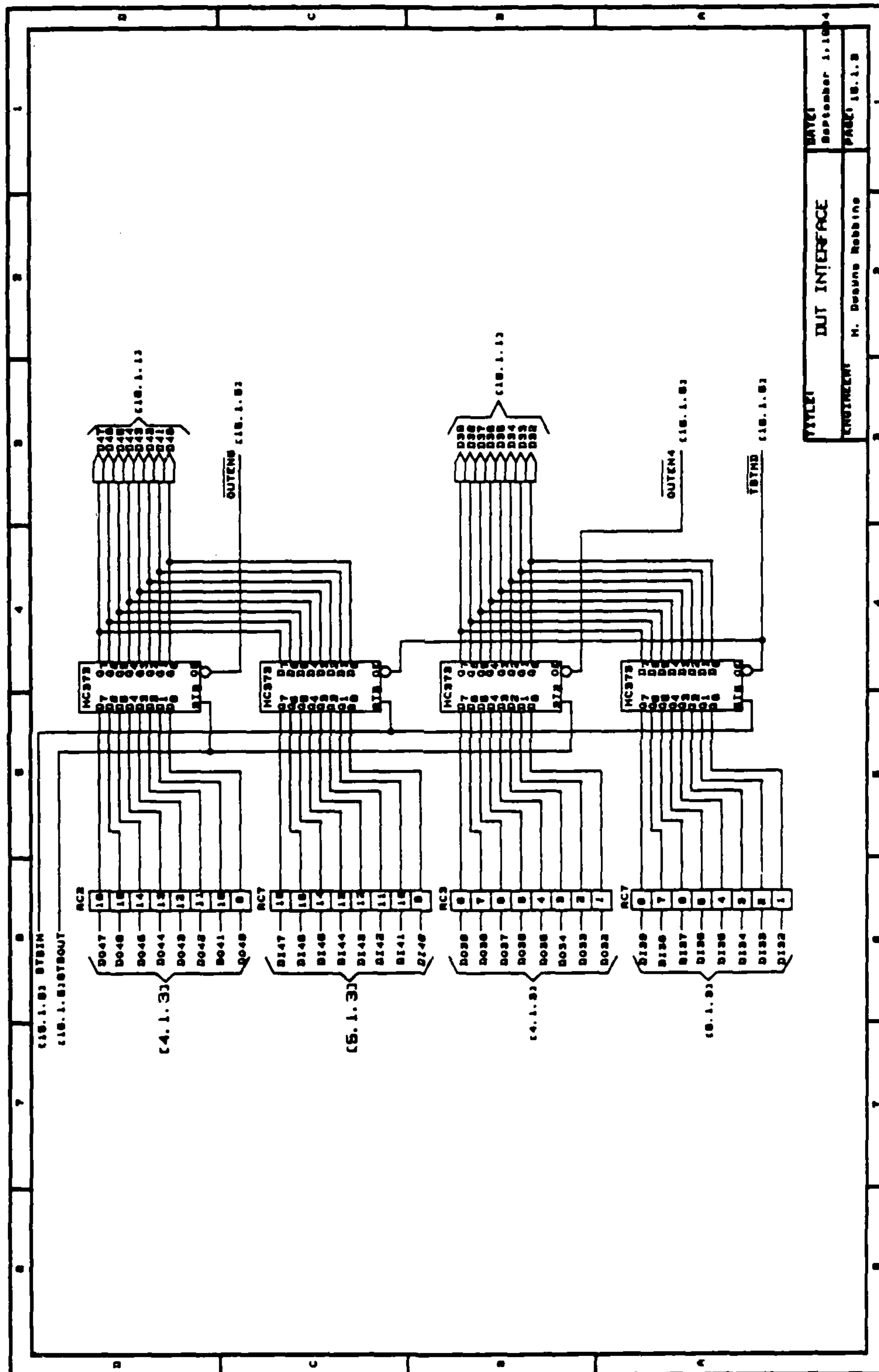

TITLE:
TEST-PHASE CONTROL
ENGINEER:
DATE:
PAGE:
AM25LS2521
LS241
A1
A2
A3
A4
A5
A6
A7
A8


TITLE:
TEST-PHASE CONTROL
ENGINEER:
DATE:
PAGE:
CLK28
STBOUT
TSTCLK/2
1K
PSV
COM
LS74
TCLEN

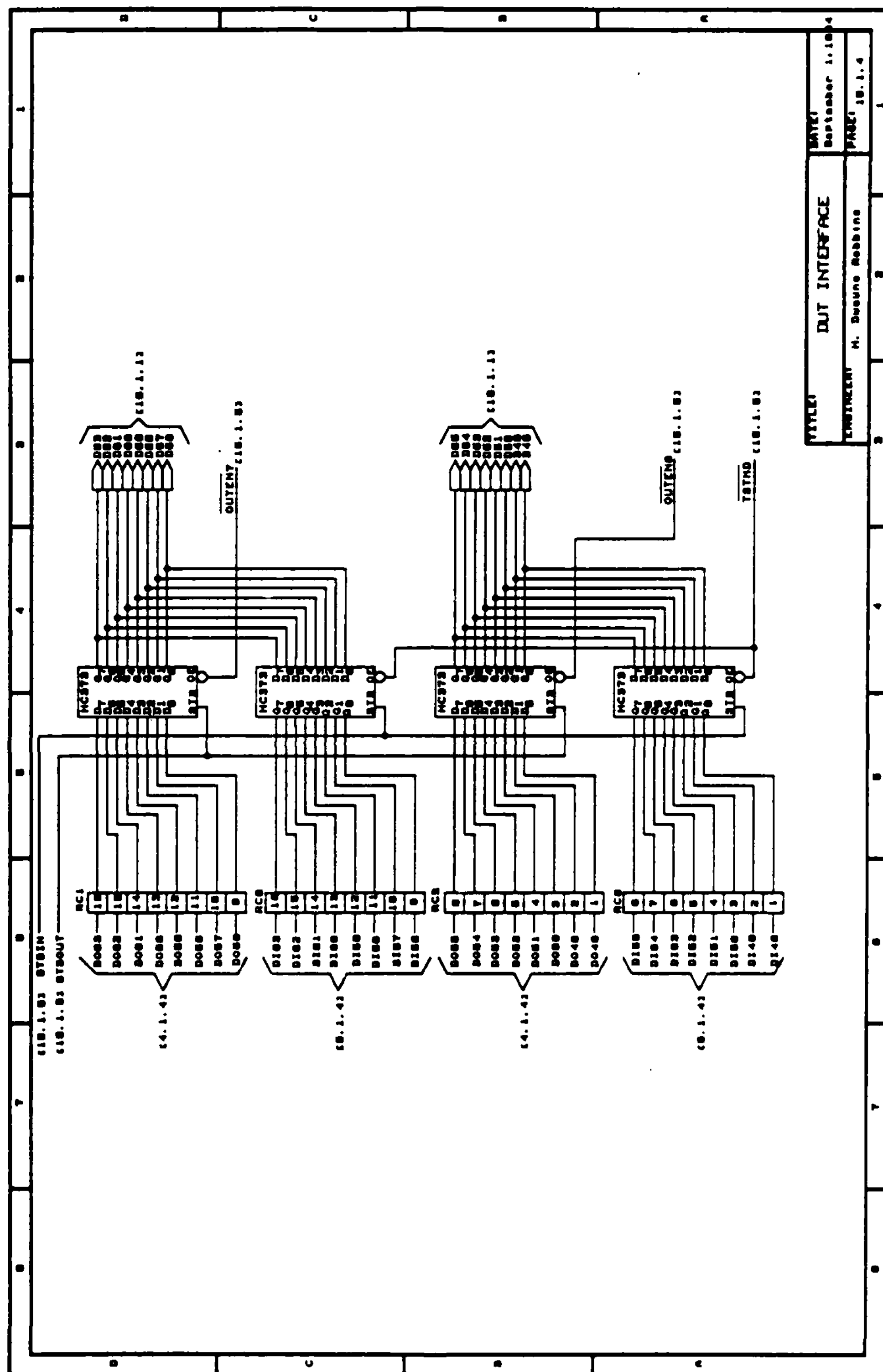

[ TO DUT BOARD ]
PCLK1
PCLK2
PCLK3
PCLK4
V1
V2
D0-D63
64
8
HC373
STB OE
D0-D63
TSTMD
DI0-DI63
STBIN
DIOOUTEN
DIR0-DIR7
STBOUT
PULT1
PULT2
PSU
COM
[ TO PROGRAMMABLE CLOCKS BOARD ]
[ TO MEMORY BOARD 1 ]
[ TO MEMORY BOARD 2 ]
[ TO PROGRAMMABLE CLOCKS BOARD ]
[ TO MEMORY BOARD 1 ]
[ TO TEST-PHASE CONTROL BOARD ]
TITLE:
DUT INTERFACE
ENGINEER: M. Duane Robbins
DATE:
September

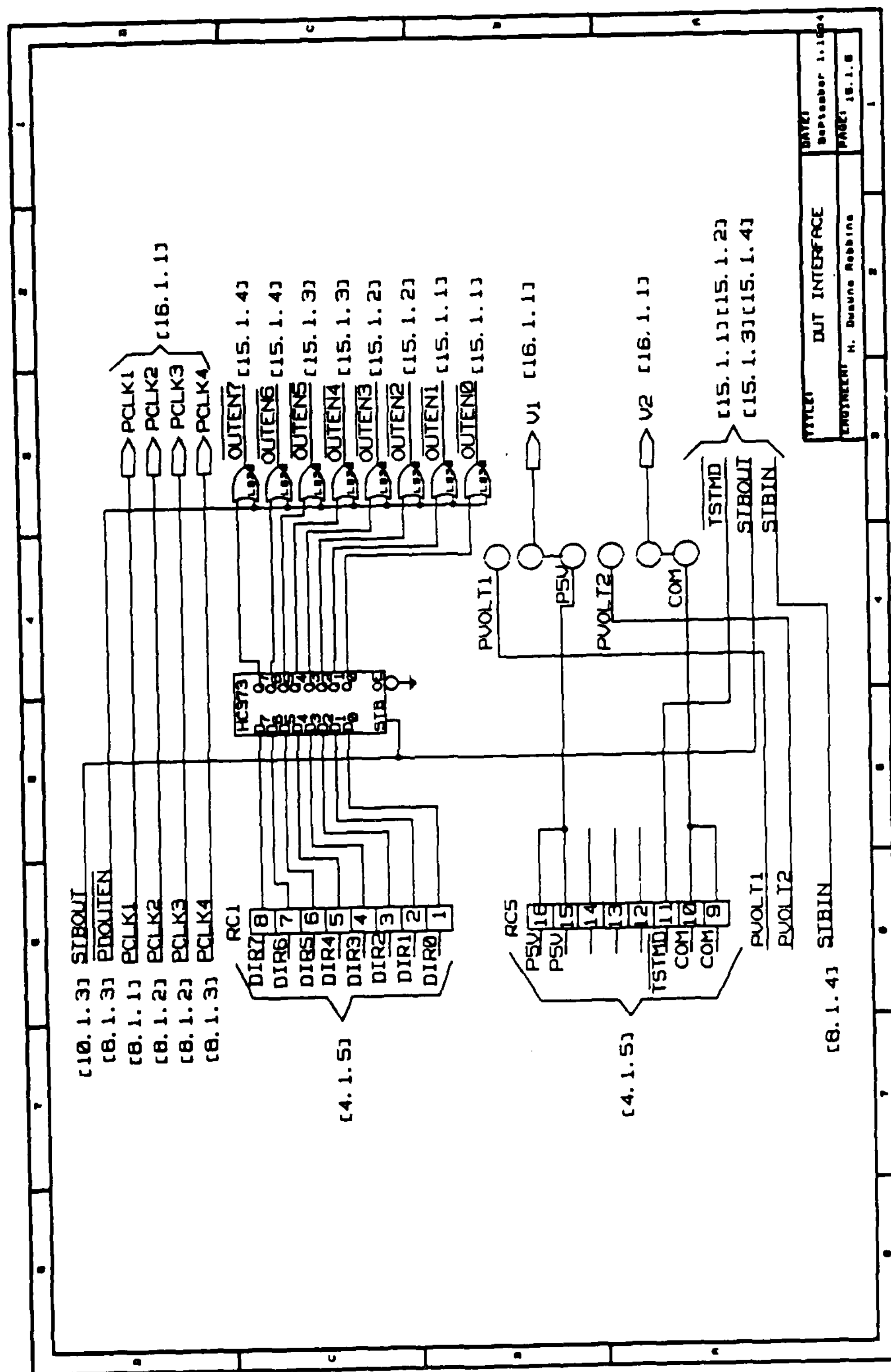

TITLE:
DUT INTERFACE
ENGINEER:
H. Dwaune Robbins
DATE:
PAGE:
15.1.1
STBIN
STBOUT
OUTEN1
OUTEN0
TSTMD


DUT INTERFACE
TITLE:
ENGINEER:
DATE:
PAGE:

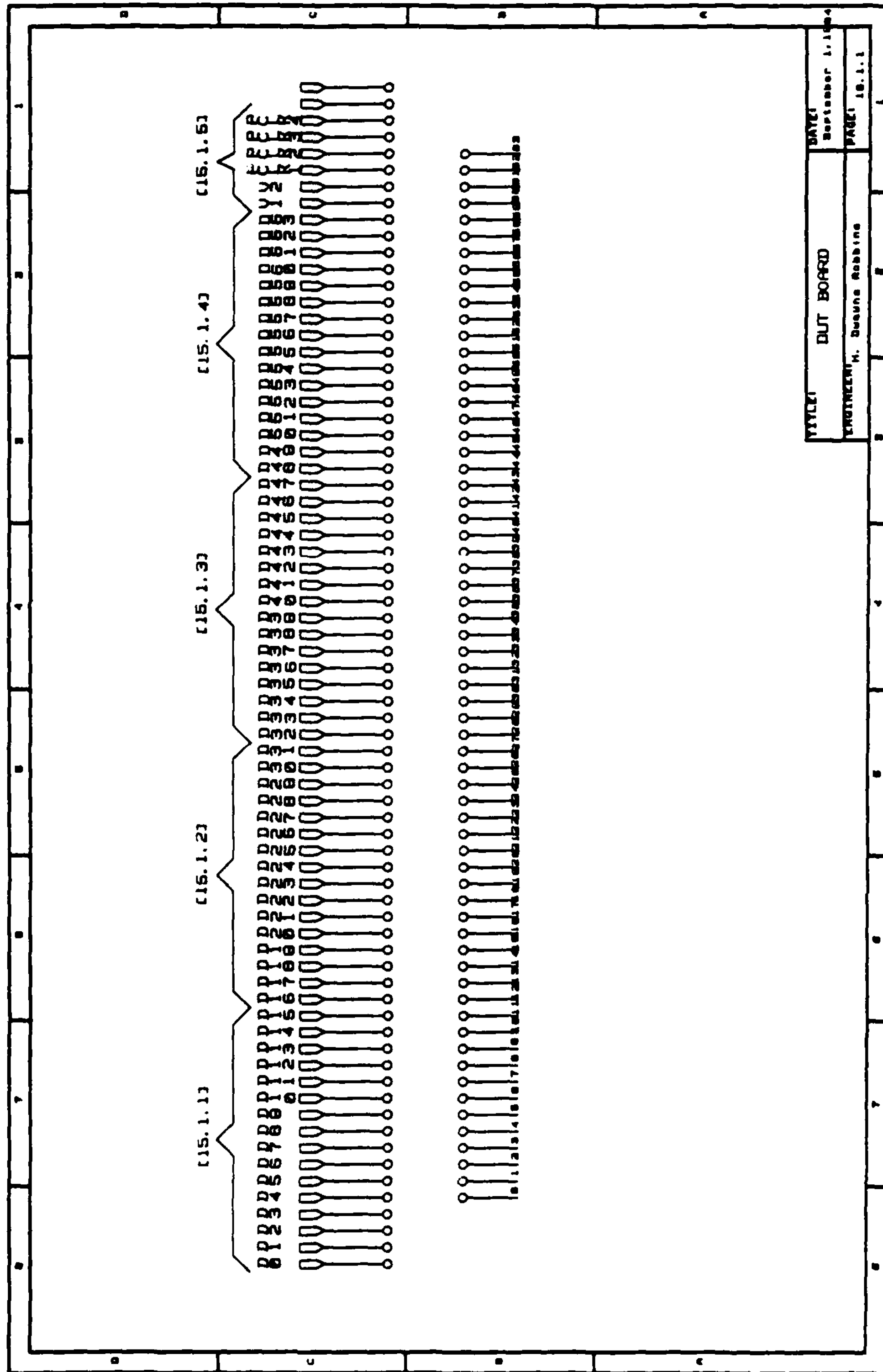


DUT INTERFACE
M. Duane Robbins


DUT INTERFACE



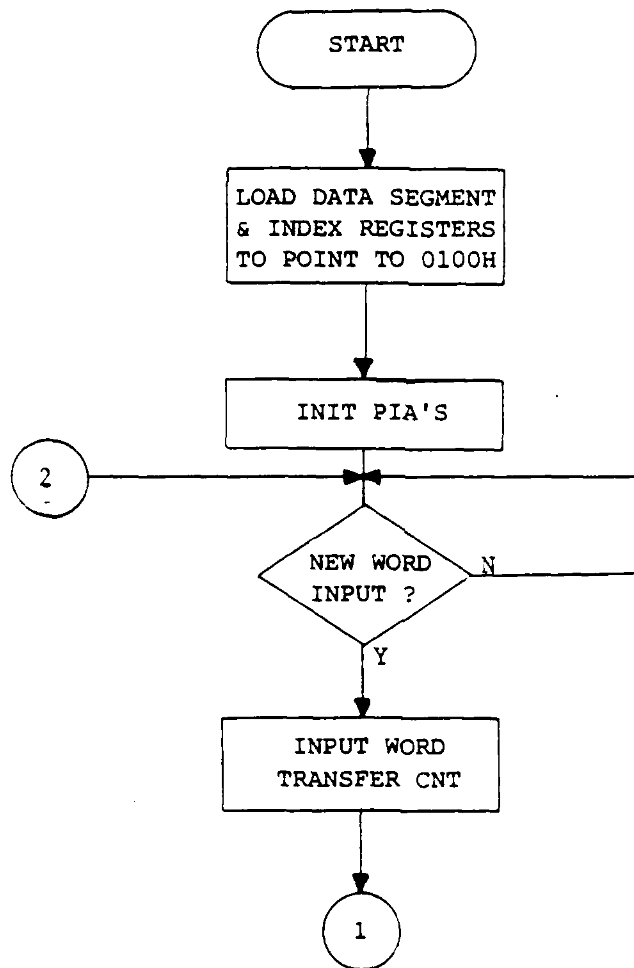
[10.1.3] STBOUT
[8.1.3] PDOUTEN
[8.1.1] PCLK1
[8.1.2] PCLK2
[8.1.2] PCLK3
[8.1.3] PCLK4
PCLK1
PCLK2
PCLK3
PCLK4
[16.1.1]
RC1
DIR7
DIR6
DIR5
DIR4
DIR3
DIR2
DIR1
DIR0
[4.1.5]
HC573
STB
OE
OUTEN7 [15.1.4]
OUTEN6 [15.1.4]
OUTEN5 [15.1.3]
OUTEN4 [15.1.3]
OUTEN3 [15.1.2]
OUTEN2 [15.1.2]
OUTEN1 [15.1.1]
OUTEN0 [15.1.1]
PVOLT1
V1 [16.1.1]
P5V
PVOLT2
V2 [16.1.1]
COM
RC5
P5V
P5V
TSTMD
COM
COM
[4.1.5]
TSTMD
STBOUT
STBIN
[15.1.1][15.1.2]
[15.1.3][15.1.4]
PVOLT1
PVOLT2
[8.1.4] STBIN
TITLE: DUT INTERFACE
ENGINEER: H. Duane Robbins
DATE:
PAGE:


[15.1.1]
[15.1.2]
[15.1.3]
[15.1.4]
[15.1.5]
TITLE:
DUT BOARD
ENGINEER:
DATE:
PAGE:
15.1.1

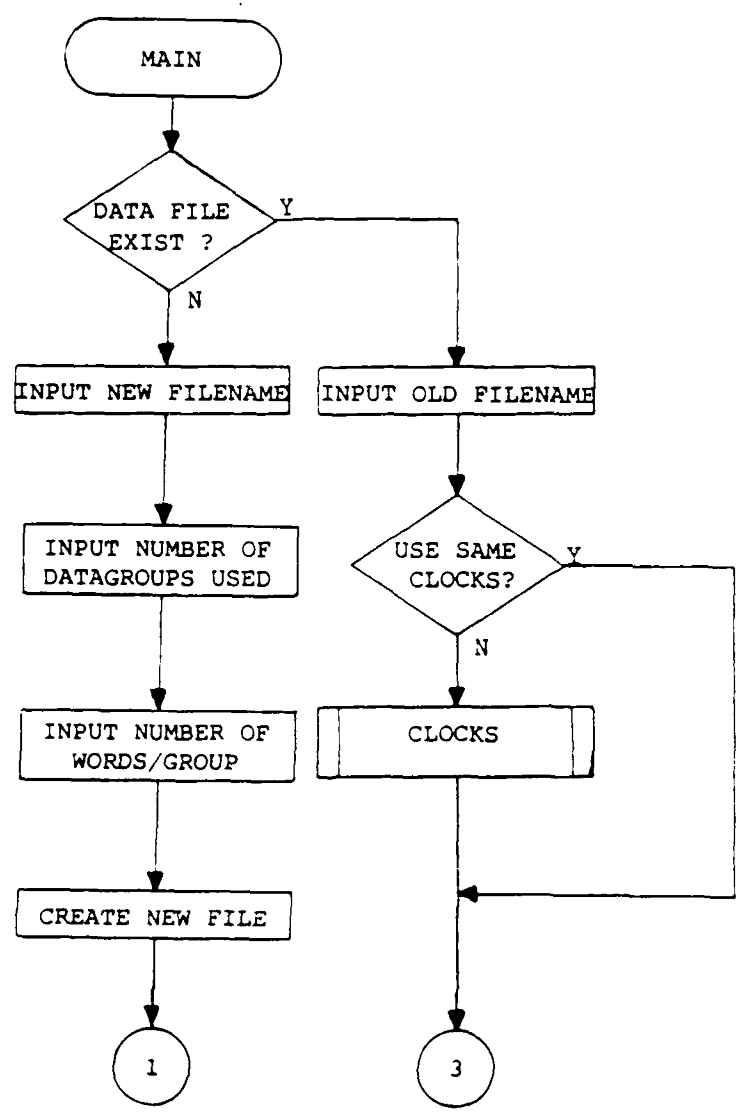
# APPENDIX C

## ASSEMBLY LANGUAGE PROGRAMS

UTILITY PROGRAM FLOWCHART



PASCAL PROGRAM FLOWCHART



# APPENDIX D

# PASCAL PROGRAM

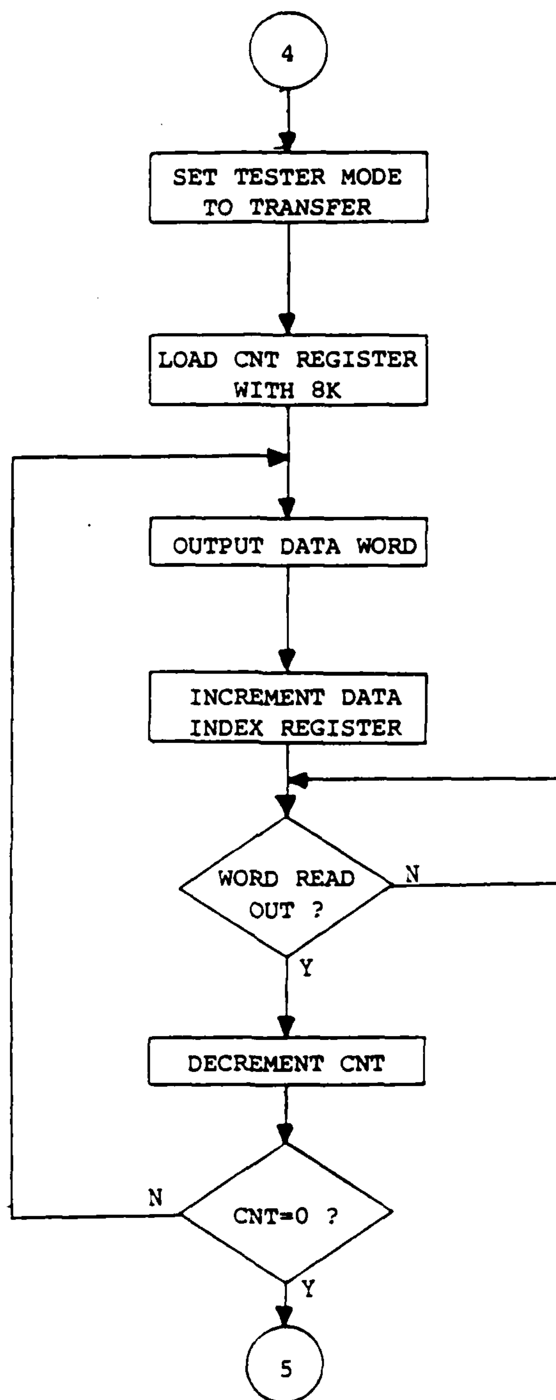
```
                XCHG     DX,BX
LPS:            IN       AX,[DX]
                AND      AX,2020H
                JZ       LPS
                LOOP     LPT
;
; CLEAR OUTPUT PORT
;
                XCHG     DX,BX
                OUT      [DX],AX
;
; JUMP TO START OF PROGRAM FOR NEXT TEST
;
                JMP      START
```

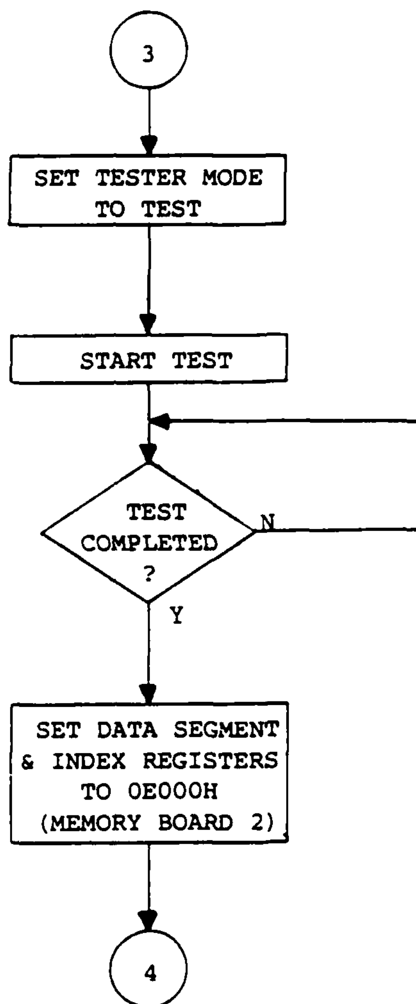
```
;
; START TEST
;
                MOV     AL,06H
                OUT     [DX],AL
;
; CHECK FOR END OF TEST
;
LP:             IN      AL,[DX]
                AND     AL,01H
                JNZ     LP
;
; SET DATA SEGMENT AND INDEX REGISTERS TO
; POINT TO DATA BUFFER ON MEMORY BOARD 2
;
                MOV     AX,0E000H
                MOV     DS,AX
                MOV     DI,00H
;
; SET TESTER TO TRANSFER MODE
;
                MOV     AL,01H
                OUT     [DX],AL
;
; INITIALIZE PIA'S
;
                MOV     DX,0FFFEH
                MOV     AL,0DH
                OUT     [DX],AL
;
; LOAD CX WITH WORD TRANSFER COUNT
;
                MOV     CX,2000H
;
; LOAD DX AND BX WITH OUTPUT PORT ADDRESSES
;
                MOV     DX,0FFFCH
                MOV     BX,0FFFAH
;
; OUTPUT DATA WORDS UNTIL COUNT = 0
;
LPT:            XCHG    DX,BX
                MOV     AX,DS:[DI]
                INC     DI
                INC     DI
                OUT     [DX],AX
;
; CHECK STATUS FOR WORD READ OUT
;
```

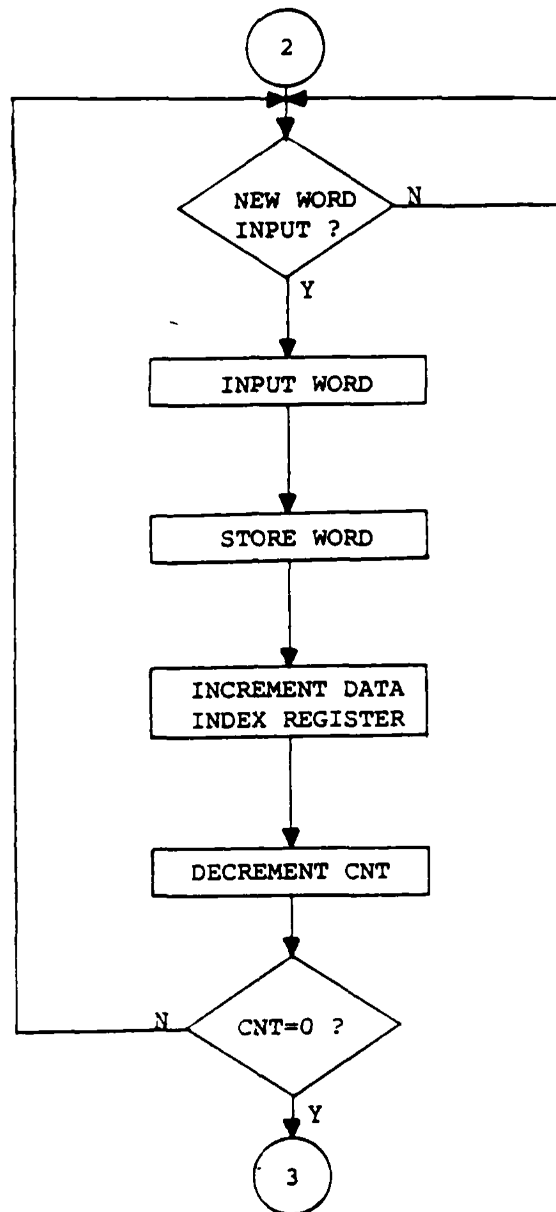
```
; INPUT DATA WORD AND STORE IN BUFFER MEMORY
;
                XCHG    DX,BX
                IN      AX,[DX]
                MOV     DS:[DI],AX
                INC     DI
                INC     DI
                LOOP    LPM
;
; LOAD COUNT REGISTER (CX) WITH 7
;
                MOV     CX,07H
;
; RE-INITIALIZE DX AND BX TO INPUT PORT ADDRESSES
;
                MOV     DX,0FFF8H
                MOV     BX,0FFFCH
;
; SET INDEX REGISTER TO FIRST ADDRESS OF
; PROGRAMMABLE CLOCKS' REGISTERS
;
                MOV     SI,0A000H
;
; INPUT 7 CLOCK WORDS
;
LPB:            XCHG    DX,BX
;
; CHECK STATUS FOR NEW WORD INPUT
;
LPSTAT2:        IN      AX,[DX]
                AND     AX,2020H
                JZ      LPSTAT2
;
; INPUT AND STORE DATA WORD
;
                XCHG    DX,BX
                IN      AX,[DX]
                XCHG    DX,SI
                OUT     [DX],AX
                INC     DX
                INC     DX
                XCHG    DX,SI
                LOOP    LPB
;
; SET TESTER MODE TO TEST
;
                MOV     AL,02H
                MOV     DX,00H
                OUT     [DX],AL
```

TESTER CONTROL ASSEMBLY LANGUAGE PROGRAM

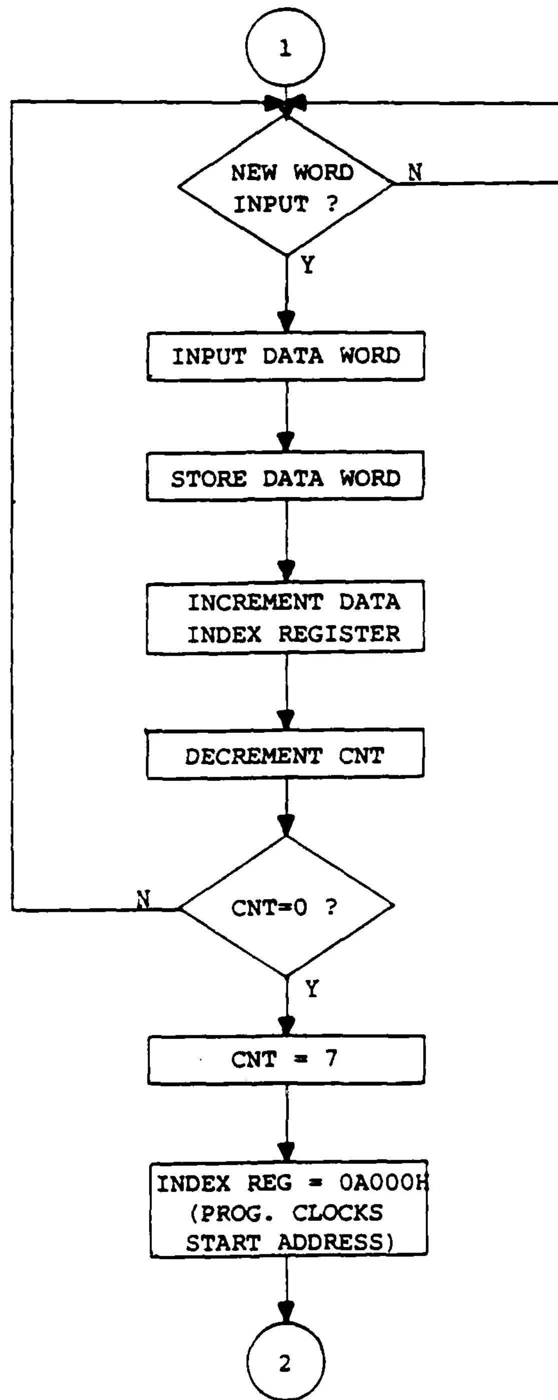
```
;
; INITIALIZE PIA'S
;
START:          MOV     DX,0FFFEH
                MOV     AX,0BCB4H
                OUT     [DX],AX
                MOV     AL,0CH
                OUT     [DX],AL
;
; SET TESTER MODE TO TRANSFER
;
                MOV     AL,01H
                MOV     DX,00H
                OUT     [DX],AL
;
; SET DATA SEGMENT AND INDEX REGISTERS
; TO BUFFER ADDRESS ON MEMORY BOARD 1
;
                MOV     AX,0F000H
                MOV     DS,AX
                MOV     DI,00H
;
; LOAD WORD TRANSFER COUNT IN CX
;
                MOV     CX,2800H
;
; LOAD DX AND BX TO POINT TO INPUT PORT
; DATA AND CONTROL ADDRESSES
;
                MOV     DX,0FFF8H
                MOV     BX,0FFFCH
;
; CLEAR INPUT PORT
;
                IN      AX,[DX]
;
; INPUT WORDS UNTIL COUNT = 0
;
LPM:            XCHG    DX,BX
;
; CHECK STATUS FOR NEW WORD INPUT
;
LPSTAT:         IN      AX,[DX]
                AND     AX,2020H
                JZ      LPSTAT
;
```


4
SET TESTER MODE
TO TRANSFER
LOAD CNT REGISTER
WITH 8K
OUTPUT DATA WORD
INCREMENT DATA
INDEX REGISTER
WORD READ
OUT ?
N
Y
DECREMENT CNT
CNT=0 ?
N
Y
5

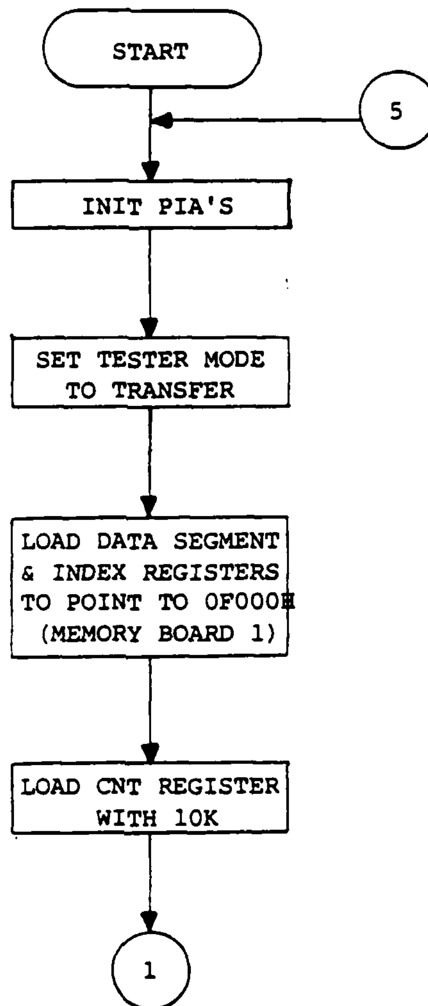

3
SET TESTER MODE TO TEST
START TEST
TEST COMPLETED ?
N
Y
SET DATA SEGMENT & INDEX REGISTERS TO 0E000H (MEMORY BOARD 2)
4



2
NEW WORD INPUT ?
N
Y
INPUT WORD
STORE WORD
INCREMENT DATA INDEX REGISTER
DECREMENT CNT
CNT=0 ?
N
Y
3


1
NEW WORD INPUT ?
N
Y
INPUT DATA WORD
STORE DATA WORD
INCREMENT DATA INDEX REGISTER
DECREMENT CNT
CNT=0 ?
N
Y
CNT = 7
INDEX REG = 0A000H (PROG. CLOCKS START ADDRESS)
2

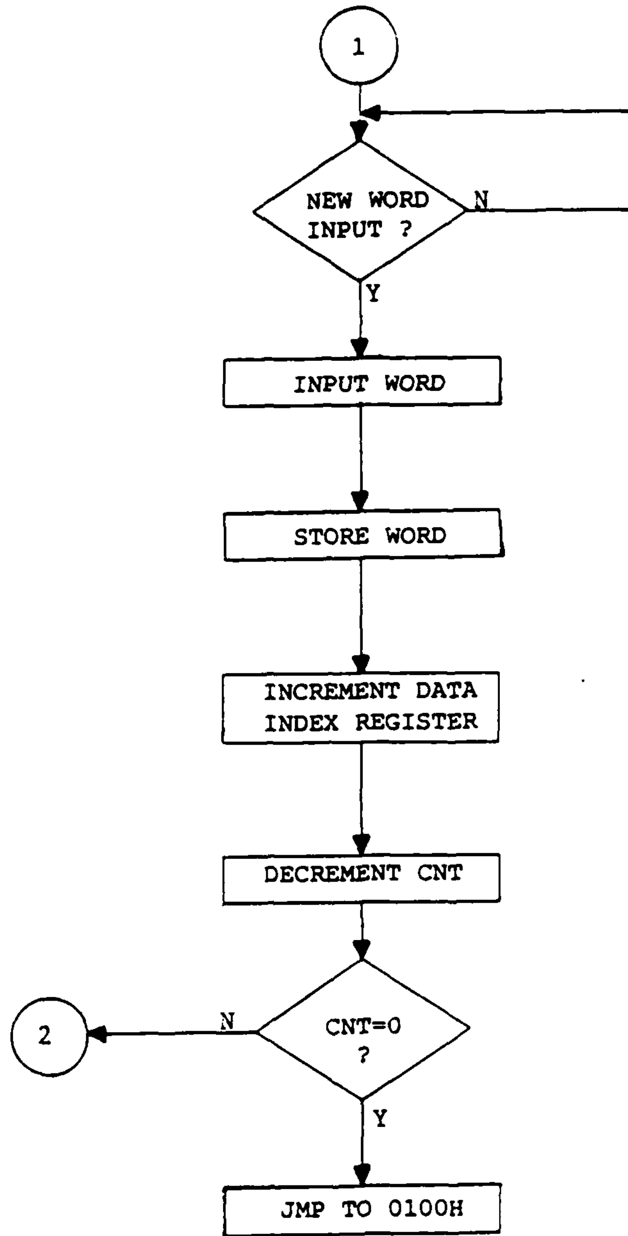
CONTROL PROGRAM FLOWCHART

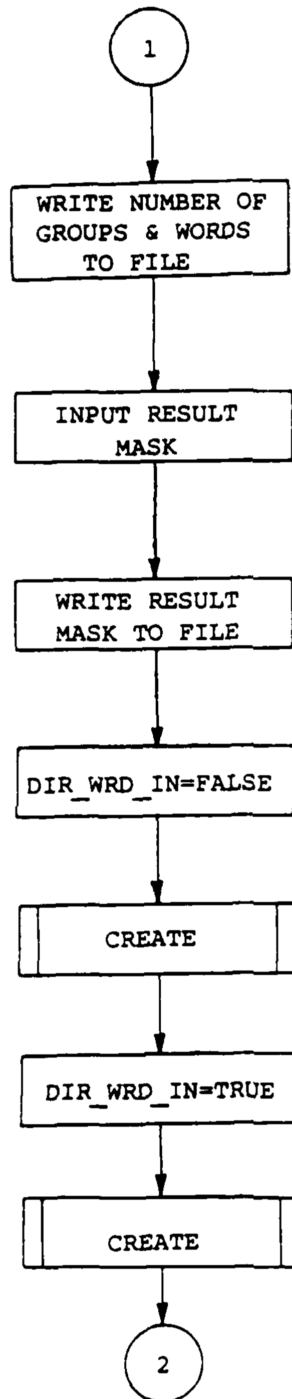


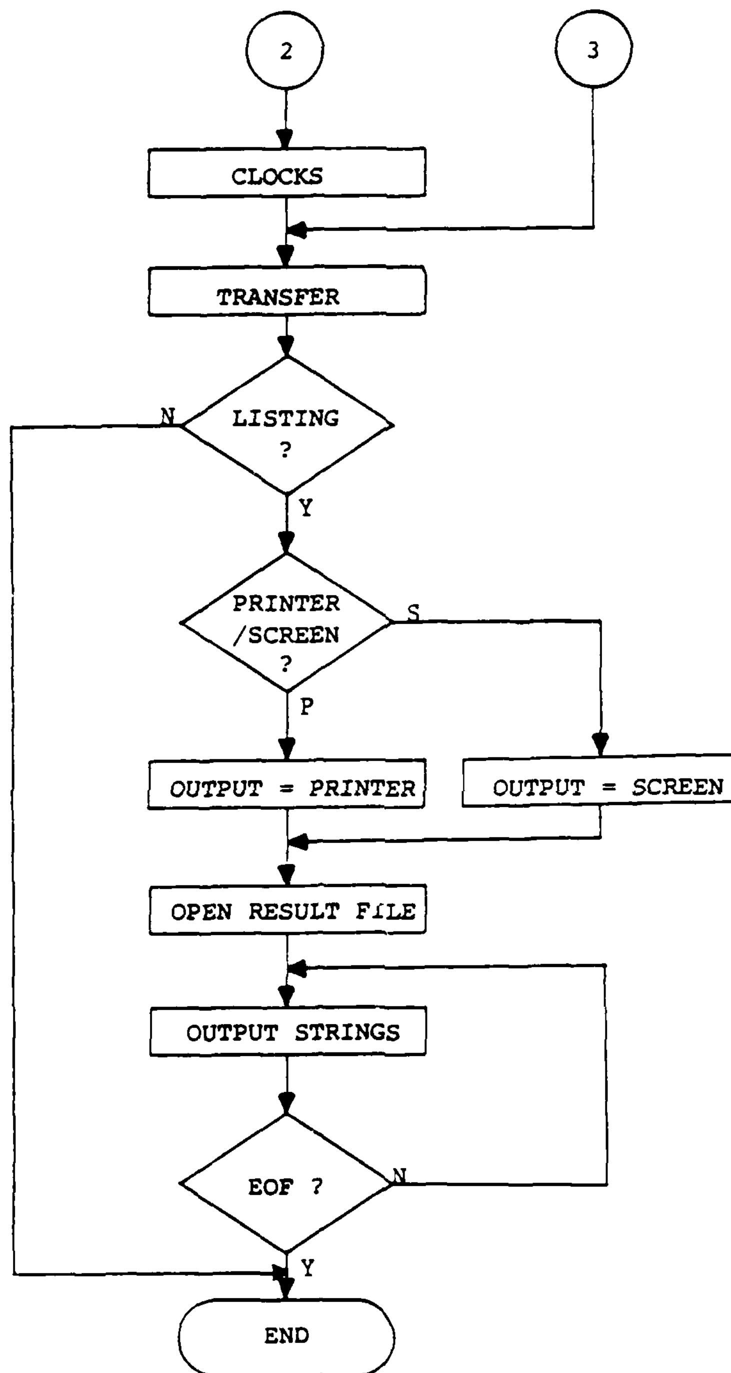
```
                AND     AL,20H
                JZ      LPSTAT2
;
; INPUT DATA WORD AND STORE IN PROGRAM BUFFER
;
                XCHG    DX,BX
                IN      AX,[DX]
                MOV     DS:[DI],AX
                INC     DI
                INC     DI
                LOOP    LPM
;
; WHEN PROGRAM IS TRANSFERRED, JUMP TO START
;
                JMP     00:100H
```

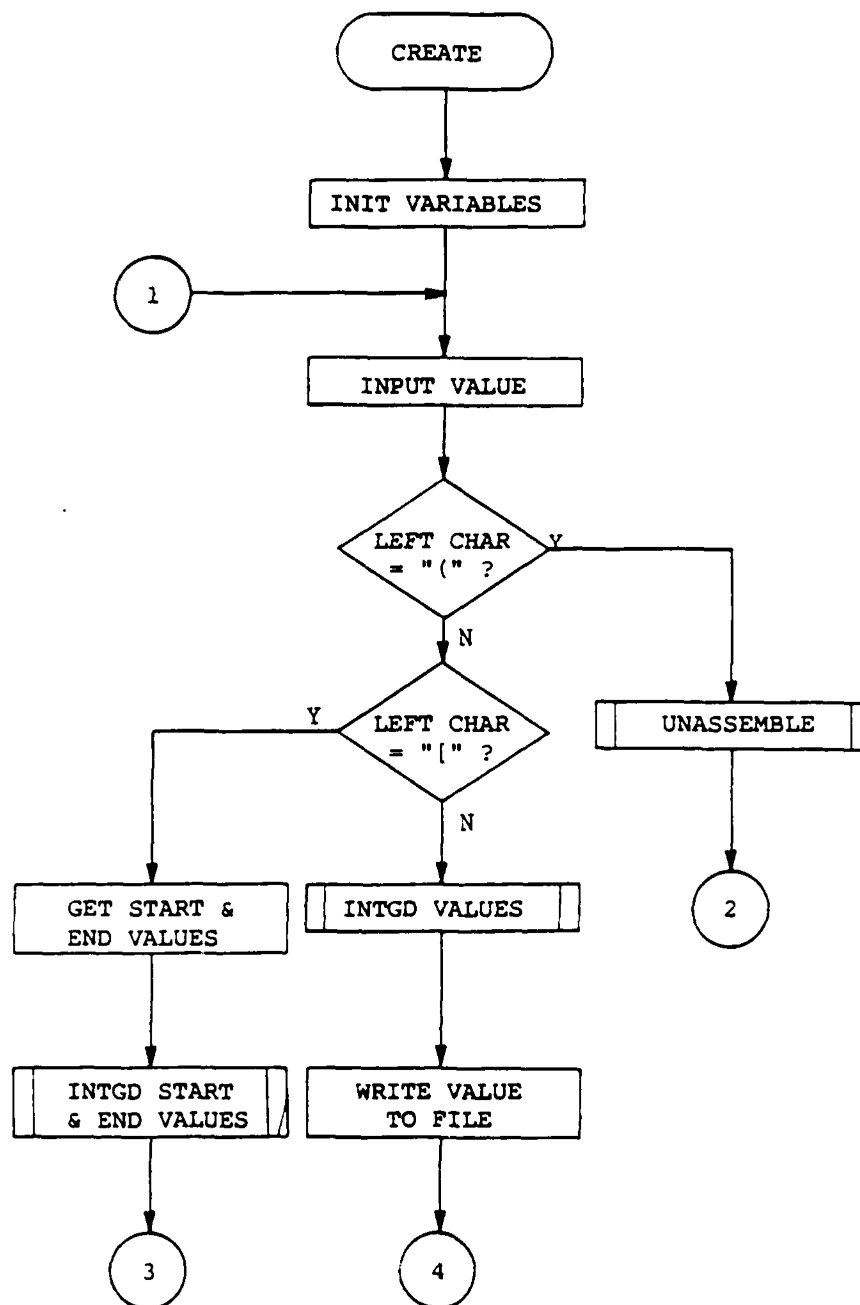
UTILITY ASSEMBLY LANGUAGE PROGRAM

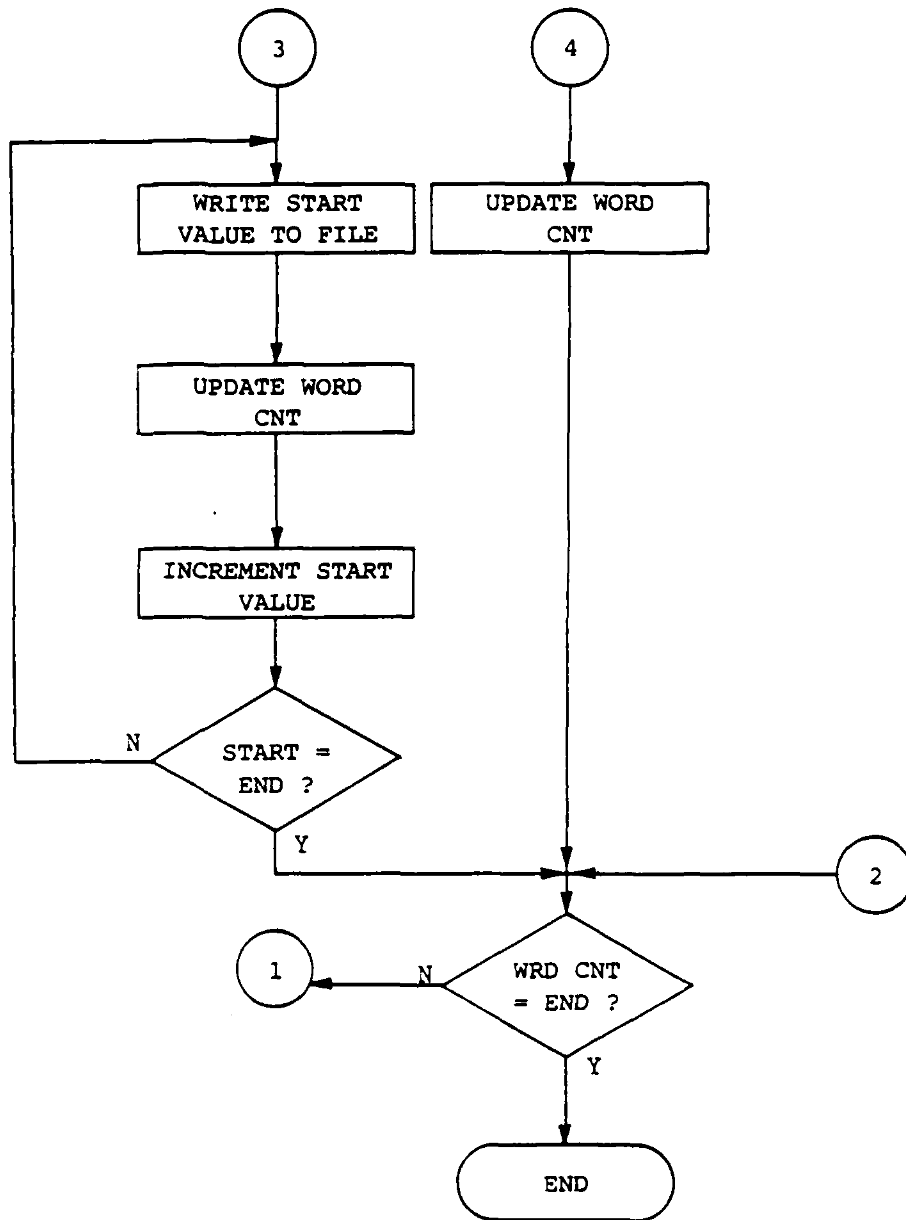
```
;
; LOAD DATA SEGMENT AND INDEX REGISTERS
; WITH START ADDRESS OF PROGRAM BUFFER
;
                MOV     AX,10H
                MOV     DS,AX
                MOV     DI,00
;
; INITIALIZE PIA'S
;
                MOV     DX,0FFFEH
                MOV     AX,0BCB4H
                OUT     [DX],AX
                MOV     AL,0CH
                OUTB    [DX],AL
;
; LOAD DX WITH INPUT PORT DATA ADDRESS, AND
; LOAD BX WITH INPORT PORT CONTROL ADDRESS
;
                MOV     DX,0FFF8H
                MOV     BX,0FFFCH
;
; CLEAR INPUT PORT
;
                IN      AX,[DX]
;
; CHECK STATUS FOR NEW WORD INPUT
;
                XCHG    DX,BX
LPSTAT1:        IN      AX,[DX]
                AND     AX,2020H
                JZ      LPSTAT1
;
; INPUT WORD TRANSFER COUNT
;
                XCHG    DX,BX
                IN      AX,[DX]
                XCHG    AX,CX
;
; INPUT WORDS UNTIL TRANSFER COUNT = 0
;
LPM:            XCHG    DX,BX
;
; CHECK STATUS FOR NEW WORD INPUT
;
LPSTAT2:        INB     AL,[DX]
```

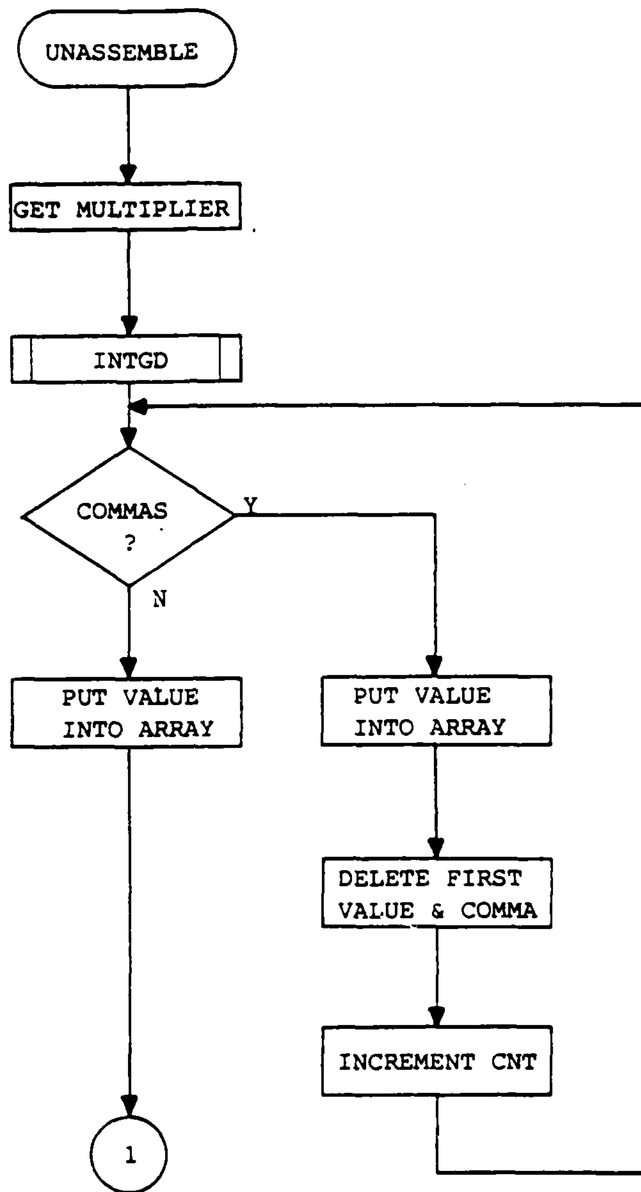

1
NEW WORD INPUT ?
N
Y
INPUT WORD
STORE WORD
INCREMENT DATA INDEX REGISTER
DECREMENT CNT
CNT=0 ?
N
2
Y
JMP TO 0100H

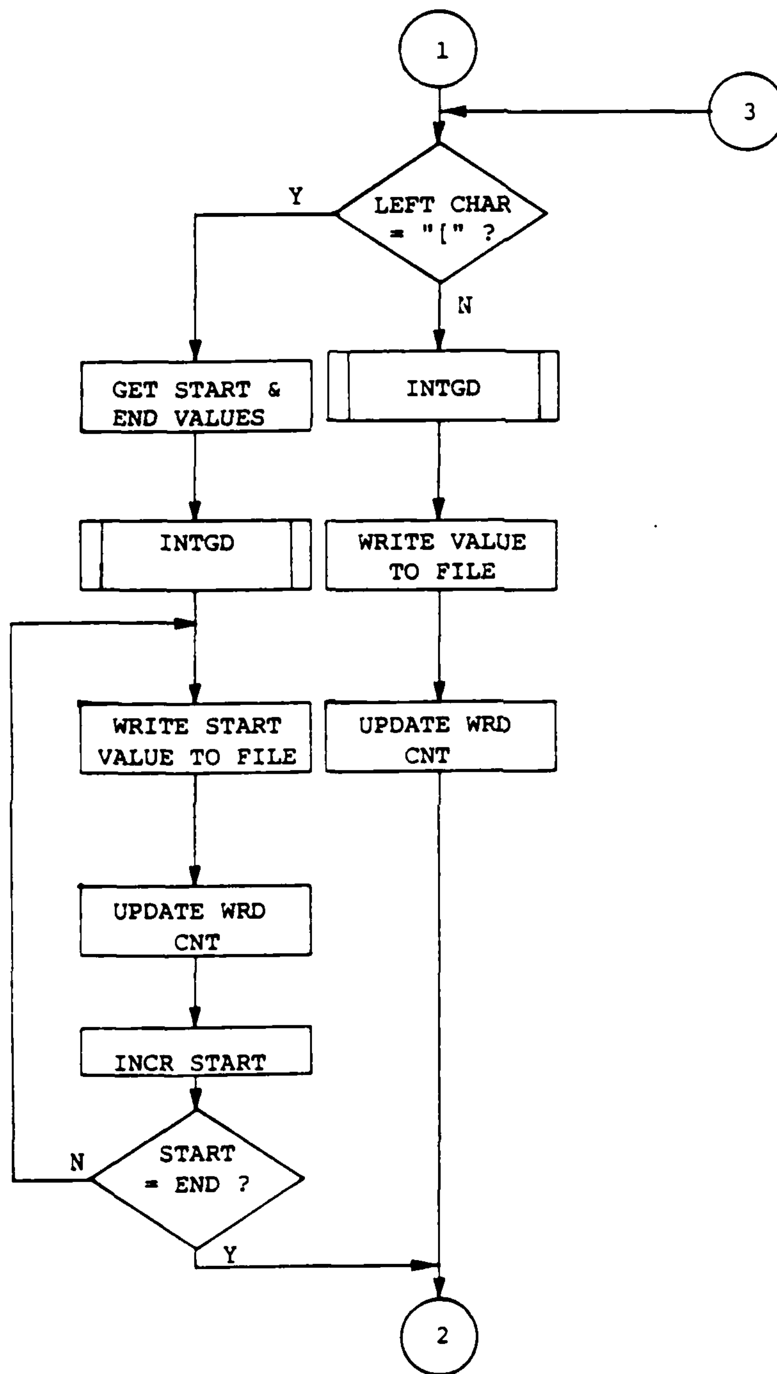

1
WRITE NUMBER OF GROUPS & WORDS TO FILE
INPUT RESULT MASK
WRITE RESULT MASK TO FILE
DIR_WRD_IN=FALSE
CREATE
DIR_WRD_IN=TRUE
CREATE
2

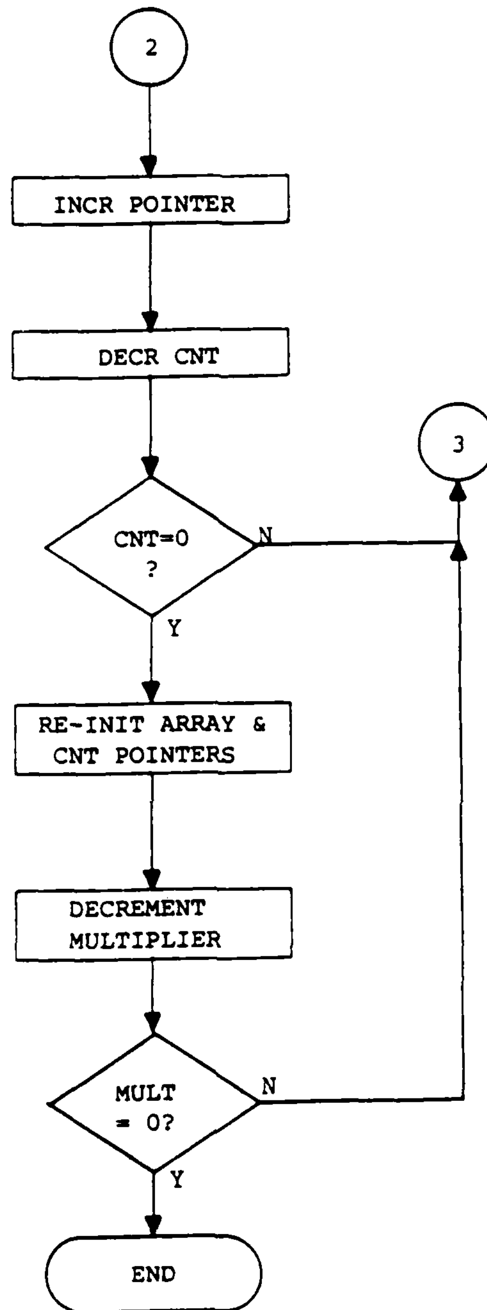

2
3
CLOCKS
TRANSFER
LISTING ?
N
Y
PRINTER /SCREEN ?
S
P
OUTPUT = PRINTER
OUTPUT = SCREEN
OPEN RESULT FILE
OUTPUT STRINGS
EOF ?
N
Y
END

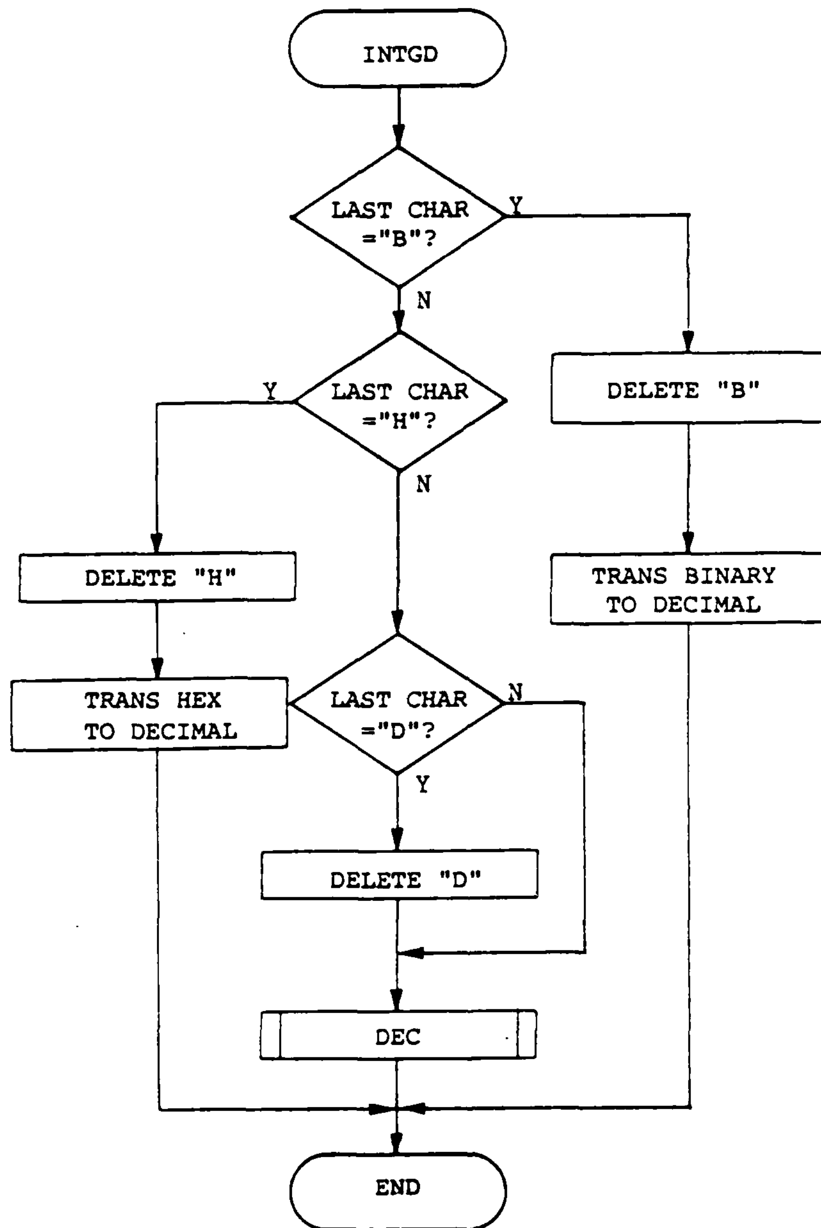

CREATE
INIT VARIABLES
1
INPUT VALUE
LEFT CHAR = "(" ?
Y
N
LEFT CHAR = "[" ?
Y
N
UNASSEMBLE
2
GET START & END VALUES
INTGD VALUES
INTGD START & END VALUES
WRITE VALUE TO FILE
3
4

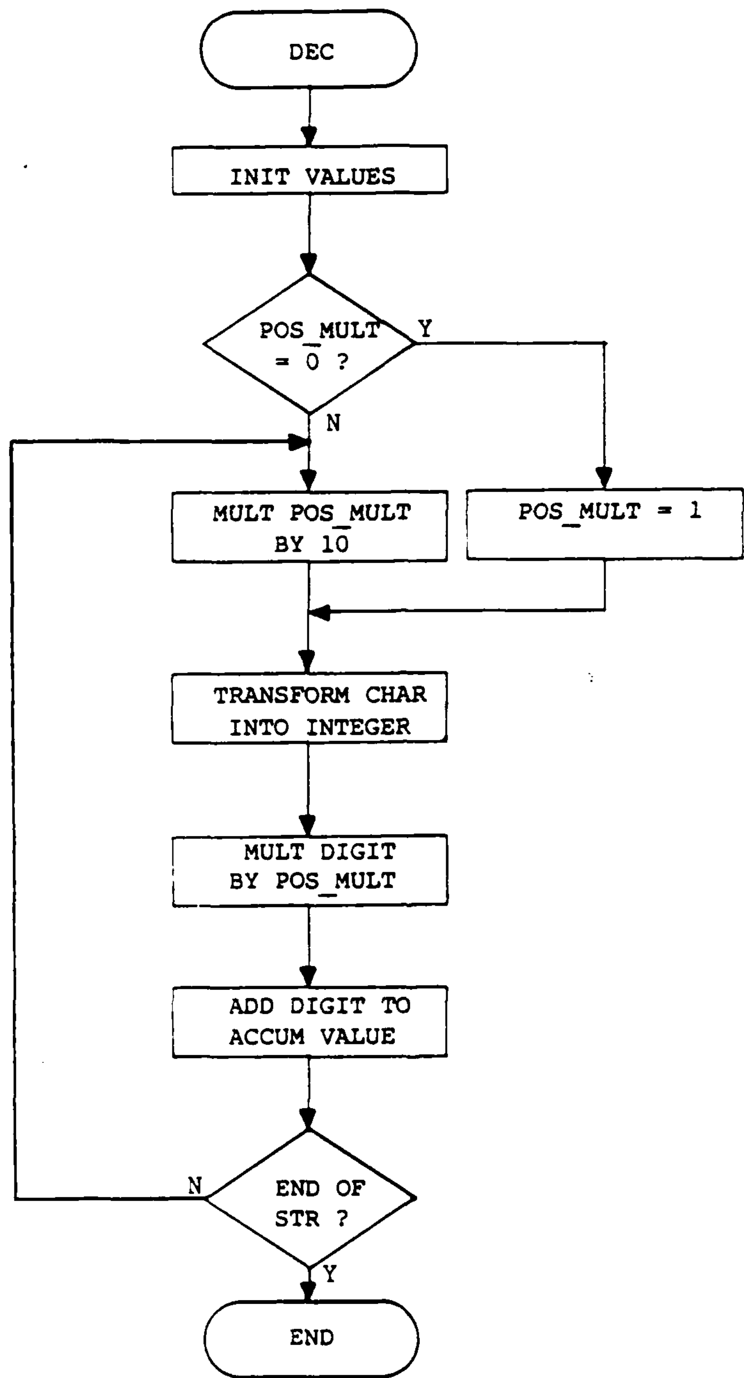

3
4
WRITE START VALUE TO FILE
UPDATE WORD CNT
UPDATE WORD CNT
INCREMENT START VALUE
N
START = END ?
Y
2
1
N
WRD CNT = END ?
Y
END

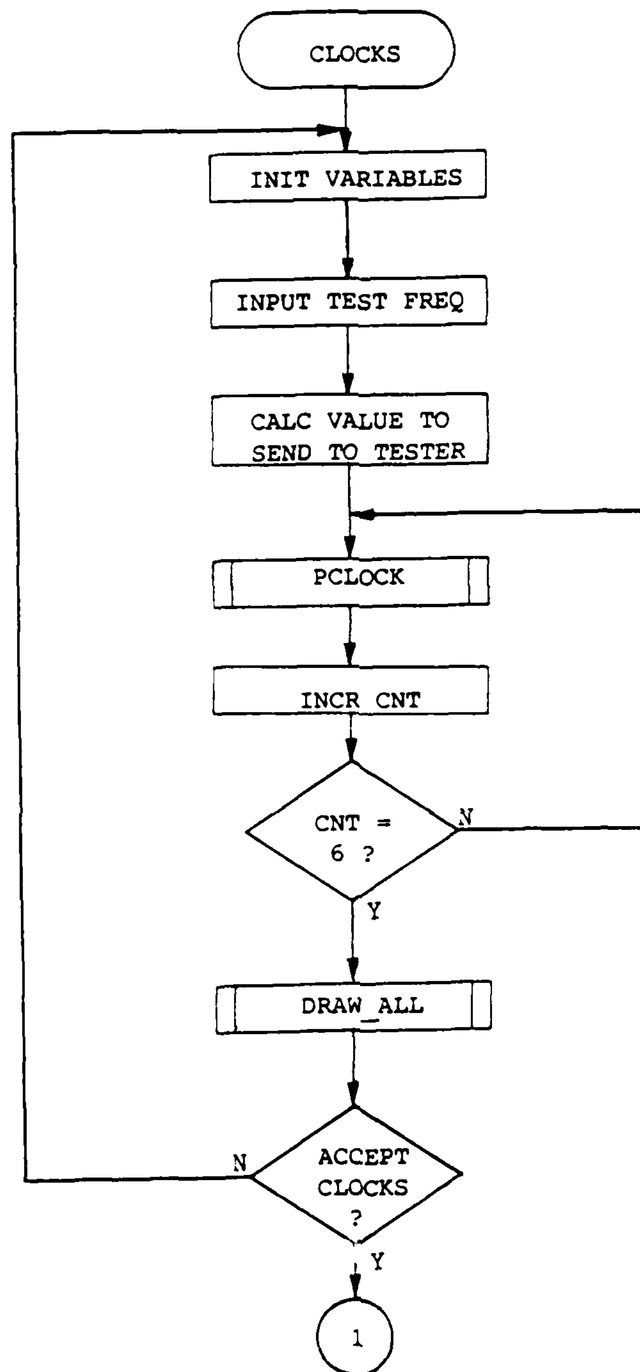

UNASSEMBLE
GET MULTIPLIER
INTGD
COMMAS ?
Y
N
PUT VALUE INTO ARRAY
PUT VALUE INTO ARRAY
DELETE FIRST VALUE & COMMA
INCREMENT CNT
1

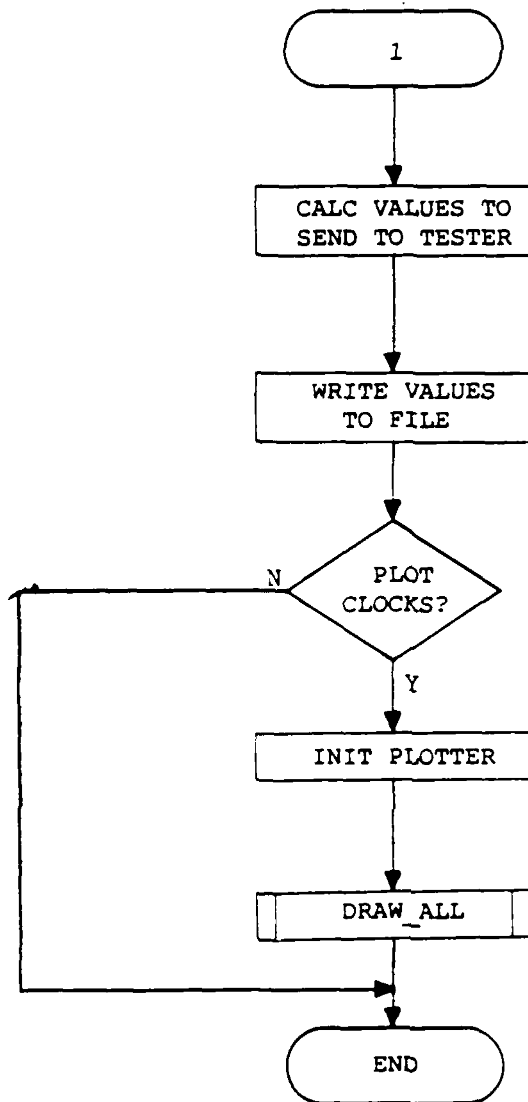

1
3
LEFT CHAR = "[" ?
Y
N
GET START & END VALUES
INTGD
INTGD
WRITE VALUE TO FILE
WRITE START VALUE TO FILE
UPDATE WRD CNT
UPDATE WRD CNT
INCR START
START = END ?
N
Y
2

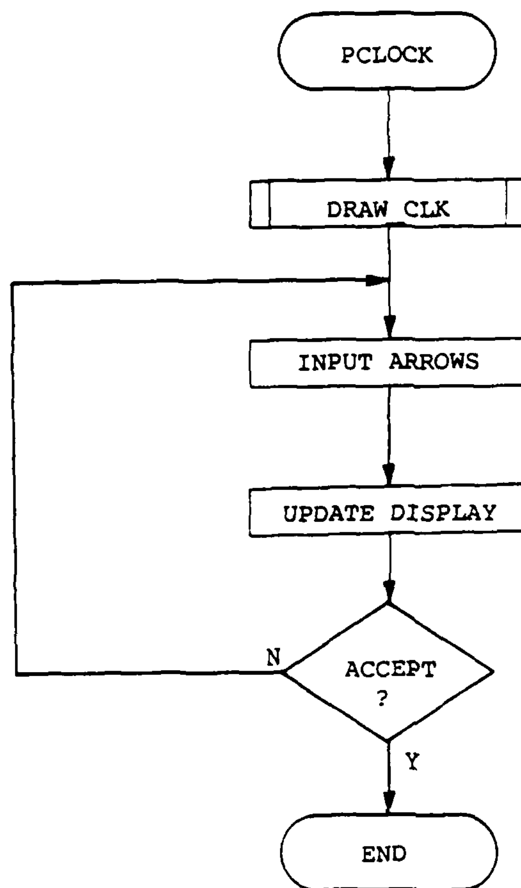

2
INCR POINTER
DECR CNT
3
CNT=0
?
N
Y
RE-INIT ARRAY &
CNT POINTERS
DECREMENT
MULTIPLIER
MULT
= 0?
N
Y
END

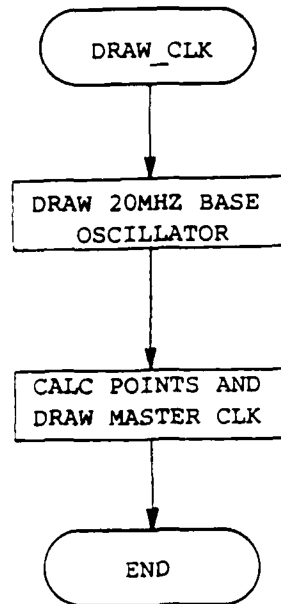

INTGD
LAST CHAR ="B"?
Y
N
DELETE "B"
TRANS BINARY TO DECIMAL
LAST CHAR ="H"?
Y
N
DELETE "H"
TRANS HEX TO DECIMAL
LAST CHAR ="D"?
N
Y
DELETE "D"
DEC
END

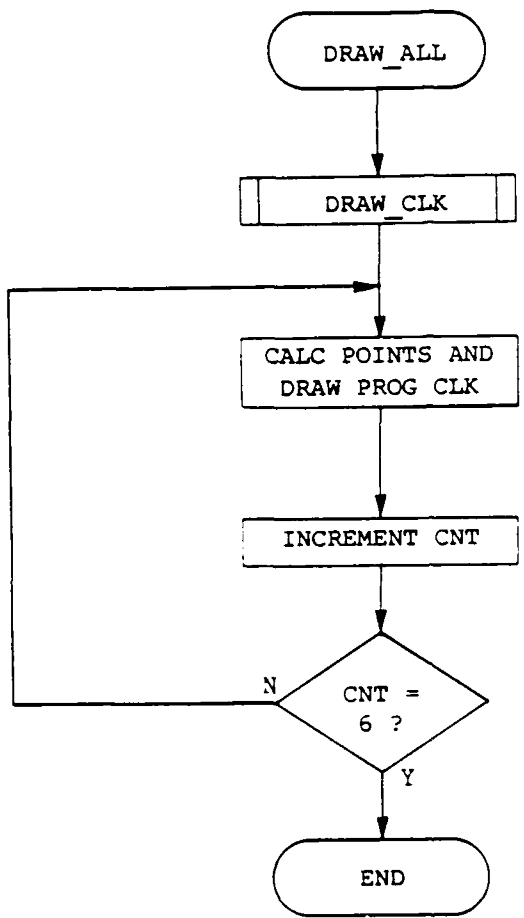

DEC
INIT VALUES
POS_MULT = 0 ?
Y
N
MULT POS_MULT BY 10
POS_MULT = 1
TRANSFORM CHAR INTO INTEGER
MULT DIGIT BY POS_MULT
ADD DIGIT TO ACCUM VALUE
END OF STR ?
N
Y
END


CLOCKS
INIT VARIABLES
INPUT TEST FREQ
CALC VALUE TO
SEND TO TESTER
PCLOCK
INCR CNT
CNT =
6 ?
N
Y
DRAW_ALL
ACCEPT
CLOCKS
?
N
Y
1


1
CALC VALUES TO
SEND TO TESTER
WRITE VALUES
TO FILE
PLOT
CLOCKS?
N
Y
INIT PLOTTER
DRAW_ALL
END


PCLOCK
DRAW CLK
INPUT ARROWS
UPDATE DISPLAY
ACCEPT
?
N
Y
END


DRAW_CLK
DRAW 20MHZ BASE OSCILLATOR
CALC POINTS AND DRAW MASTER CLK
END


DRAW_ALL
DRAW_CLK
CALC POINTS AND
DRAW PROG CLK
INCREMENT CNT
CNT =
6 ?
N
Y
END

```
                          WRITELN;
                          WRITELN('PLEASE RE-INPUT
                             DATA WORDS.');
                          WRITELN;
                          WRITELN;
                          OOPS:=TRUE;
                          STOP:=TRUE;
                        END;
        END;                            ( END N LOOP )
      END;                              ( END M LOOP )
    END;                    ( END PROCEDURE UNASSEMBLE )


      ( THIS PROCEDURE RETURNS A BOOLEAN ARRAY GIVEN
        A HEX, BINARY, OR DECIMAL STRING. )

PROCEDURE BOOL(BOOLI : INTEGER;  VAR PDIRV : TST_VAR);

VAR  CNT_VAR     :  INTEGER;

BEGIN                          ( PROCEDURE BOOL )


  FOR CNT_VAR:=1 TO 8 DO
    BEGIN
       IF ODD(BOOLI) THEN PDIRV[CNT_VAR]:=TRUE
                     ELSE PDIRV[CNT_VAR]:=FALSE;
      BOOLI:=TRUNC(BOOLI/2);
    END;
  END;                      ( PROCEDURE BOOL )


           ( THIS PROCEDURE CREATES THE DATA FILE. )

PROCEDURE CREATE(WRDS_PER_BLOCK,
                 NUMBER_OF_BLOCKS : INTEGER);

VAR
     LNGTH,CNT_VAR,
     PO,PM,PN,K          : INTEGER;
     STR,
     LEFT,RIGHT,
     VALUE,TEMPA         : S;

BEGIN                              ( CREATE )
  WHICH_GROUP:=1;
  WORDSLEFT:=0;
  STOP:=FALSE;
  OOPS:=FALSE;
```

```
        STRDELETE(ANUM[N],1,K+1);
        K:=STRPOS(']',ANUM[N]);
        STRDELETE(ANUM[N],K,1);
        PM:=INTGD(TMPNA);
        PN:=INTGD(ANUM[N]);
        FOR PO:= PM TO PN DO
          BEGIN
            F^:=PO;
            PUT(F);
            WORDSLEFT:=WORDSLEFT+1;
            IF WORDSLEFT=WRDS_PER_BLOCK
                THEN IF (M ¬= TIMES) AND
                       (N ¬= COUNT-1) AND (PO ¬= PN)
                       THEN BEGIN
                         WORDSLEFT:=0;
                         WHICH_GROUP:=WHICH_GROUP+2;
                       END
                ELSE BEGIN
                         N:=COUNT+1;
                         M:=TIMES+1;
                         WHICH_GROUP:=DATAGROUPS+1;
                         WRITELN;
                         WRITELN;
                         WRITELN('YOU INPUT TOO MANY
                            WORDS FOR THIS GROUP!');
                         WRITELN;
                         WRITELN('PLEASE RE-INPUT
                            DATA WORDS.');
                         WRITELN;
                         WRITELN;
                         OOPS:=TRUE;
                         STOP:=TRUE;
                        END;
           END;
          ANUM[N]:=TMPNB;
         END;
IF WORDSLEFT=WRDS_PER_BLOCK
    THEN IF (M ¬= TIMES) AND (N ¬= COUNT-1)
            THEN BEGIN
                    WORDSLEFT:=0;
                    WHICH_GROUP:=WHICH_GROUP+2;
                  END
            ELSE BEGIN
                    N:=COUNT+1;
                    M:=TIMES+1;
                    WHICH_GROUP:=DATAGROUPS+1;
                    WRITELN;
                    WRITELN;
                    WRITELN('YOU INPUT TOO MANY
                       WORDS FOR THIS GROUP!');
```

```
    COUNT,
    M,N,
    J,K,
    PM,PN,
    PO        : INTEGER;
    ANUM      : ARRAY[1..100] OF S;

BEGIN                                  ( UNASSEMBLE )
  TMS :=STOR;
  STRDELETE(TMS,1,1);
  J:=STRPOS('(',TMS);
  K:=STRLEN(TMS) - J;
  STRDELETE(TMS,J,K+1);
  TIMES:=INTGD(TMS);
  TMPNA:=STOR;
  STRDELETE(TMPNA,1,1);
  STRDELETE(TMPNA,1,STRPOS('(',TMPNA));
  TMPNB:=TMPNA;
  STOP:=FALSE;
  COUNT:=1;
  REPEAT
    K:=STRPOS(',',TMPNA);
    IF K=0 THEN BEGIN
                STOP:=TRUE;
                K:=STRPOS(')',TMPNA);
                END;
    STRDELETE(TMPNA,K,STRLEN(TMPNA)-K+1);
    STRDELETE(TMPNB,1,K);
    ANUM[COUNT]:=TMPNA;
    COUNT:=COUNT+1;
    TMPNA:=TMPNB;
  UNTIL STOP;
 FOR M:=1 TO TIMES DO
   BEGIN
     FOR N:=1 TO COUNT-1 DO
       BEGIN
       IF LEFTMOST(ANUM[N]) <> '['
         THEN
           BEGIN
             F^:=INTGD(HEXIT(ANUM[N]));
             PUT(F);
             WORDSLEFT:=WORDSLEFT+1;
           END
         ELSE
           BEGIN
             TMPNB:=ANUM[N];
             STRDELETE(ANUM[N],1,1);
             K:=STRPOS('.',ANUM[N]);
             TMPNA:=ANUM[N];
             STRDELETE(TMPNA,K,STRLEN(ANUM[N])-K+1);
```

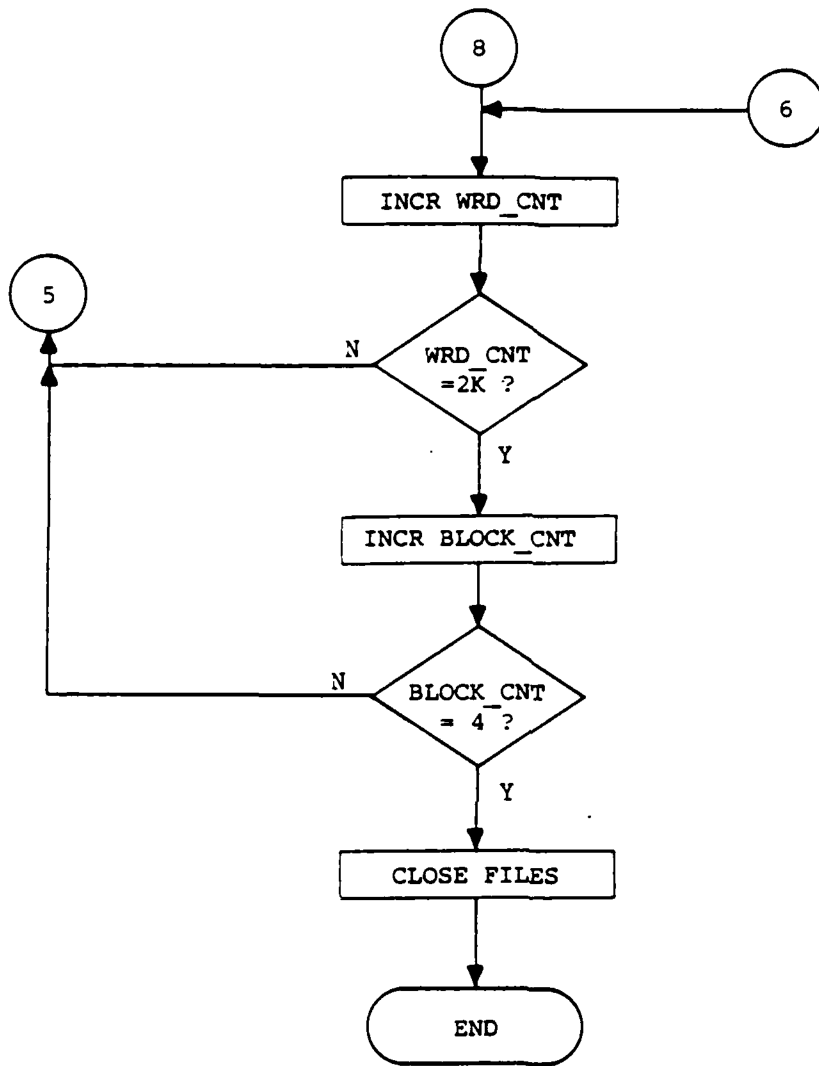
```
                                               K:=INTGD(STOR);
                                               STB:=INT_TO_HEXSTRNG(K);
                                                   END
                                              ELSE BEGIN
                                               K:=INTGD(STOR);
                                               STB:=INT_TO_HEXSTRNG(K);
                                                     END;

HEXIT:=STB;
END;                                              ( FUNCTION HEXIT )


           ( THIS FUNCTION RETURNS A BOOLEAN STRING
             FROM AN INTEGER )

FUNCTION  INT_TO_BOOLS(FIN:INTEGER) : S;

VAR FSTR  :  S;
    FCT   :  INTEGER;

BEGIN
  FSTR:='';
  WHILE FIN > 0 DO
    BEGIN
      IF ODD(FIN) THEN FSTR:='1'+FSTR
                  ELSE FSTR:='0'+FSTR;
      FIN:=TRUNC(FIN/2);
    END;
  FOR FCT:=1 TO (8-STRLEN(FSTR)) DO
    BEGIN
      FSTR:='0'+FSTR;
    END;
  INT_TO_BOOLS:=FSTR;
END;                                        (FUNCTION INT_TO_BOOLS)


     ( THIS PROCEDURE UNASSEMBLES A STRING INPUT
       TO THE DATAFILE CREATION PROCEDURE AND
       CREATES THE FILE VALUES FOR REPEATING SETS
       OF DATA. )

PROCEDURE UNASSEMBLE (STOR,RT,LF:S;
                      LN,WRDS_PER_BLOCK:INTEGER);

VAR
    STOP     : BOOLEAN;
    TMS,
    TMPNB,
    TMPNA    : S;
    TIMES,
```

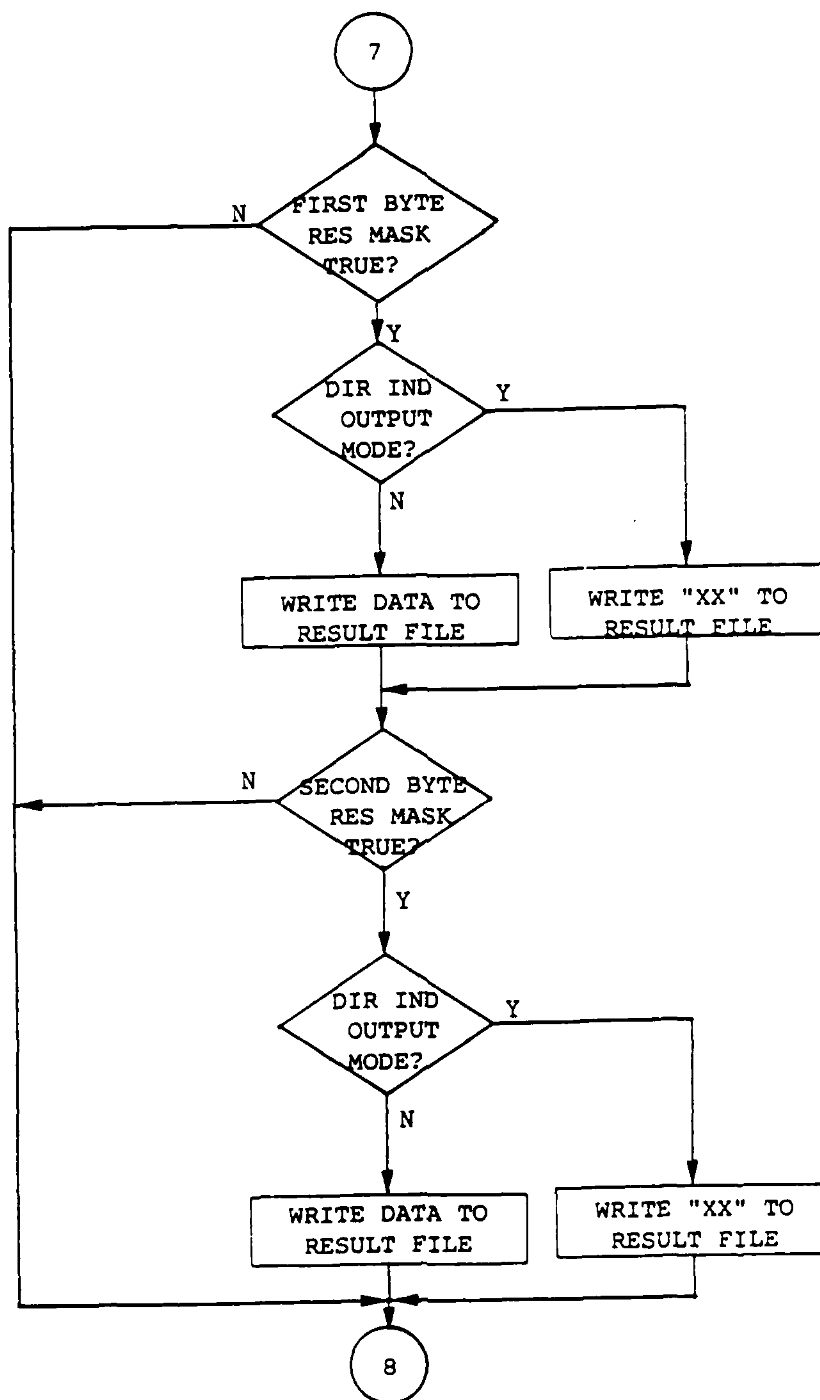
```
      REM[CNT_VAR]:=K MOD 16;
      K:=TRUNC(K/16);
    UNTIL K=0;
    FOR J:=CNT_VAR DOWNTO 1 DO
     BEGIN
       CASE REM[J] OF
         0:TMPSB:='0';
         1:TMPSB:='1';
         2:TMPSB:='2';
         3:TMPSB:='3';
         4:TMPSB:='4';
         5:TMPSB:='5';
         6:TMPSB:='6';
         7:TMPSB:='7';
         8:TMPSB:='8';
         9:TMPSB:='9';
         10:TMPSB:='A';
         11:TMPSB:='B';
         12:TMPSB:='C';
         13:TMPSB:='D';
         14:TMPSB:='E';
         15:TMPSB:='F';
       END;
       STRAPPEND(TMPSA,TMPSB);
     END;
  STRAPPEND(TMPSA,'H');
  INT_TO_HEXSTRNG:=TMPSA;
  END;                            (INT_TO_HEXSTRNG)


 ( THIS FUNCTION RETURNS A HEX STRING WITH AN
   APPENDED "H" FROM A BINARY OR DECIMAL STRING. )

FUNCTION HEXIT(STOR :S) : S;

VAR
        STB     : S;
        L,K     : INTEGER;

    BEGIN                             ( FUNCTION HEXIT )
      L:=STRLEN(STOR);
      IF RIGHTMOST(STOR) = 'H'
           THEN STB := STOR
           ELSE IF RIGHTMOST(STOR)='B'
                     THEN BEGIN
                           K:=INTGD(STOR);
                           STB:=INT_TO_HEXSTRNG(K);
                        END
                     ELSE IF RIGHTMOST(STOR)='D'
                             THEN BEGIN
```

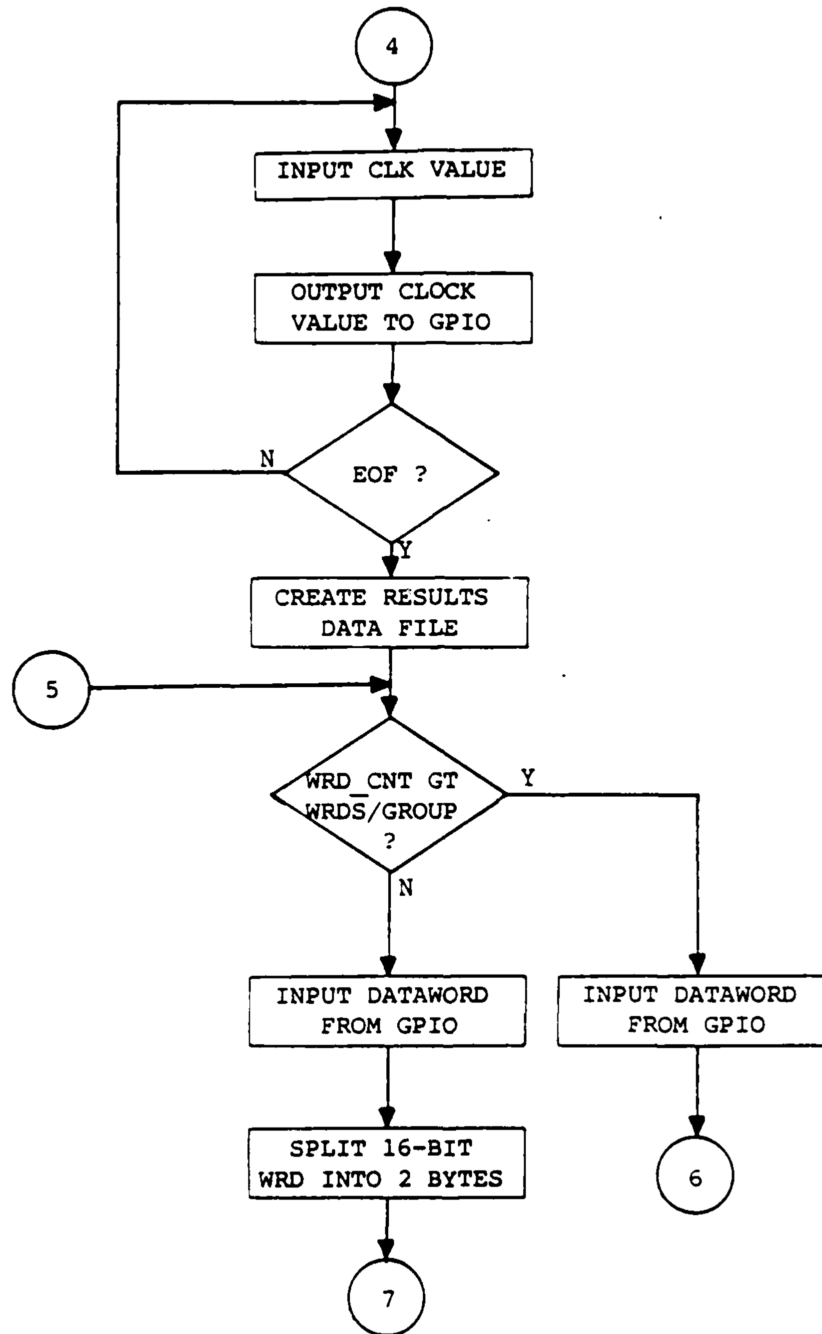
```
          IF TMPN='4' THEN EX:=4;
          IF TMPN='5' THEN EX:=5;
          IF TMPN='6' THEN EX:=6;
          IF TMPN='7' THEN EX:=7;
          IF TMPN='8' THEN EX:=8;
          IF TMPN='9' THEN EX:=9;
          D:=Z*EX+D;
          STRDELETE(TMPA,STRLEN(TMPA),1);
        END;
      DEC:=D;
    END;                          ( DEC )

BEGIN                              ( INTGD )
      NUMBER:=0;
      L:=STRLEN(TMPSA);
      IF RIGHTMOST(TMPSA)='B'
           THEN BEGIN
                   STRDELETE(TMPSA,L,1);
                   NUMBER:=BINARY(TMPSA);
                 END
           ELSE IF RIGHTMOST(TMPSA)='H'
                     THEN BEGIN
                             STRDELETE(TMPSA,L,1);
                             NUMBER:=HEX(TMPSA);
                           END
                     ELSE IF RIGHTMOST(TMPSA)='D'
                               THEN BEGIN
                                STRDELETE(TMPSA,L,1);
                                NUMBER:=DEC(TMPSA);
                                    END
                               ELSE NUMBER:=DEC(TMPSA);
INTGD:=NUMBER;
END;                                  ( END INTGD )


         ( THIS FUNCTION RETURNS A HEX STRING FROM
           A DECIMAL INTEGER. )

FUNCTION INT_TO_HEXSTRNG( K : INTEGER) : S;

  VAR  TMPSA,
       TMPSB      : S;
       REM        : ARRAY[0..10] OF INTEGER;
       CNT_VAR,J  : INTEGER;

  BEGIN                        ( FUNCTION INT_TO_HEX. )
    TMPSA:='';
    CNT_VAR:=0;
    REPEAT
      CNT_VAR:=CNT_VAR+1;
```

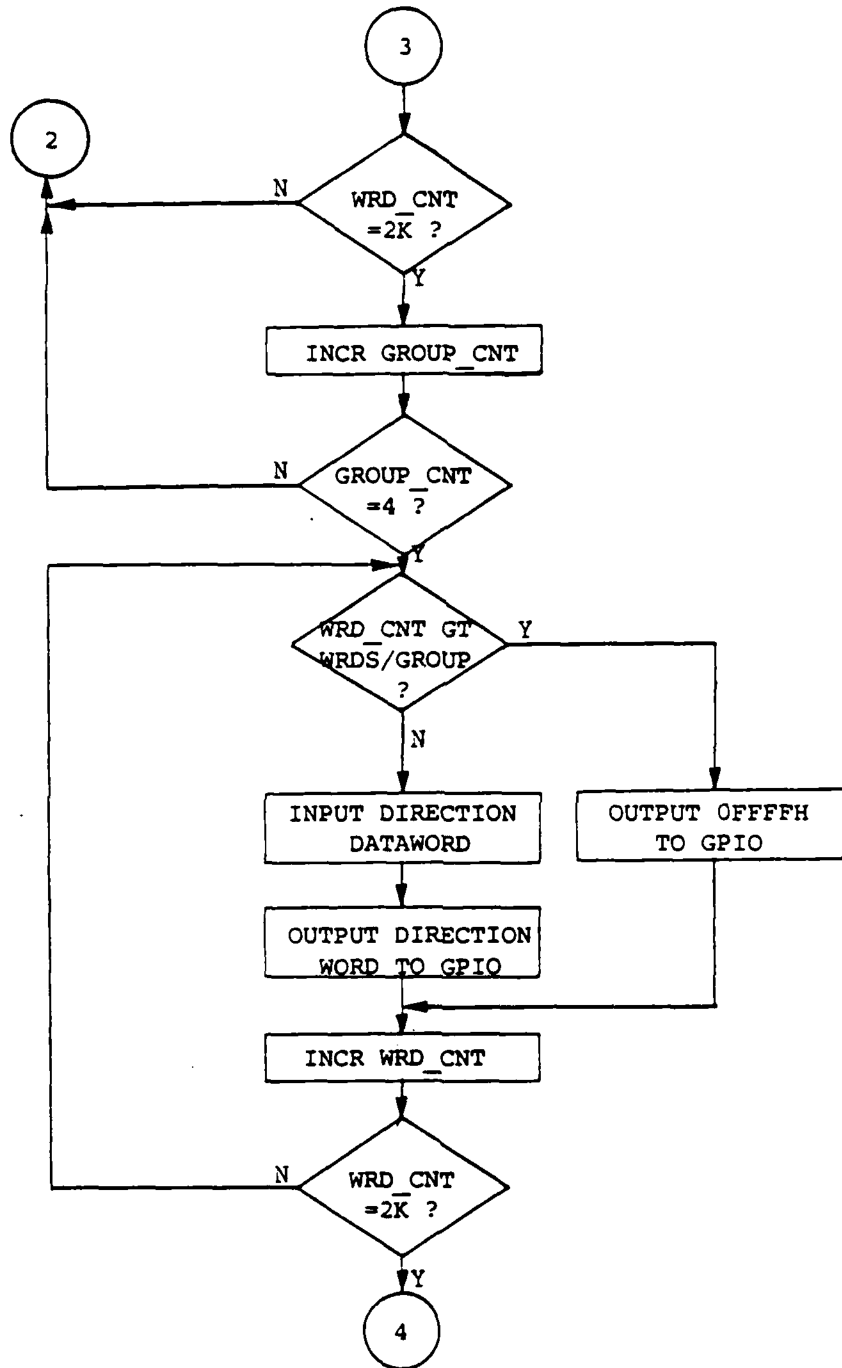
```
        ( THIS FUNCTION RETURNS THE RIGHTMOST CHARACTER
          OF A STRING. )

FUNCTION RIGHTMOST (STRX : S) : S;

BEGIN
  STRDELETE(STRX,1,(STRLEN(STRX)-1));
  RIGHTMOST := STRX;
END;


        ( THIS FUNCTION RETURNS THE LEFTMOST
          CHARACTER OF A STRING. )

FUNCTION LEFTMOST ( STRX : S) : S;

BEGIN
  STRDELETE(STRX,2,(STRLEN(STRX)-1));
  LEFTMOST := STRX;
END;


     ( THIS FUNCTION RETURNS A DECIMAL INTEGER
       WHEN GIVEN A HEX, BINARY, OR DECIMAL STRING. )

FUNCTION INTGD(TMPSA : S) : INTEGER;

VAR   NUMBER,L     : INTEGER;

      ( THIS FUNCTION RETURNS A DECIMAL INTEGER FROM
        A DECIMAL STRING. )

  FUNCTION DEC (TMPA : S) : INTEGER;

    VAR TMPN       : S;
        D,EX,C,Z   : INTEGER;

    BEGIN                       (FUNCTION DEC)
      D:=0;
      Z:=0;
      FOR C:=1 TO STRLEN(TMPA) DO
        BEGIN
          Z:=Z*10;
          IF Z=0 THEN Z:=1;
          TMPN:=RIGHTMOST(TMPA);
          IF TMPN='0' THEN EX:=0;
          IF TMPN='1' THEN EX:=1;
          IF TMPN='2' THEN EX:=2;
          IF TMPN='3' THEN EX:=3;
```

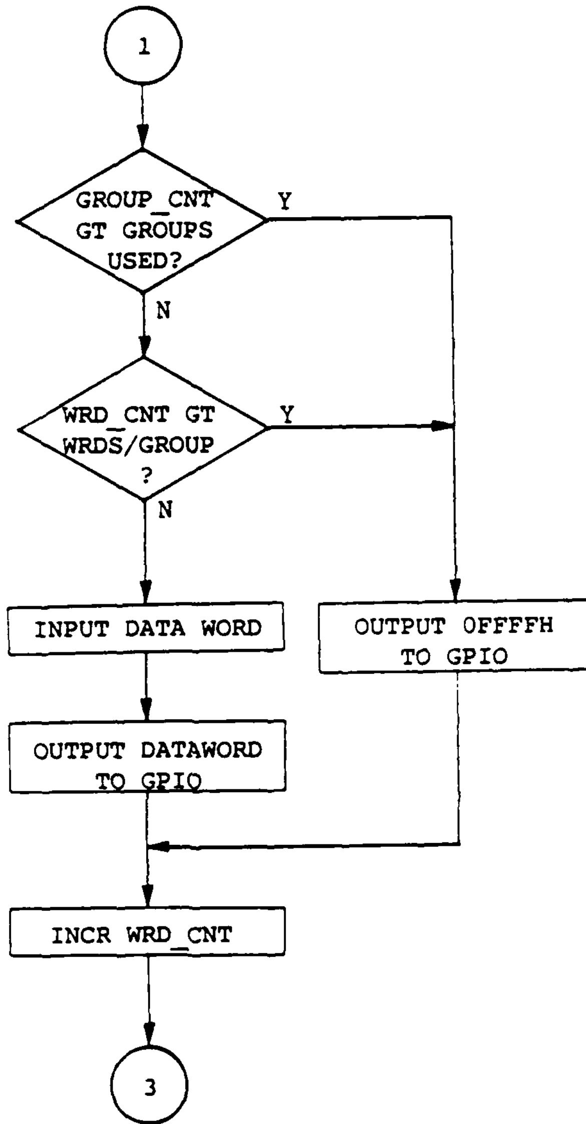
```
( THIS PROGRAM IS DESIGNED TO CREATE AND DOWNLOAD
  FILES TO THE FUNCTIONAL TESTER, AND READ THE
  RESULTS FROM THE D.U.T. BACK INTO A FILE. )

$REF 50$
$SYSPROG ON$

PROGRAM HDR (INPUT,OUTPUT);

IMPORT    IODECLARATIONS,
          DGL_LIB,
          GENERAL_0,
          GENERAL_1,
          GENERAL_2,
          GENERAL_3,
          GENERAL_4;

TYPE
      S            =  STRING[255];
      TST_VAR      =  ARRAY[1..8] OF BOOLEAN;
      D_A          =  ARRAY[1..2050,1..8] OF INTEGER;
      PTRD_A       =  ¢D_A;
      CLK_ARRAY    =  ARRAY[1..7] OF S;

VAR
      CHR_ARRY        : PACKED ARRAY[1..79] OF CHAR;
      CK_ARRAY        : CLK_ARRAY;
      OK_TO_DO,
      DIR_WRD_IN,
      STOP,OOPS       : BOOLEAN;
      P               : FILE OF PACKED ARRAY[1..79]
                        OF CHAR;
      T_F             : FILE OF S;
      F               : FILE OF INTEGER;
      WRDS_PER_BLOCK,
      NUMBER_OF_BLOCKS,
      WHICH_GROUP,
      WORDSLEFT,
      DATAGROUPS,
      DATAWORDS,
      CNT_VAR          : INTEGER;
      DATAFILE,
      RESULT,
      TESTFILE,
      STEMP,STEMP1     : S;
      IO_RESVAR,
      DIRVAR          : TST_VAR;
```

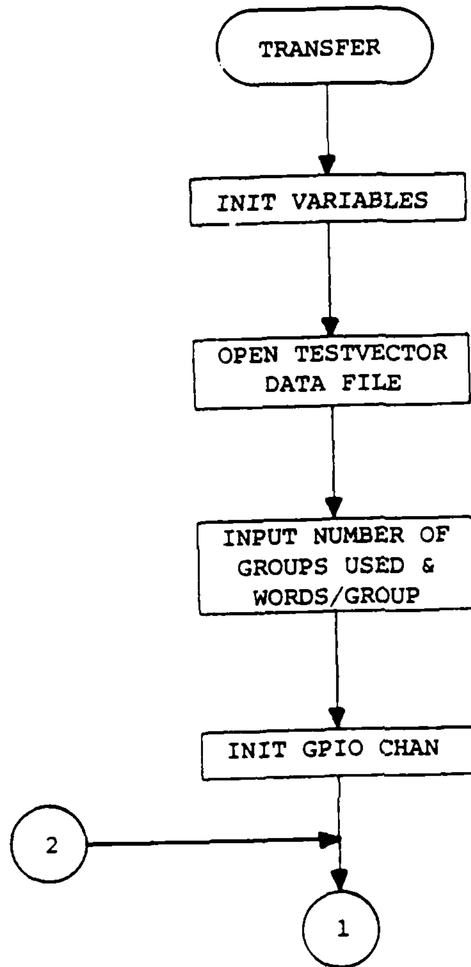

8
6
INCR WRD_CNT
5
N
WRD_CNT
=2K ?
Y
INCR BLOCK_CNT
N
BLOCK_CNT
= 4 ?
Y
CLOSE FILES
END


7
FIRST BYTE
RES MASK
TRUE?
N
Y
DIR IND
OUTPUT
MODE?
Y
N
WRITE DATA TO
RESULT FILE
WRITE "XX" TO
RESULT FILE
SECOND BYTE
RES MASK
TRUE?
N
Y
DIR IND
OUTPUT
MODE?
Y
N
WRITE DATA TO
RESULT FILE
WRITE "XX" TO
RESULT FILE
8


4
INPUT CLK VALUE
OUTPUT CLOCK VALUE TO GPIO
N
EOF ?
Y
CREATE RESULTS DATA FILE
5
WRD_CNT GT WRDS/GROUP ?
Y
N
INPUT DATAWORD FROM GPIO
INPUT DATAWORD FROM GPIO
SPLIT 16-BIT WRD INTO 2 BYTES
6
7


3
2
N
WRD_CNT
=2K ?
Y
INCR GROUP_CNT
N
GROUP_CNT
=4 ?
Y
WRD_CNT GT
WRDS/GROUP
?
Y
N
INPUT DIRECTION
DATAWORD
OUTPUT 0FFFFH
TO GPIO
OUTPUT DIRECTION
WORD TO GPIO
INCR WRD_CNT
N
WRD_CNT
=2K ?
Y
4


1
GROUP_CNT GT GROUPS USED?
Y
N
WRD_CNT GT WRDS/GROUP ?
Y
N
INPUT DATA WORD
OUTPUT OFFFFH TO GPIO
OUTPUT DATAWORD TO GPIO
INCR WRD_CNT
3


TRANSFER
INIT VARIABLES
OPEN TESTVECTOR DATA FILE
INPUT NUMBER OF GROUPS USED & WORDS/GROUP
INIT GPIO CHAN
2
1

```
WHILE NOT STOP DO
  BEGIN
    WRITELN;
    IF NOT DIR_WRD_IN THEN
        WRITE('DATA GROUP NUMBERS  ':19,
             WHICH_GROUP:1,'&':1,WHICH_GROUP+1:1)
       ELSE
       WRITE('DIRECTION':9);
    WRITE(' WORDS LEFT =':13,
         WRDS_PER_BLOCK-WORDSLEFT:4);
    WRITELN;
    WRITELN;
    WRITE('>>');
    READLN(VALUE);
    VALUE:=STRRTRIM(VALUE);
    VALUE:=STRLTRIM(VALUE);
    LNGTH:=STRLEN(VALUE);
    LEFT:=LEFTMOST(VALUE);
    RIGHT:=RIGHTMOST(VALUE);
    IF (LEFT <> '(') AND (LEFT <> '[')
        THEN BEGIN
               STR:=HEXIT(VALUE);
               F^:=INTGD(STR);
               PUT(F);
               WORDSLEFT:=WORDSLEFT+1;
               IF WORDSLEFT=WRDS_PER_BLOCK
                   THEN BEGIN
                    WHICH_GROUP:=WHICH_GROUP+2;
                    WORDSLEFT:=0;
                        END;
             END;
    IF LEFT = '('
       THEN  UNASSEMBLE(VALUE,RIGHT,LEFT,LNGTH,
                          WRDS_PER_BLOCK);
       IF LEFT = '[' THEN
        BEGIN
          STRDELETE(VALUE,1,1);
          K:=STRPOS('.',VALUE);
          TEMPA:=VALUE;
          STRDELETE(TEMPA,K,STRLEN(VALUE)-K+1);
          STRDELETE(VALUE,1,K+1);
          K:=STRPOS(']',VALUE);
          STRDELETE(VALUE,K,1);
          PM:=INTGD(TEMPA);
          PN:=INTGD(VALUE);
          FOR PO:= PM TO PN DO
            BEGIN
              F^:=PO;
              PUT(F);
              WORDSLEFT:=WORDSLEFT+1;
```

```
                    IF WORDSLEFT=WRDS_PER_BLOCK
                        THEN IF PO = PN
                          THEN BEGIN
                             WORDSLEFT:=0;
                             WHICH_GROUP:=WHICH_GROUP+2;
                           END
                           ELSE
                            BEGIN
                             PO := PN+1;
                             WHICH_GROUP:=DATAGROUPS+1;
                             WRITELN;
                             WRITELN;
                             WRITELN('YOU INPUT TOO MANY
                                WORDS FOR THIS GROUP!');
                             WRITELN;
                             WRITELN('PLEASE RE-INPUT
                                DATA WORDS.');
                             WRITELN;
                             WRITELN;
                             OOPS:=TRUE;
                             STOP:=TRUE;
                            END;
                   END;
            END;
         IF WHICH_GROUP > NUMBER_OF_BLOCKS THEN STOP:=TRUE;
        END
      END;                                 ( CREATE )



          ( THIS PROCEDURE GETS THE TEST-CLOCK
            VALUES FOR THE DATA FILE. )

PROCEDURE CLOCKS(VAR VALUE1 : CLK_ARRAY);

TYPE      LISTI = ARRAY[1..3,1..6] OF INTEGER;
          LISTR = ARRAY[1..2,1..6] OF REAL;

CONST     MASTERCLK = 20000000;

VAR       I1,I2,I3,I4,
          I5,TSTNUM,DI,
          TSTFRQ,
          CONTROL,
          ERROR,
          ESCAP               :  INTEGER;
          SCH                 :  CHAR;
          R1,R2               :  REAL;
          S1,S2               :  S;
          CORRECT             :  BOOLEAN;
```

```
          MLISTI            :  LISTI;
          MLISTR            :  LISTR;


      ( THIS PROCEDURE POSITIONS THE CURSOR
        AT THE BOTTOM OF THE SCREEN. )

PROCEDURE SKIPPAGE;

  VAR   I : INTEGER;

  BEGIN                                  ( SKIPPAGE )
    WRITELN(CHR(12));
    FOR I:=1 TO 24 DO
      BEGIN
        WRITELN;
      END;
  END;                                   ( SKIPPAGE )


     ( THIS PROCEDURE DRAWS THE GRAPHICS ARROW. )

PROCEDURE ARROW(ARR_COLOR : INTEGER;
                ARR_X,ARR_Y : REAL);


  BEGIN                                     ( ARROW )
    MOVE(ARR_X,ARR_Y);
    SET_COLOR(ARR_COLOR);
    LINE(ARR_X,ARR_Y+0.1);
    LINE(ARR_X,ARR_Y);
    LINE(ARR_X+0.02,ARR_Y+0.02);
    LINE(ARR_X,ARR_Y);
    LINE(ARR_X-0.02,ARR_Y+0.02);
    SET_COLOR(1);
  END;                                      ( ARROW )

 ( THIS PROCEDURE BLANKS OUT THE AREA FOR MESSAGES. )

PROCEDURE B_BOX( ZY:REAL);

  VAR   YV,YVF : REAL;

  BEGIN                                     ( B_BOX )
    SET_COLOR(0);
    YV:=ZY+0.1;
    YVF:=ZY-0.01;
    REPEAT
      MOVE(-1.0,YV);
      LINE(1.0,YV);
```

```
      YV:=YV-0.005;
    UNTIL YV-=YVF;
    SET_COLOR(1);
  END;                                ( B_BOX )


      ( THIS PROCEDURE DRAWS THE TESTCLOCK AND THE
        BASE OSCILLATOR TWICE. )

PROCEDURE DRAW_HCLK(VAR INTERVAL:REAL);

VAR     FR1,FR2  : REAL;
        FS2      : S;

    BEGIN                             ( DRAW_HCLK )
      FR1:=MASTERCLK/TSTFRQ;
      INTERVAL:=(2.0/FR1)/2;
      MOVE(-0.95,0.9);
      GTEXT('BASE OSCILLATOR (20 MHz)');
      MOVE(-1.0,0.7);
      FR2:=-1.0;
      REPEAT
        FR2:=FR2+INTERVAL;
        LINE(FR2,0.7);
        LINE(FR2,0.75);
        LINE(FR2,0.7);
      UNTIL FR2 >=1;
      MOVE(-0.95,0.5);
      FS2:='';
      STRWRITE(FS2,1,DI,TSTFRQ:1);
      FS2:='TEST FREQ.= '+FS2+' Hz.';
      GTEXT(FS2);
      MOVE(-1.0,0.35);
      LINE(-1.0,0.4);
      LINE(-0.5,0.4);
      LINE(-0.5,0.35);
      LINE(0.0,0.35);
      LINE(0.0,0.4);
      LINE(0.5,0.4);
      LINE(0.5,0.35);
      LINE(1.0,0.35);
      LINE(1.0,0.4);
    END;                              ( DRAW_HCLOCK )


         ( THIS PROCEDURE DRAWS THE TESTCLOCK AND
           THE BASE OSCILLATOR. )

    PROCEDURE DRAW_CLK(VAR INTERVAL:REAL);
```

```
    VAR    FR1,FR2   : REAL;
           FS2       : S;

    BEGIN                                 ( DRAW_CLK )
      FR1:=MASTERCLK/TSTFRQ;
      INTERVAL:=2.0/FR1;
      MOVE(-0.95,0.9);
      GTEXT('BASE OSCILLATOR (20 MHz)');
      MOVE(-1.0,0.7);
      FR2:=-1.0;
      REPEAT
        FR2:=FR2+INTERVAL;
        LINE(FR2,0.7);
        LINE(FR2,0.75);
        LINE(FR2,0.7);
      UNTIL FR2 >=1;
      MOVE(-0.95,0.5);
      FS2:='';
      STRWRITE(FS2,1,DI,TSTFRQ:1);
      FS2:='TEST FREQ.= '+FS2+' Hz.';
      GTEXT(FS2);
      MOVE(-1.0,0.35);
      LINE(-1.0,0.4);
      LINE(0.0,0.4);
      LINE(0.0,0.35);
      LINE(1.0,0.35);
      LINE(1.0,0.4);
    END;                                ( DRAW_CLK )


                         ( DRAWS ALL OF THE CLOCKS. )

PROCEDURE DRAW_ALL(VAR MLISTR:LISTR;
                   VAR MLISTI:LISTI);

  VAR   I1          : INTEGER;
        R1,R2,RA,
        RX1,RX2     : REAL;
        S1          : S;

  BEGIN                                 ( DRAW_ALL )
    DRAW_HCLK(R1);
    SET_CHAR_SIZE(0.045,0.065);
    FOR I1:=1 TO 6 DO
      BEGIN
        R1:=0.35-I1*0.18;
             RA:=MLISTR[1,I1];
             R2:=MLISTR[2,I1];
             IF RA<0 THEN RX1:=-1.0+((1-ABS(RA))/2);
             IF RA>0 THEN RX1:=-1.0+((1+RA)/2);
```

```
          IF RA=0 THEN RX1:=-0.5;
          IF R2<0 THEN RX2:=-1.0+((1-ABS(R2))/2);
          IF R2>0 THEN RX2:=-1.0+((1+R2)/2);
          IF R2=0 THEN RX2:=-0.5;
     IF RX1 > RX2 THEN
        BEGIN
          MOVE(-1.0,R1+0.05);
          LINE(RX2,R1+0.05);
          LINE(RX2,R1);
          LINE(RX1,R1);
          LINE(RX1,R1+0.05);
          LINE(0.0,R1+0.05);
          IF MLISTI[3,I1]=1 THEN
                BEGIN
                  RX1:=RX1+1;
                  RX2:=RX2+1;
                  LINE(RX2,R1+0.05);
                  LINE(RX2,R1);
                  LINE(RX1,R1);
                  LINE(RX1,R1+0.05);
                  LINE(1.0,R1+0.05);
                END;
          IF MLISTI[3,I1]=2 THEN LINE(1.0,R1+0.05);
       END;
     IF RX1 < RX2 THEN
        BEGIN
           MOVE(-1.0,R1);
           LINE(RX1,R1);
           LINE(RX1,R1+0.05);
           LINE(RX2,R1+0.05);
           LINE(RX2,R1);
           LINE(0.0,R1);
           IF MLISTI[3,I1]=1 THEN
                 BEGIN
                   RX1:=RX1+1;
                   RX2:=RX2+1;
                   LINE(RX1,R1);
                   LINE(RX1,R1+0.05);
                   LINE(RX2,R1+0.05);
                   LINE(RX2,R1);
                   LINE(1.0,R1);
                 END;
            IF MLISTI[3,I1]=2 THEN LINE(1.0,R1);
          END;
     CASE I1 OF
       1 : S1:='PCLK1';
       2 : S1:='PCLK2';
       3 : S1:='PCLK3';
       4 : S1:='PCLK4';
       5 : S1:='PDOUTEN';
```

```
          6 : S1:='PSTBIN';
        END;
        MOVE (-0.95,0.42-I1*0.18);
        GTEXT(S1);
      END;
      SET_LINE_STYLE(5);
      MOVE(0.0,0.4);
      LINE(0.0,-0.73);
      SET_LINE_STYLE(1);
      SET_CHAR_SIZE(0.065,0.085);
    END;                               ( DRAW_ALL )


( THIS PROCEDURE SETS UP THE PROGRAMMABLE
  CLOCKS FROM USER INPUT. )

PROCEDURE  PCLOCK(IV           : INTEGER;
                    VAR   PLISTI : LISTI;
                    VAR   PLISTR : LISTR);

  VAR   NS,LNS,
        PU,PD,PT       : INTEGER;
        INTERVAL,
        DX,DY,
        LDX,LDY,
        VU,VD,
        ZY,VT           : REAL;
        PS2             : S;
        PS1,PSL         : CHAR;

  BEGIN                            ( PROCEDURE PCLOCK )
    DRAW_CLK(INTERVAL);
    DX:=-1.0+3*INTERVAL;
    DY:=0.0;
    LDX:=DX;
    LDY:=DY;
    VU:=-1.0+3*INTERVAL;
    VD:=-1.0+4*INTERVAL;
    ARROW(1,DX,DY);
    NS:=150;
    IF IV <>6
       THEN
         BEGIN
           PU:=150;
           PD:=200;
           VU:=-1.0+INTERVAL*3;
           VD:=-1.0+INTERVAL*4;
         END
       ELSE BEGIN
              PU:=200;
```

```
          PD:=150;
          VU:=-1.0+INTERVAL*4;
          VD:=-1.0+INTERVAL*3;
        END;
PSL:=CHR(28);
PS1:=' ';
REPEAT
  SKIPPAGE;
  IF IV=5 THEN WRITELN('THIS IS FOR CLOCK
                   "PDOUTEN" [LOW TRUE].')
          ELSE IF IV=6 THEN WRITELN('THIS IS
                   FOR CLOCK "PSTBIN."')
          ELSE WRITELN('THIS IS FOR PROGRAMABLE
                   CLOCK PCLK':34,IV:1);
  IF (PS1=CHR(8)) OR (PS1=CHR(28)) THEN PSL:=PS1;
  PS1:=' ';
  READCHAR(2,PS1);
  LNS:=NS;
  IF PS1=CHR(28)THEN
            BEGIN
              NS:=NS+50;
              LDX:=DX;
              DX:=DX+INTERVAL;
            END;
  IF PS1= CHR(8) THEN
            BEGIN
              NS:=NS-50;
              LDX:=DX;
              DX:=DX-INTERVAL;
            END;
  IF (PS1=CHR(13)) AND (PSL=CHR(28)) THEN
      BEGIN
        LDX:=DX;
        DX:=DX+(INTERVAL*10);
        NS:=NS+50*10;
      END;
  IF (PS1=CHR(13)) AND (PSL=CHR(8)) THEN
      BEGIN
        LDX:=DX;
        DX:=DX-(INTERVAL*10);
        NS:=NS-10*50;
      END;
  IF (DX<-1.0+INTERVAL*3) OR (DX>1.0) THEN
                        BEGIN
                          NS:=LNS;
                          DX:=LDX;
                        END;
  IF (PS1=CHR(31)) AND (PD<>NS) THEN
               BEGIN
                 PU:=NS;
```

```
                  VU:=DX;
                END;
IF (PS1=CHR(10)) AND (PU<>NS) THEN
                BEGIN
                  PD:=NS;
                  VD:=DX;
                END;
IF PS1='S' THEN
                 BEGIN
                   PT:=PD;
                   PD:=PU;
                   PU:=PT;
                   VT:=VD;
                   VD:=VU;
                   VU:=VT;
                 END;
ARROW(0,LDX,LDY);
ARROW(1,DX,DY);
ZY:=-0.35;
B_BOX(ZY);
PS2:='';
STRWRITE(PS2,1,DI,NS:1);
MOVE(-1.0,-0.35);
PS2:=PS2+'ns.';
GTEXT(PS2);
ZY:=-0.5;
B_BOX(ZY);
MOVE(-1.0,-0.5);
PS2:='';
STRWRITE(PS2,1,DI,PU:1);
PS2:='UP TRANS. '+PS2+'ns';
GTEXT(PS2);
ZY:=-0.65;
B_BOX(ZY);
MOVE(-1.0,-0.65);
PS2:='';
STRWRITE(PS2,1,DI,PD:1);
PS2:='DOWN TRANS. '+PS2+'ns';
GTEXT(PS2);
IF ((PS1=CHR(31)) OR (PS1=CHR(10)) OR
          (PS1='S')) AND (VU>VD) THEN
          BEGIN
            B_BOX(-0.2);
            MOVE(-1.0,-0.1);
            LINE(VD,-0.1);
            LINE(VD,-0.15);
            LINE(VU,-0.15);
            LINE(VU,-0.1);
            LINE(1.0,-0.1);
          END;
```

```
      IF ((PS1=CHR(31)) OR (PS1=CHR(10)) OR
                 (PS1='S')) AND (VU<=VD)THEN
                 BEGIN
                   B_BOX(-0.2);
                   MOVE(-1.0,-0.15);
                   LINE(VU,-0.15);
                   LINE(VU,-0.1);
                   LINE(VD,-0.1);
                   LINE(VD,-0.15);
                   LINE(1.0,-0.15);
                 END;
    IF PS1=CHR(13) THEN PS1:=PSL;
    UNTIL (PS1=CHR(3)) AND ((IV<>5) OR (PD<PU)) AND
                               ((IV<>6) OR (PD>PU));
    PLISTI[1,IV]:=PU;
    PLISTI[2,IV]:=PD;
    PLISTR[1,IV]:=VU;
    PLISTR[2,IV]:=VD;
    ZY:=-0.5;
    REPEAT
      B_BOX(ZY);
      ZY:=ZY-0.15;
    UNTIL ZY < -0.70;
    IF IV<5 THEN
       BEGIN
         REPEAT
           SKIPPAGE;
           WRITELN('DO YOU WANT THIS CLOCK:
                             1-EVERY CYCLE');
           WRITELN('         2-EVERY OTHER CYCLE');
           WRITE('>>');
           READLN(PS1);
         UNTIL (PS1='1') OR (PS1='2');
       END
       ELSE PS1:='1';
    IF PS1='1' THEN PLISTI[3,IV]:=1
               ELSE PLISTI[3,IV]:=2;
    CLEAR_DISPLAY;
  END;                                    ( PCLOCK )


BEGIN                            ( PROCEDURE CLOCKS )
    CONTROL:=0;
    I5:=0;
    TSTNUM:=0;
    ERROR:=0;
    GRAPHICS_INIT;
    DISPLAY_INIT(3,CONTROL,ERROR);
    CLEAR_DISPLAY;
    SET_LINE_STYLE(1);
```

```
  SET_CHAR_SIZE(0.065,0.085);
CORRECT:=FALSE;

   ( GETS USER INPUT FOR THE TESTCLOCK FREQUENCY )

  REPEAT
    WRITELN;
    WRITELN;
    REPEAT
      WRITELN('INPUT THE DESIRED TESTCLOCK
               FREQUENCY IN Hz');
      WRITELN;
      WRITE('>>');
      READLN(TSTFRQ);
      WRITELN;
      IF TSTFRQ<>0 THEN
                     BEGIN
                       R1:=256-(MASTERCLK/TSTFRQ);
                       I1:=TRUNC(R1);
                       I2:=I1+1;
                       IF (I1>155) AND (I2<256)
                          THEN CORRECT:=TRUE
                    ELSE WRITELN('VALUE IS OUT
                         OF ACCEPTED RANGE.');
                      END;
      UNTIL CORRECT;

(CALCULATES FREQUENCY INTEGER FOR OUTPUT FILE)

      WRITELN;WRITELN;
      R2:=MASTERCLK/(256-I1);
      I3:=TRUNC(R2);
      R2:=MASTERCLK/(256-I2);
      I4:=TRUNC(R2);

   ( OBTAINS USER CHOICE OF FREQUENCY WHEN CHOSEN
    FREQUENCY IS UNOBTAINABLE )

  IF (TSTFRQ<>I3) AND (TSTFRQ<>I4) THEN
    BEGIN
      REPEAT
        WRITELN(TSTFRQ:8,'Hz CANNOT BE OBTAINED.');
        WRITELN;
        WRITELN('THE TWO CHOICES
             IN THAT RANGE ARE:');
        WRITELN('1-':2,I3:8,'Hz');
        WRITELN('2-':2,I4:8,'Hz');
        WRITELN;
        WRITELN('INPUT THE NUMBER OF YOUR CHOICE.');
        WRITELN;
```

```
          WRITE('>>':2);
          READLN(I5);
        UNTIL (I5=1) OR (I5=2);

          (GETS FILE NUMBER FOR TEST FREQUENCY)

        IF I5=1 THEN
                  BEGIN
                    TSTNUM:=I1;
                    TSTFRQ:=I3;
                  END
                ELSE  IF I5=2 THEN
                  BEGIN
                    TSTNUM:=I2;
                    TSTFRQ:=I4;
                  END;
     END;

     IF TSTNUM=0 THEN
                  IF TSTFRQ=I3 THEN TSTNUM:=I1
                               ELSE TSTNUM:=I2;
     WRITELN(CHR(12));
     I1:=1;

       ( CALLS THE PROGRAMABLE CLOCK PROCEDURE )

     WHILE I1<7 DO
       BEGIN
         PCLOCK(I1,MLISTI,MLISTR);
         WRITELN(CHR(12));
         I1:=I1+1;
       END;

               ( DRAWS ALL CLOCKS ON THE SCREEN )

   DRAW_ALL(MLISTR,MLISTI);
   SKIPPAGE;
   WRITELN('PRESS <A> TO ACCEPT,
      PRESS <R> TO REDEFINE ALL CLOCKS.');
   REPEAT
     READ(SCH);
   UNTIL (SCH='A') OR (SCH='R');
   IF SCH='R' THEN CLEAR_DISPLAY;
 UNTIL SCH='A';
 SKIPPAGE;
 WRITELN('DO YOU WANT A PLOT OF THE CLOCKS
                              <Y>ES <N>O?');
 WRITE('>>');
 READLN(S1);
```

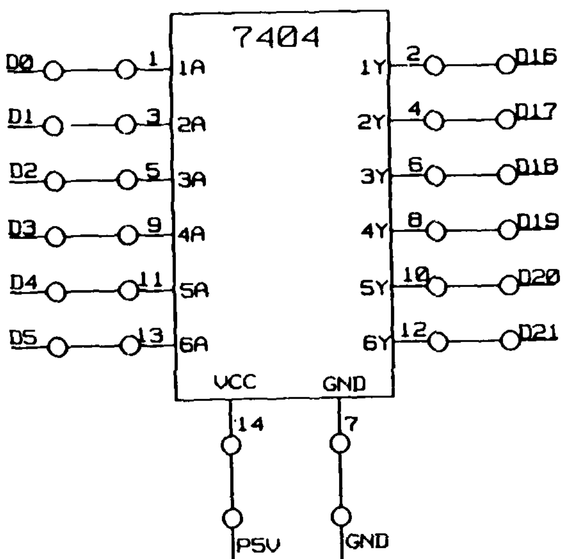
```
(SENDS A PLOT TO THE PLOTTER)

IF S1='Y' THEN
            BEGIN
              SKIPPAGE;
              REPEAT
                ESCAP:=0;
                WRITE('TURN ON PLOTTER,
                       POSITION PAPER, ':33);
                WRITE('THEN PRESS <ENTER>.');
                READLN(S1);
                TRY
                  DISPLAY_INIT(705,CONTROL,ERROR);
                  DRAW_ALL(MLISTR,MLISTI);
                  MOVE(-1.0,1.0);
                  LINE(1.0,1.0);
                  LINE(1.0,-1.0);
                  LINE(-1.0,-1.0);
                  LINE(-1.0,1.0)
                RECOVER
                IF ESCAPECODE = -27 THEN
                    BEGIN
                      SKIPPAGE;
                      ESCAP:=-27;
                      WRITE('THE PLOTTER
                           IS NOT ON. ':23);
                    END;
              UNTIL ESCAP=0;
              DISPLAY_INIT(3,CONTROL,ERROR);
            END;

CLEAR_DISPLAY;
WRITELN(CHR(12));

(GETS THE NUMBER FOR EACH CLOCK TO GO INTO THE FILE)

  FOR I1:=1 TO 6 DO
    BEGIN
      I2:=MLISTI[1,I1];
      I3:=MLISTI[2,I1];
      I2:=TRUNC(I2/50)-3;
      I3:=TRUNC(I3/50)-3;
      IF MLISTI[3,I1] = 2 THEN
                          BEGIN
                            I2:=I2+128;


                            I3:=I3+128;
                          END;
      VALUE1[I1+1]:=INT_TO_HEXSTRNG(I2*256+I3);
```

```
    END;
   VALUE1[1]:=INT_TO_HEXSTRNG(TSTNUM);
  END;                          ( PROCEDURE CLOCKS )


       ( PROCEDURE DOWNLOADS A DATA FILE TO THE
         TESTER AND THEN UPLOADS DATA FROM THE
         TESTER TO A RESULTS FILE )

PROCEDURE TRANSFR;

VAR
     D_ARRY   : PTRD_A;
     ZIPLOK   : S;
     I,J,
     CNTA,
     DATA1,
     DATA2,
     W,B       : INTEGER;

BEGIN                                 ( TRANSFR )
    OOPS:=TRUE;
    WRITELN;
    WRITELN('TRANSFER TO FUNCTIONAL TESTER
                          IS IN PROGRESS.');
    WRITELN;
    RESET(F,DATAFILE);
      TRY
        DATAGROUPS:=F^;
        GET(F);
        DATAWORDS:=F^;
        GET(F);
        STEMP:=INT_TO_BOOLS(F^);
        GET(F);
        FOR W:=1 TO 8 DO
          BEGIN
            IF RIGHTMOST(STEMP) = '1'
               THEN IO_RESVAR[W]:=TRUE
               ELSE IO_RESVAR[W]:=FALSE;
             STRDELETE(STEMP,9-W,1);
           END;
        IORESET(12);
        FOR B:=1 TO 4 DO
          BEGIN
            FOR W:=1 TO 2048 DO
              BEGIN
                IF (B > DATAGROUPS) OR (W > DATAWORDS)
                   THEN
                     WRITEWORD(12,65535)
                     ELSE
```

# 7404 INVERTER TEST RESULTS

| | | | |
|---|---|---|---|
| XX | 0H | * | 3FH |
| XX | 1H | * | 3EH |
| XX | 2H | * | 3DH |
| XX | 3H | * | 3CH |
| XX | 4H | * | 3BH |
| XX | 5H | * | 3AH |
| XX | 6H | * | 39H |
| XX | 7H | * | 38H |
| XX | 8H | * | 37H |
| XX | 9H | * | 36H |
| XX | AH | * | 35H |
| XX | BH | * | 34H |
| XX | CH | * | 33H |
| XX | DH | * | 32H |
| XX | EH | * | 31H |
| XX | FH | * | 30H |
| XX | 10H | * | 2FH |
| XX | 11H | * | 2EH |
| XX | 12H | * | 2DH |
| XX | 13H | * | 2CH |
| XX | 14H | * | 2BH |
| XX | 15H | * | 2AH |
| XX | 16H | * | 29H |
| XX | 17H | * | 28H |
| XX | 18H | * | 27H |
| XX | 19H | * | 26H |
| XX | 1AH | * | 25H |
| XX | 1BH | * | 24H |
| XX | 1CH | * | 23H |
| XX | 1DH | * | 22H |
| XX | 1EH | * | 21H |
| XX | 1FH | * | 20H |
| XX | 20H | * | 1FH |
| XX | 21H | * | 1EH |
| XX | 22H | * | 1DH |
| XX | 23H | * | 1CH |
| XX | 24H | * | 1BH |
| XX | 25H | * | 1AH |
| XX | 26H | * | 19H |
| XX | 27H | * | 18H |
| XX | 28H | * | 17H |
| XX | 29H | * | 16H |
| XX | 2AH | * | 15H |
| XX | 2BH | * | 14H |
| XX | 2CH | * | 13H |
| XX | 2DH | * | 12H |

# 7404 HEX INVERTER TEST CONNECTIONS



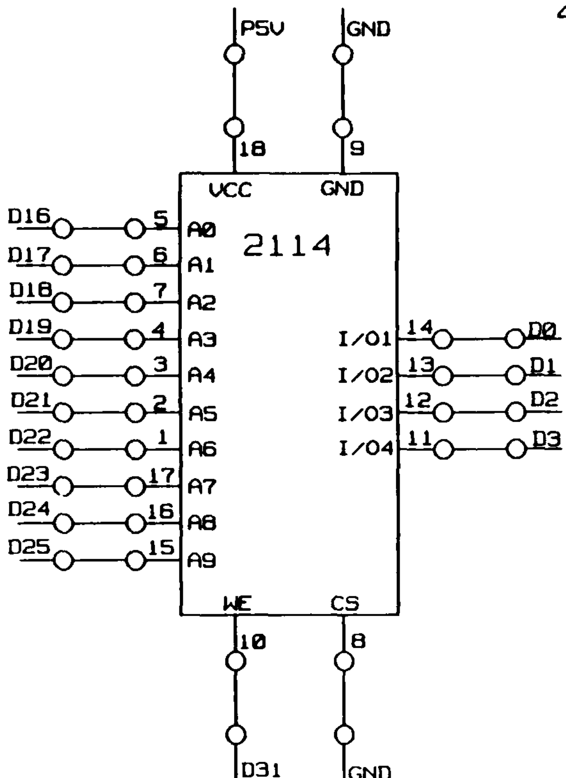

| TITLE: 7404 INVERTER TEST | DATE: October 25, 198 |
|---|---|
| ENGINEER: H. D. Robbins | PAGE: 1 |

# 2114 RAM TEST RESULTS

| | | | | | |
|---|---|---|---|---|---|
| 0H | 0H | 0H | AH | * | XX |
| 0H | 1H | 0H | 5H | * | XX |
| 0H | 2H | 0H | 0H | * | XX |
| 0H | 3H | 0H | 1H | * | XX |
| 0H | 4H | 0H | 2H | * | XX |
| 0H | 5H | 0H | 3H | * | XX |
| 0H | 6H | 0H | 4H | * | XX |
| 0H | 7H | 0H | 5H | * | XX |
| 0H | 8H | 0H | 6H | * | XX |
| 0H | 9H | 0H | 7H | * | XX |
| 0H | AH | 0H | 8H | * | XX |
| 0H | BH | 0H | 9H | * | XX |
| 0H | CH | 0H | AH | * | XX |
| 0H | DH | 0H | BH | * | XX |
| 0H | EH | 0H | CH | * | XX |
| 0H | FH | 0H | DH | * | XX |
| 0H | 10H | 0H | EH | * | XX |
| 0H | 11H | 0H | FH | * | XX |
| 0H | 12H | 0H | AH | * | XX |
| 0H | 13H | 0H | 5H | * | XX |
| 80H | 0H | 0H | XX | * | AH |
| 80H | 1H | 0H | XX | * | 5H |
| 80H | 2H | 0H | XX | * | 0H |
| 80H | 3H | 0H | XX | * | 1H |
| 80H | 4H | 0H | XX | * | 2H |
| 80H | 5H | 0H | XX | * | 3H |
| 80H | 6H | 0H | XX | * | 4H |
| 80H | 7H | 0H | XX | * | 5H |
| 80H | 8H | 0H | XX | * | 6H |
| 80H | 9H | 0H | XX | * | 7H |
| 80H | AH | 0H | XX | * | 8H |
| 80H | BH | 0H | XX | * | 9H |
| 80H | CH | 0H | XX | * | AH |
| 80H | DH | 0H | XX | * | BH |
| 80H | EH | 0H | XX | * | CH |
| 80H | FH | 0H | XX | * | DH |
| 80H | 10H | 0H | XX | * | EH |
| 80H | 11H | 0H | XX | * | FH |
| 80H | 12H | 0H | XX | * | AH |
| 80H | 13H | 0H | XX | * | 5H |

# 2114 1024 X 4BIT RAM

## TEST CONNECTIONS




| TITLE: 2114 RAM TEST | DATE: October 28, 1984 |
| --- | --- |
| ENGINEER: H. D. Robbins | PAGE: 1 |

# APPENDIX E

## TEST RESULTS

```
    READLN(STEMP);
   END;
  GET(P);
  UNTIL EOF(P);
 END;
END;
```

```
WRITELN('                          <P>RINTER');
WRITELN('                          <S>CREEN');
WRITELN('                          <N>O LISTING');
WRITELN;
WRITE('>>');
READLN(STEMP);
IF STEMP = 'P' THEN
 BEGIN
  WRITESTRINGLN(9,' ');
  WRITESTRINGLN(9,' ');
  WRITESTRING(9,'
    DATA                    ');
  WRITESTRINGLN(9,'
    RESULT');
  WRITESTRINGLN(9,' ');
  WRITESTRINGLN(9,' ');
  WRITESTRING(9,'-------------------------
               --------------');
  WRITESTRINGLN(9,'-----------------------
                 ---------------');
  WRITESTRINGLN(9,' ');
  RESET(P,RESULT);
   REPEAT
    CHR_ARRY:=P^;
    CHR_ARRY[79]:=' ';
    STRMOVE(79,CHR_ARRY,1,STEMP,1);
    WRITESTRINGLN(9,STEMP);
    GET(P);
   UNTIL EOF(P);
  END;
IF STEMP = 'S' THEN
 BEGIN
  WRITELN(CHR(12));
  CNT_VAR:=5;
  WRITELN(' ':18,'DATA':4,' ':35,'RESULT');
  WRITE('------------------------------
        ---------':40);
  WRITELN('----------------------------
            -----------':40);
  RESET(P,RESULT);
  REPEAT
   CHR_ARRY:=P^;
   CHR_ARRY[79]:=' ';
   WRITELN(CHR_ARRY);
   CNT_VAR:=CNT_VAR+1;
   IF CNT_VAR >20 THEN
    BEGIN
     CNT_VAR:=1;
     WRITELN;
     WRITELN(' ':28,'PRESS <ENTER> FOR MORE');
```

```
 WRITELN;
 TRANSFR;
END
ELSE BEGIN
 OOPS:=TRUE;
 WRITELN('INPUT THE NAME OF YOUR DATA FILE');
 WRITE('>>');
 READLN(TESTFILE);
 DATAFILE:=TESTFILE;
 CNT_VAR:=STRPOS('.',TESTFILE);
 IF CNT_VAR>0 THEN
    STRDELETE(TESTFILE,CNT_VAR,
    (STRLEN(TESTFILE)-CNT_VAR);
 WRITELN;
 WRITELN('DO YOU WANT TO USE THE SAME
          TEST CLOCKS [Y/N] ?');
 REPEAT
  WRITE('>>');
  READLN(STEMP);
 UNTIL (STEMP = 'Y') OR (STEMP = 'N');
 IF STEMP = 'N'
  THEN
   BEGIN
    OPEN(F,DATAFILE);
    GET(F);
    DATAGROUPS:=2*F^;
    GET(F);
    DATAWORDS:=F^;
    GET(F);
    CNT_VAR:=(DATAWORDS*
     (TRUNC(DATAGROUPS/2) + 1)) + 4;
    SEEK(F,CNT_VAR);
    CLOCKS(CK_ARRAY);
    FOR CNT_VAR:= 1 TO 7 DO
     BEGIN
      F^:=INTGD(CK_ARRAY[CNT_VAR]);
      PUT(F);
     END;
    CLOSE(F,'CRUNCH');
    WRITELN;
    WRITELN;
   END;
  TRANSFR;
 END;
IF OOPS THEN
 BEGIN
   WRITELN;
   WRITELN;
   WRITELN('WOULD YOU LIKE A LISTING OF THE
                              RESULT FILE TO:');
```

```
WHILE OOPS DO
 BEGIN
  DIR_WRD_IN:=FALSE;
  DATAFILE:=TESTFILE;
  STRAPPEND(DATAFILE,'.D');
  REWRITE(F,DATAFILE);
  F^:=ROUND(DATAGROUPS/2);
  PUT(F);
  F^:=DATAWORDS;
  PUT(F);
  WRITELN;
  WRITELN('PLEASE ENTER AN 8-BIT BINARY NUMBER
           WITH A "1" FOR ');
  WRITE('EACH 8-BIT DATA GROUP WHICH
         CONTAINS TEST RESULT ');
  WRITELN('DATA.');
  WRITELN;
  WRITE('>>');
  READLN(STEMP);
  IF RIGHTMOST(STEMP)<>'B'
     THEN STEMP:=STEMP+'B';
  F^:=INTGD(STEMP);
  PUT(F);
  WRITELN('ENTER THE VALUES OF THE DATA
           WORDS(16-bit).');
  CREATE(DATAWORDS,DATAGROUPS);
END;
OOPS:=TRUE;
WHILE OOPS DO
 BEGIN
  WRITELN;
  DIR_WRD_IN:=TRUE;
  WRITE('PLEASE ENTER THE "DIRECTION" WORDS,');
  WRITELN('WITH THE LEAST SIGNIFICANT ');
  WRITELN('BIT AS THE DIRECTION FOR DATA GROUP
           NUMBER 1');
  CREATE(DATAWORDS,1);
END;
CLOCKS(CK_ARRAY);
FOR CNT_VAR:= 1 TO 7 DO
 BEGIN
  F^:=INTGD(CK_ARRAY[CNT_VAR]);
  PUT(F);
 END;
CLOSE(F,'CRUNCH');
WRITELN;
WRITELN;
WRITELN('FILE "':6,DATAFILE,'" CREATED':9);
WRITELN;
WRITELN;
```

```
( ************************************************** )
( *                                                * )
( **************     MAIN PROGRAM     ************** )
( *                                                * )
( ************************************************** )


BEGIN                                  ( MAIN PROGRAM )
  WRITELN;
  WRITELN;
  WRITE(' ':20,'WELCOME TO THE
       MSU FUNCTIONAL TESTER':37);
  FOR CNT_VAR:=1 TO 10 DO WRITELN;
  WRITELN;
  REPEAT
    WRITELN('DOES THE TEST DATA FILE ALLREADY EXIST?
              ...[Y/N]');
    WRITE('>>':2);
    READLN(STEMP);
  UNTIL (STEMP = 'Y') OR (STEMP = 'N');
  WRITELN;
  IF STEMP = 'N'
     THEN BEGIN
      WRITELN;
      WRITELN('ENTER THE NAME YOU WANT FOR YOUR
                DATA FILE');
      WRITE('>>');
      READLN(TESTFILE);
      WRITELN;
      REPEAT
       WRITE('HOW MANY GROUPS OF EIGHT(8) DATA
              LINES WILL ');
       WRITELN('BE USED FOR OUTPUT?');
       WRITE('>>':2);
       READLN(DATAGROUPS);
       WRITELN;
       IF DATAGROUPS>8 THEN
          WRITELN('INVALID NUMBER');
      UNTIL DATAGROUPS<=8;
      WRITELN;
      REPEAT
       WRITELN('HOW MANY DATA WORDS WILL BE USED?');
       WRITE('>>':2);
       READLN(DATAWORDS);
       WRITELN;
       IF DATAWORDS>2048 THEN
          WRITELN('INVALID NUMBER');
      UNTIL DATAWORDS<=2048;
      OOPS:=TRUE;
```

```
                    INT_TO_HEXSTRNG(D_ARRY^[W,B])+'  ';
                         END;
                       B:=B-1;
                   END;
              I:=TRUNC((40-STRLEN(STEMP))/2);
              FOR J:=1 TO I DO
                  BEGIN
                    STEMP:=' '+STEMP;
                  END;
              IF NOT ODD(STRLEN(STEMP)) THEN I:=I-1;
              FOR J:=1 TO I DO
              BEGIN
                  STEMP:=STEMP+' ';
              END;
              STEMP1:=T_F^;
              I:=TRUNC((40-STRLEN(STEMP1))/2);
              FOR J:=1 TO I DO
                  BEGIN
                    STEMP1:=' '+STEMP1;
                  END;
              STEMP:=STEMP+'*'+STEMP1;
              GET(T_F);
              STRMOVE(STRLEN(STEMP),STEMP,1,CHR_ARRY,1);
              FOR J:=1 TO 79 DO
                BEGIN
                  IF CHR_ARRY[J]=CHR(0)
                     THEN CHR_ARRY[J]:=' ';
                END;
              CHR_ARRY[79]:=CHR(13);
              P^:=CHR_ARRY;
              PUT(P);
              END;
            CLOSE(T_F,'PURGE');
            CLOSE(P,'CRUNCH');
            END;
            WRITELN;
            WRITELN('DATA LOADED');
        RECOVER
          IF ESCAPECODE = IOESCAPECODE THEN
                  BEGIN
                    WRITELN;
                    WRITELN;
                    WRITELN('ERROR IN LOADING DATA
                           FROM FUNCTIONAL TESTER');
                    OOPS := FALSE;
                  END
              ELSE ESCAPE(ESCAPECODE);
        END;                                    ( IF OOPS DO )
    END;                                        ( END TRANSFR )
```

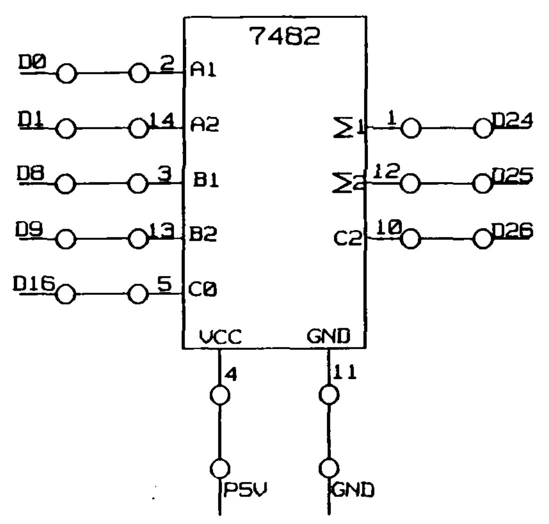
```
FOR W:=1 TO DATAWORDS DO
  BEGIN
    B:=8;
    STEMP:='';
    FOR CNTA:=1 TO 8 DO
      BEGIN
        IF D_ARRY^[W,B]<16
        THEN STEMP:=STEMP+' ';
        IF D_ARRY^[W,B]<>200000
         THEN IF D_ARRY^[W,B] <>100000
          THEN STEMP:=STEMP +
          INT_TO_HEXSTRNG(D_ARRY^[W,B])+'  '
          ELSE STEMP:=STEMP+' XX'+'  ';
        B:=B-1;
      END;
    T_F^:=STEMP;
    PUT(T_F);
  END;
OPEN(F,DATAFILE);
SEEK(F,4);
GET(F);
NEW(D_ARRY);
FOR B:=1 TO DATAGROUPS DO
    BEGIN
      FOR W:=1 TO DATAWORDS DO
          BEGIN
            DATA1:=F^;
            GET(F);
            DATA2:=TRUNC(DATA1/256);
            DATA1:=DATA1-DATA2*256;
            D_ARRY^[W,B*2-1]:=DATA1;
            D_ARRY^[W,B*2]:=DATA2;
          END;
    END;
RESET(T_F);
REWRITE(P,RESULT);
FOR W:=1 TO DATAWORDS DO
    BEGIN
      BOOL(F^,DIRVAR);
      GET(F);
      STEMP:='';
      FOR CNTA:=1 TO DATAGROUPS*2 DO
          BEGIN
            B:=DATAGROUPS*2-CNTA+1;
            IF DIRVAR[B] THEN
             STEMP:=STEMP+' XX'+'  '
               ELSE BEGIN
                  IF D_ARRY^[W,B]-16
                  THEN STEMP:=STEMP+' ';
                  STEMP:=STEMP +
```

```
          BEGIN
            READWORD(12,DATA1);
            SEEK(F,DATAWORDS*DATAGROUPS+4);
            GET(F);
            FOR W:=1 TO 2046 DO
              BEGIN
                IF W > DATAWORDS
                   THEN READWORD(12,DATA1)
                   ELSE
                    BEGIN
                      READWORD(12,DATA1);
                      IF DATA1<0
                        THEN DATA1:=DATA1+65536;
                      DATA2:=TRUNC(DATA1/256);
                      DATA1:=DATA1-DATA2*256;
                      D_ARRY^[W,B*2-1]:=200000;
                      D_ARRY^[W,B*2]:=200000;
                      IF IO_RESVAR[B*2-1] OR
                         IO_RESVAR[B*2]
                         THEN BEGIN
                          BOOL(F^,DIRVAR);
                          GET(F);
                          IF IO_RESVAR[B*2-1]
                          THEN IF DIRVAR[B*2-1]
                          THEN
                          D_ARRY^[W,B*2-1]:=DATA1
                          ELSE
                          D_ARRY^[W,B*2-1]:=100000;
                          IF IO_RESVAR[B*2]
                          THEN IF DIRVAR[B*2]
                          THEN
                          D_ARRY^[W,B*2]:=DATA2
                          ELSE
                          D_ARRY^[W,B*2]:=100000;
                              END;
                   END;
           END;                (* END W LOOP *)
            READWORD(12,DATA1);
            IF IO_RESVAR[B*2-1]
               THEN BEGIN
                D_ARRY^[2047,B*2-1]:=100000;
                D_ARRY^[2048,B*2-1]:=100000;
                    END;
            IF IO_RESVAR[B*2]
               THEN BEGIN
                D_ARRY^[2047,B*2]:=100000;
                D_ARRY^[2048,B*2]:=100000;
                    END;
         END;                (* END B LOOP *)
     REWRITE(T_F);
```

```
                  BEGIN
                    WRITEWORD(12,F^);
                    GET(F);
                  END;
          END;
      END;
    FOR W:=1 TO 2048 DO
      BEGIN
       IF W>DATAWORDS THEN WRITEWORD(12,65535)
                       ELSE
                         BEGIN
                           WRITEWORD(12,F^);
                           GET(F);
                         END;
       END;
    REPEAT
      DATA1:=F^;
      WRITEWORD(12,DATA1);
      GET(F);
    UNTIL EOF(F);
    WRITELN('TRANSFER TO FUNCTIONAL
                TESTER IS COMPLETE');
    RECOVER
      IF ESCAPECODE = IOESCAPECODE
        THEN BEGIN
         WRITELN;
         WRITELN(IOERROR_MESSAGE(IOE_RESULT));
         WRITELN;
         WRITELN('PLEASE CORRECT THE ERROR
                    AND RERUN THE PROGRAM');
         OOPS:=FALSE;
         WRITELN;
             END;

  IF OOPS
    THEN BEGIN
      TRY
        RESULT:=TESTFILE;
        STRAPPEND(RESULT,'.R');
        OPEN(F,DATAFILE);
        WRITELN;
        WRITELN('LOADING DATA FROM FUNCTIONAL
                       TESTER TO FILE ',RESULT);
        NEW(D_ARRY);
        FOR B:=1 TO 8 DO
          BEGIN
            D_ARRY^[2047,B]:=200000;
            D_ARRY^[2048,B]:=200000;
          END;
        FOR B:=1 TO 4 DO
```

```
XX  2EH        *        11H
XX  2FH        *        10H
XX  30H        *        FH
XX  31H        *        EH
XX  32H        *        DH
XX  33H        *        CH
XX  34H        *        BH
XX  35H        *        AH
XX  36H        *        9H
XX  37H        *        8H
XX  38H        *        7H
XX  39H        *        6H
XX  3AH        *        5H
XX  3BH        *        4H
XX  3CH        *        3H
XX  3DH        *        2H
XX  3EH        *        1H
XX  3FH        *        0H
```

# 7482 2-BIT FULL ADDER
## TEST CONNECTIONS




| TITLE: 7482 ADDER TEST | DATE: October 25, 19[illegible]4 |
| --- | --- |
| ENGINEER: H. D. Robbins | PAGE: 1 |

7482 2-BIT BINARY FULL ADDER

| | | | | | |
|---|---|---|---|---|---|
| XX | OH | OH | OH | * | OH |
| XX | OH | OH | 1H | * | 1H |
| XX | OH | OH | 2H | * | 2H |
| XX | OH | OH | 3H | * | 3H |
| XX | OH | 1H | OH | * | 1H |
| XX | OH | 1H | 1H | * | 2H |
| XX | OH | 1H | 2H | * | 3H |
| XX | OH | 1H | 3H | * | 4H |
| XX | OH | 2H | OH | * | 2H |
| XX | OH | 2H | 1H | * | 3H |
| XX | OH | 2H | 2H | * | 4H |
| XX | OH | 2H | 3H | * | 5H |
| XX | OH | 3H | OH | * | 3H |
| XX | OH | 3H | 1H | * | 4H |
| XX | OH | 3H | 2H | * | 5H |
| XX | OH | 3H | 3H | * | 6H |
| XX | 1H | OH | OH | * | 1H |
| XX | 1H | OH | 1H | * | 2H |
| XX | 1H | OH | 2H | * | 3H |
| XX | 1H | OH | 3H | * | 4H |
| XX | 1H | 1H | OH | * | 2H |
| XX | 1H | 1H | 1H | * | 3H |
| XX | 1H | 1H | 2H | * | 4H |
| XX | 1H | 1H | 3H | * | 5H |
| XX | 1H | 2H | OH | * | 3H |
| XX | 1H | 2H | 1H | * | 4H |
| XX | 1H | 2H | 2H | * | 5H |
| XX | 1H | 2H | 3H | * | 6H |
| XX | 1H | 3H | OH | * | 4H |
| XX | 1H | 3H | 1H | * | 5H |
| XX | 1H | 3H | 2H | * | 6H |
| XX | 1H | 3H | 3H | * | 7H |

# REFERENCES

# References

Atlas, Joseph & Nielsen, Robert High-Speed Digital Test Capability For Emerging Technology . IEEE AUTOTESTCON '83 Transactions, 1983.

Bipolar Microprocessor Logic and Interface Data Book . Advanced Micro Devices , Inc., 1983.

CMOS Digital Data Book . Harris Corporation, 1984.

iAPX 86/88, 186/188 User's Manual Programmer's Reference . Intel Corporation, 1983.

The 8086 Family User's Manual . Intel Corporation, 1980.

8086 Assembly Language Programming Manual . Intel Corporation, 1984.

MCS-86 User's Manual . Intel Corporation, 1984.

SDK-86 MCS-86 System Design Kit Assembly Manual . Intel Corporation, 1984.

O'Keefe, Rob, Gebhard, Dick, & O'Connell, Tim Test Patterns For Static RAMs . IEEE 1979 Test Conference, LSI & Boards, 1979.

Puri, Prem The Functional Tester:An Aid To The Designer . IEEE 1980 Test Conference, 1980.

Robinson, Arthur L. One Billion Transistors on a Chip? . Science, Vol. 223, No. 4631, 6 January 1984.

Short, Kenneth L. Microprocessors and Programmed Logic . Englewood Cliff, N.J.: Prentice-Hall, Inc., 1981.

The TTL Data Book for Design Engineers . Texas Instruments Incorporated, 1973.

END

FILMED

10-85

DTIC