A LOW COST, FLEXIBLE, AUTOMATIC-TESTING-SYSTEM

FOR DIGITAL CIRCUITS

Michael Kerrigan Hawes

A Project Report Submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in partial fulfillment of the requirements for the Degree of Master of Science in Engineering

Johannesburg 1985

DECLARATION

I declare that this Project Report is my own, unaided work. It is submitted in partial fulfilment of the requirements for the degree of Master of Science in Engineering at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

K Haug

(Name of candidate)

17th day of JUNE 1985

ABSTRACT

The aim of the project discussed in this Report was to develop the hardware structure of an Automatic-Test-System for use as a teaching facility in the area of Digital-Circuit-Testing at the University. Commercial systems, while obviously suitable are very expensive and would not be cost-effective in an environment where the diversity of circuits to be tested is large, but the actual volumes are small.

This Report discusses the various current techniques of Digital-Circuit-Testing, and then motivates the system developed. The system proposed can act as a Dynamic-Functional-Tester which drives and senses patterns at the operational speed of digital circuits, or as a Logic-Analyser-System which senses patterns, stopping when a Trigger-Pattern is detected. The Logic-Analyser function is included as it was relatively simple to implement, and Logic-Analysers have proved useful in general testing operation.

The Test-Hardware has a number of external connections which can function as either drivers, or sensors, or both. These connecting pins drive the Test-Pattern and then sense the Returned-Pattern. As drivers the pins are placed by the controller into either a high or low state, and when used as sensors, the pins are set into Tri-State condition. The input to the sensors can be compared against a single reference value for detecting Bi-state, or against dual references for detecting Tri-State. Behind each pin is dedicated Pattern-Output-Memory and Pattern-Returned-Memory.

The Test-Hardware was designed to be Multibus Compatible and is controlled by an 8086 Single-Board-Computer. This computer in turn is connected via a RS-232C serial link to a Main-Control-Computer. Up to the point that it was developed, the Test-Hardware worked successfully.

ACKNOWLEDGEMENTS

I am grateful to my supervisor Prof. M. G. Rodd for the support and encouragement he has offered me during the course of this degree. Thanks also to my colleagues and the technical staff who have assisted me in the course of my work. Thanks are also due to the University of the Witwatersrand and the CSIR for the financial support which made this project possible.

CONTENTS

LIST OF APPENDICES

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

Actual-Response	–	The response actually obtained from the Device-Under-Test when applying the *Input-Stimulus*.
ATE	–	Automatic-Test-Equipment.
Device-Under-Test	–	The *Circuit or Device* that is to be tested by the Automatic-Test-*Equipment*.
Expected-Response	–	The correct response expected from the *Device*-Under-Test when applying the Input-Stimulus.
Functional-Tester	–	An *Automatic-Tester* that tests the Device-Under-Test from an external *functional* point.
In-Circuit-Tester	–	An Automatic-Tester that tests the Device-*Under*-Test from an internal view, by testing the *individual circuit* components directly.
Input-Stimulus	–	The pattern applied to the Device-Under-Test during the testing operation.
Known-Good-*Board*	–	A version of the Device-Under-Test that is Known *to function* correctly.
Logic-Simulation	–	A software model *of the Device-Under*-Test that models the functional operation of *the circuit*.
Test-Pattern	–	The pattern applied to the Device-Under-Test as the *Input*-Stimulus during the testing operation.

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In this Chapter Automatic-Test-Equipment in general will be discussed, followed by the reason a Local-Test-System was developed for the University Environment. The aims of the Local-System will be mentioned followed by a brief overview of the hardware developed for this project.

1.2 WHY USE AUTOMATIC-TEST-EQUIPMENT

Automatic-Test-Equipment basically describes any equipment used for testing a device automatically, but this project addresses itself to Electronic Equipment used in the automatic testing of Electrical Circuits. The Test-Equipment is automated to make testing the circuit easier, quicker and ultimately more cost-effective.

Traditionally most circuits were tested with a multimeter and oscilloscope. To make measurement easier methods using a number of special purpose instruments such as the Logic-Analyser, and the Logic Probe used in digital circuits, were developed. These methods still require significant manual input, and are very time consuming. With these methods testing was generally confined to when the system was completely assembled and failed to operate correctly. The earlier a fault is detected the cheaper it is to repair. A system failure in the field,

after installation, can be very expensive to repair, especially if the system is critical, and production time is lost due to the failure.

One method used to get around the problem of lost production time, is to modularise the system. A failure in the field is fixed by replacing the faulty module. The module is taken back to the service workshop and repaired in due course. This method works effectively, but can be expensive in terms of replacement modules that need to be stocked. If the modules in question are expensive, then repairing the modules is necessary, and this can be slow and expensive if the manual methods are used.

The introduction of Integrated Circuits has made it possible to include more functions in a much smaller volume. This trend goes contrary to the modularity principle, as more and more functions are packed into a single module. Modules become complex, expensive and more difficult to test. This is evident in the digital electronics field, particularly with the present use of VLSI technology, and imminent introduction of ULSI circuits.

As systems become more complex, the cost of repairing a faulty system at installation time increases. It is much cheaper to replace faulty modules before they are integrated into the system. A method of testing modules is therefore desirable. One method used, is to build a specific Test-Circuit for the module in question, which is expensive as each module needs its own Test-Circuit. The Test-Circuit is usually more complex than the module it is designed to test.

It is therefore extremely desirable to have a method to automatically test a variety of modules. Flexibility is needed in order to test a number of different modules. The Automatic-Test-Equipment described herein was designed to fill this requirement.

1.2 WHAT IS AUTOMATIC-TEST-EQUIPMENT

Automatic-Test-Equipment generally refers to complex computer controlled equipment used in the automatic testing of various devices. For the purposes of this project the testing of electrical circuits will be discussed. The flexibility requirement establishes the need for, and is met by, computer control. If the Test-Equipment is under computer control then flexibility is achieved by making the Test-System programmable.

1.2.1 ANALOG CIRCUITS / DIGITAL CIRCUITS

Electrical circuits can broadly be divided into Analog and Digital. Automatic-Test-Equipment normally has interfaces dedicated to testing either analog or digital circuits.

When testing Analog Circuits it is necessary to be able to apply and measure voltage, current, power etc, at specific time intervals, depending on the time constants of the Device-Under-Test. Analog circuitry is interfaced to the Test-Equipment by means of analog conditioning circuitry, Analog-to-Digital, and Digital-to-Analog converters. Automatic-Test-Equipment for testing analog circuits can operate up to a maximum speed, normally limited by the conversion time of the Analog to Digital converters. Fast Analog-to-Digital converters are very costly.

With Digital Circuits it is only necessary to apply and measure digital voltages at specific intervals, usually determined by the clock speed of the Device-Under-Test. The interface to the Test-Equipment generally involves an analog interface with Voltage Threshold detection and Voltage Drivers. If the circuit is to be tested at operational speed, then the Test-Equipment needs to function up to this speed. With clocked systems it is sometimes possible to run a number of tests at a slower speeds by slowing the clock, and in this case the Test-Equipment need not operate at the operational speed of the Device-Under-Test.

1.3.2 FUNCTIONAL-TESTERS / IN-CIRCUIT-TESTERS

Automatic-Test-Equipment used for testing populated circuit boards falls broadly into two categories, namely Functional-Testers and In-Circuit-Testers.

Functional-Testers are connected to the external inputs and outputs of the circuit in question, normally at the appropriate edge connector. The circuit is then tested by the application of Test-Patterns and the measurement of the response. A guided probe can be used to apply and measure signals internal to the circuit. In Digital-Testers this guided probe usually takes the form of a Signature-Analyser (Ref. Blyth 1982), which simplifies the serial stream of node transitions.

In-Circuit-Testers function by applying and measuring signals to and from the circuit components directly, rather than externally. (eg to and from the edge connector) One method used is called Back Driving where the Device-Under-Test is connected to the Automatic-Tester via a Bed-Of-Nails and then each component in turn is checked for correct operation and connection to other components. This is achieved by forcing the inputs of a component to a particular state and measuring the outputs. A guided probe in this instance would be used to get detailed information not normally available from the connection via the Bed-Of-Nails. Another form of In-Circuit-Testing is In-Circuit-Emulation (Ref. Popky 1982), used in digital microprocessor based circuits. A Signature-Analyser is often used as a guided probe in this method.

1.4 WHY DEVELOP A LOCAL-TEST-SYSTEM

The field of Automatic-Test-Equipment is very complex with large amounts of money being spent every year on development, by the Industry Leaders in Test-Equipment. It would be ambitious as an individual to compete with companies which have been specialising in Automatic-Test-Equipment for a number of years.

There was a need at the University for an Automatic-Testing-System to be available for demonstration and educational purposes. Commercial Test-Systems, while highly suitable for this purpose, are very expensive and could not be used cost effectively in the University Environment. The University does development work and does not go into the manufacture of large numbers of systems. The diversity of circuits in this environment is large but the actual numbers involved are small. The purchase of an expensive commercial Automatic-Test-System would not be the best use of the limited funds available.

The actual need at the University was for a low cost Automatic-Test-System, which was flexible enough to be used for a number of purposes.

1.5 AIMS OF THE LOCAL-TEST-SYSTEM DEVELOPED

The main criteria used in the development of a Local-Test-System were low cost and flexibility. The idea was to develop as flexible a Test-System as possible while keeping the costs low.

A decision was made to develop a Test-System solely for the testing of Digital Circuits. Testing analog circuits would be far more expensive, as the interface involves high speed Analog to Digital and Digital to Analog Converters. The department is involved in a number of digital projects, and therefore a Tester for digital circuits would be more useful.

In the development of a Local-Test-System it was decided not to use the Bed-of-Nails connection method. The reasons for this is the huge expense and the mechanical complexity. Another disadvantage of this method is that boards tested have to be specially designed with locating holes etc.

In-Circuit-Emulation was discarded as this method is very specific to the micro-processor emulated, and therefore would not be sufficiently flexible. It is necessary to develop a complex interface for each micro-

processor to be emulated. Another disadvantage of In-Circuit-Emulation is that it is only possible to test digital circuits which are controlled by the particular micro-processor emulated.

A Functional-Tester was needed which was not limited by the above constraints. There were three possible Functional-Testers considered, namely High Frequency Pattern Generation, High-Speed-Functional-Testers, and Slow-Speed-Functional-Testers. The High Frequency Pattern Generation method was discarded mainly because of initialisation problems (detailed later) encountered in sequential circuits when using this method. The Slow-Speed-Functional-Testers were discarded as these testers have problems when testing circuits with dynamic components, and speed related problems are not detected. A method of High-Speed-Functional-Testing was chosen for this project.

The Method Chosen is called 'Dynamic-Functional-Testing' and is described briefly by Blyth (1982). This method drives and senses patterns at the operational speed of digital circuits, by having dedicated local memory behind the electronics of the drivers and sensors. The local memory is loaded at the slow I/O rate of the Controlling-Computer. The test is done by a high speed burst when the locally stored Test-Patterns are clocked out, and the Actual-Response is stored in the memory allocated to the monitor pins. Once the test burst is completed the Actual-Response is then read at the slower speed of the Controlling-Computer and analysed. The maximum speed chosen was 16 MHz which should be fast enough for the majority of digital circuits currently in use.

The Tester was also designed to operate as a Logic-Analyser which senses patterns, stopping when a Trigger-Pattern is detected. The Logic-Analyser function was included as it was relatively easy to impliment, and Logic-Analysers have proved useful in general testing operations.

A Signature-Analyser was not developed as the Probe hardware is relatively simple, and the real difficulty lies in applying repetitive patterns to the Device-Under-Test. Signature-Analysis however would be easy to add to the Test-Hardware developed, and with suitable software would be valuable in fault tracing. The hardware developed could be used as the pattern application part of a Signature-Analysis-System.

1.6 OVERVIEW OF THE LOCAL-TEST-SYSTEM DEVELOPED

The Test-System developed consists of three MULTIBUS compatible cards, connected to an Intel 86/12A Single-Board-Computer. The 86/12A in turn is connected via an RS 232 C serial link to the Main-Control-Computer which in this case was an Apple II+ compatible, running FLEX on a 6809 co-processor card. A special serial card was developed due to the limitations of the standard serial card available. The basic functional layout of the Test-System is detailed in Figure 1 and the physical layout is detailed in Figure 2.

The functions of the three MULTIBUS compatible cards are as follows :-

Card 1) Multibus Interface, clock generation and control circuitry.
Card 2) Pattern Input and Output memory, latches and multiplexers.
Card 3) Analog interface circuitry.

The Test-Hardware has dedicated memory assigned to each driver/sensor pin. Each pin has dedicated Pattern-Output-Memory and Pattern-Returned-Memory. The Test-Hardware has various modes of operation so that it can either act as an Automatic-Digital-Tester or as a Logic-Analyser. The various Operational Modes can be described as follows :-

1) Digital-Tester Mode
   Go from the Start to the End of memory driving the Output-Patterns and sensing the Returned-Patterns. This does a single pass of the memory. This mode is the normal mode of operation for testing a digital circuit in burst mode.

2) Logic-Analyser Mode
   Keep Looping by repetitively going from the Start to the End of memory, and repeat until the Trigger-Pattern matches the Sensed-Pattern. Store the address of the trigger occurrence, then continue through half of the memory before stopping. Half the Pattern-Returned-Memory contains what happened before the

trigger and half contains what happened after the trigger. This mode is to make the Test-System operate as a Logic-Analyser.

3) Triggered Digital-Tester Mode
Wait until the Sensed-Pattern matches the Trigger-Pattern then go from the Start to the End of the memory driving from the Pattern-Output-Memory, and inputting into the Pattern-Returned-Memory. This does a single pass of the memory. This mode would be used to test a circuit once a certain state has been reached.

The analog interface board can be designed for specific applications. The present analog interface was designed to be as general purpose as possible. The interface board is connected on one side to the Test-Hardware and on the other side to the Device-Under-Test. The present interface board was designed with the purpose of testing TTL levels and is able to detect Tri-State. There are a number of pins which are both drivers and sensors. When the pin is defined as a sensor then the output driver is placed in Tri-State, and when the pin is defined as a driver then it is driven high or low. While the pin is being driven high or low the actual level is sensed and read into the Pattern-Returned-Memory. At any time during the testing of the Device-Under-Test the pin can be changed from being a driver to being a sensor or vice versa. Behind each pin is dedicated Pattern-Output-Memory and Pattern-Returned-Memory.

The different Configuration Modes can be summarised in the following Table :-

	Memory/Pin	No. Pins	Max Speed	Detect State
1)	1K	32	8 MHz	Bi -State
2)	1K	16	8 MHz	Tri-State
3)	2K	16	16 MHz	Bi -State
4)	2K	8	16 MHz	Tri-State

Table 1 : Different Configuration Modes

Modes 3 and 4 use memory multiplexing so that the speed and the amount of memory are doubled but the number of driver/sensor pins is halved. The difference between detecting Tri-State and Bi-State is that in the Tri-State mode dual thresholds are used and an input line is needed for each threshold. This means that the number of lines in Tri-State detection is half that of Bi-State detection.

The Test-System developed can also be used as a Logic-Analyser as it is able to go into a looping mode and then trigger on a certain pattern being matched. If the Test-System were to be used as a normal Logic-Analyser then all pins would be defined only as sensors, and driven into tri-state.

The structure of the Test-System is detailed in the following Figure :-

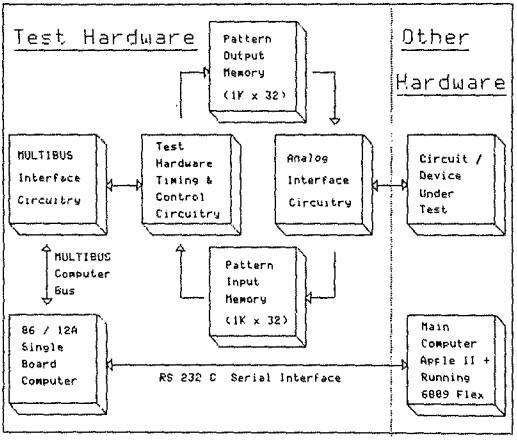


Figure 1 : Basic Block Diagram

The aim of this project was to develop both the hardware structure of an Automatic-Digital-Test-System, and the basic software needed to drive the Test-System and perform certain tests. The test listings are contained in Appendix N and Appendix O. A full Automatic-Test-System can be developed by the addition of suitable software. This software however is fairly complex and would require a significant amount of time and effort to develop. The hardware can also be used as a Logic-Analyser with the addition of relatively simple software.

The Test-Hardware consisted of four MULTIBUS compatible boards, one of which was an Intel 86/12A Single-Board-Computer. The other three boards were developed for the purposes of this project. The basic physical layout can be seen in the following Figure :-

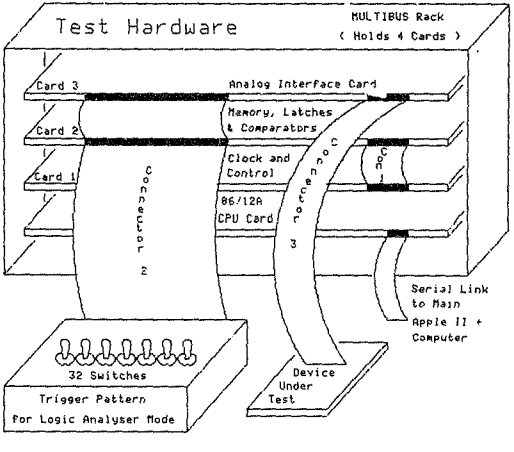


Figure 2 : Basic Physical Layout

1.7 CONCLUSION

The hardware developed for this project had the main aim of being a low cost, flexible, Automatic-Test-System for digital circuits. The choice of a Dynamic-Functional-Tester with a Logic-Analyser mode, was considered appropriate to fit these requirements.

The 86/12A Single-Board-Computer was chosen as a fast local computer was needed to control the Test-Hardware. The availability of the 86/12A Single-Board-Computer and an 8086 Emulator at the University, were important factors contributing to this choice. A personal Computer was connected to the 86/12A via the serial link to act as the main Data-Base computer in the Test-System. The availability of this computer was the main contributing factor in this particular choice.

The need for a Locally Developed Test-System at the University for demonstration and educational purposes was the requirement behind this project. The typical University environment is one of a large diversity of circuits, produced in small numbers. Commercial systems could therefore not be cost-effective in this environment.

A general discussion of Automatic-Test-Systems in the next chapter will be followed by description of the hardware and software used in Test-Systems. This report will then be concluded by a detailed description of the actual hardware developed.

CHAPTER 2

FUNCTIONAL OVERVIEW OF TEST-SYSTEMS

2.1 INTRODUCTION

A functional overview of the basic principles involved in Test-Systems, will be followed by a discussion of the various hardware configurations used in Automatic-Test-Systems.

Bare Board testing will be followed by a discussion of methods used for testing populated circuit boards, namely Functional-Testing and In-Circuit Testing. Signature-Analysis which is applicable to both Finctional and In-Circuit Testing will be discussed under Functional Testing. The chapter is then concluded by a discussion of the two methods used in obtaining the Expected-Response.

Many Test-Systems in practice use a combination of the various methods described here. It is very common for a Dynamic-Functional-Tester to also have a Signature-Analyser attached to help in the tracing of faults.

2.2 GENERAL TEST PRINCIPLES

The increase in complexity of circuits has made the need for Automatic-Testing greater. Without Automatic-Testers, faulty circuits are only detected at or after system installation time, when tracing and fixing the fault can be very costly. The earlier a fault is detected the cheaper it is to fix.

At the start the printed circuit boards can be tested before the integrated circuits are soldered on. The populated circuit boards can then be tested functionally before being installed into the system. In the long run these tests can only serve to save a lot of time and money.

The general method of testing any digital circuit involves applying an Input-Stimulus to the Device-Under-Test, the measuring of the Actual-Response, and the comparing of the Actual-Response with the Expected-Response. The Expected-Response is the correct response expected from the Device-Under-Test on application of the Input-Stimulus. This basic principle applies to all Automatic-Digital-Circuit-Tests and a number of aspects have to be addressed :-

(a) Where is the Input-Stimulus applied and Actual-Response measured.
(b) How is the Input-Stimulus obtained.
(c) How is the Expected-Response obtained.
(d) Action if Actual and Expected-Response differ.

The point of application of the Input-Stimulus and measurement of the Actual-Response, is determined by the design of the Test-System Hardware. Functional-Testers are usually attached to the Edge-Connector of the Device-Under Test. In-Circuit Testers on the other hand are connected to all the components on the board, with a Bed-Of-Nails connection. In-Circuit Emulators replace the Micro-Processor with a connector. A guided probe can be generally used with most types of testers, and has the dual purpose of being able to inject and measure signals.

The Input-Stimulus is generated automatically by using Automatic-Test-Pattern Generation or Pseudo Random Pattern Generation, or manually by writing a Test-Program.

The Expected Response is usually obtained by using a Logic-Analyser or a Known-Good-Board.

The action taken if the Actual and Expected-Response differ, ranges from discarding the faulty board to tracing and fixing the fault. The fault can be traced by searching a Dictionary-Of-Faults, or by using a Guided Probe and comparing the faulty board with a Known-Good-Board.

2.3 BARE BOARD TESTING

Checking the bare board before the components are inserted can save money in the long run. The earlier faults are detected the better, since the cost of rejection is a lot less and faults are easier and cheaper to fix at this time.

In industry the bare board is often checked by simple visual inspection. With simple double-sided boards some of the obvious faults could be found, but with the trend towards denser, multilayer boards, this is even less effective than previously. The trend towards VLSI components densely packed on a board, means that the price of rejecting a populated circuit board is increasing. It follows that the need for effective bare board testers is increasing.

The method normally used for automatically testing bare boards will now be discussed.

2.3.1 BED-OF-NAILS TEST FIXTURES

To test the bare board it is necessary to be able to check the connections between all the component points on the board. This can be done using a Bed-Of-Nails Test Fixture.

The Bed-Of-Nails Test Fixture is basically a number of probes in a frame. The board to be tested fits into the frame and is then brought into contact with the probes by means of a vacuum or some sort of pressure plate. The probes come in various shapes and sizes and are usually spring loaded so that electrical contact is made with each point. These test jigs also sometimes have special spiked probes for making contact through solder flux so that populated boards can also be tested.

Each probe in turn has a stimulus applied to it and the other probes are monitored. In this way open circuits and short circuits can be detected. With some equipment it is possible to measure resistance and capacitance

between the various test points.

This method of testing boards is not confined to bare boards but can also be used in fully populated circuit boards. In the case of fully populated boards the stimulus is limited so as not to activate the components, when only the connections are being tested. A method of In-Circuit-Testing, called Back-Driving also uses a Bed-Of-Nails for testing fully populated circuit boards.

The types of probes used in Bed-Of-Nails Test Fixtures varies depending on the particular application in mind. An aspect of production cost to be considered, is that the boards to be tested need to be specially designed with this method in mind, with special locating holes etc. This is discussed in an article by Bedard (1982).

The main reason why this method was discarded for this project is the large cost due to the huge mechanical and electrical complexity. The number of probes is large as there is a connection to each component pin.

2.4 FUNCTIONAL-TESTING

Functional-Testing of a circuit board is concerned with the function of the board from an external point of view. Functional-Testers are typically connected to the Edge Connector of the board to be tested. With most Functional-Testers a Signature-Analyser can be used to measure signals not available at the edge connector, to aid in the tracking of faults, however Signature-Analysis is not restricted to Functional Testers as it can be used with some In-Circuit Testers.

Functional-Testers typically exercise the board, by applying patterns and reading the Actual-Response. The Actual-Response is compared with Expected-Response to see if the board is operational.

2.4.1 SIGNATURE-ANALYSIS

Signature-Analysis is discussed here under Functional-Testers even though this technique is used with a number of In-Circuit Testers.

A method of detecting faults is to stimulate the Device-Under-Test, and the Known-Good-Board, and then to compare the nodal movement of the two. The stimulation can be generated either internally from the board, or externally from a Functional-Tester, or from an In-Circuit-Tester. Stimulation usually consists applying a repetitive pattern. A method of compressing the data available at the nodes, is needed.

One simple method of tracking node movement is to count node transitions before repetition takes place. This method however is not time-sensitive and the same count will result as long as there are the same number of transitions. The timing of the transitions is immaterial. A method developed which is time-sensitive, is called Signature-Analysis.

Signature-Analysis involves shifting the incoming bit stream into a Shift-Register. Various bits in the Shift-Register are shifted back and added to the incoming bit stream. The resultant number in the Shift-Register is called the Signature and is time and transition sensitive.

The main purpose of Signature-Analysis is to reduce the complexity of a bit stream and this can be used in various test methods. Methods which use Signature-Analysis range from Pattern-Spraying-Testers, Internal-Testers in the Device-Under-Test, Dynamic-Functional-Testers and In-Circuit-Testers. Blyth (1982) mentions some possible uses.

Generally Signature-Analysis can be used with a wide range of Automatic-Testers used in the testing of Digital Circuits. The principle of simplifying a serial stream of node changes is useful. The actual hardware of the Shift-Register, which is the heart of a Signature-Analysis-System, is very simple, as can be seen from the Figure on the following page.

The basic block diagram of the Shift-Register is in the Figure below :-

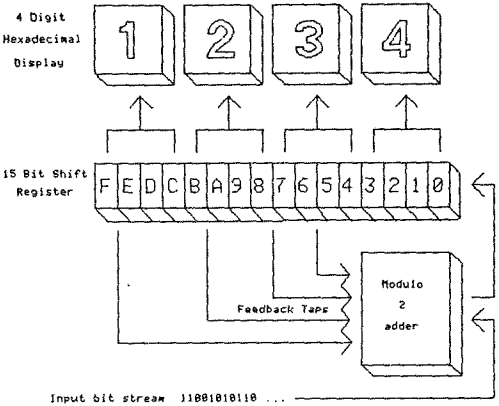


Figure 3 : Signature-Analysis

2.4.2 HIGH FREQUENCY PATTERN GENERATION

This method applies pseudo random Test-Patterns to both the Device-Under-Test and a Known-Good-Board. The Sensed-Patterns are compared and in this way the Device-Under-Test can be declared good or bad. If the board is bad then it is possible to probe various test points on the boards, and to do a comparison using a Signature-Analyser, and in this way track down the source of the error.

A simple application of this method is to compare the Device-Under-Test with a Known-Good-Board. The two boards are compared and any difference in response reported by flagging the faulty pin. The faulty pin is then

traced back through the circuit comparing the Known-Good-Board with the Device-Under-Test. This method, whit very simple, is fairly effective.

With complex systems like microprocessor or memory based systems the main disadvantage of this method is that without proper initialisation the results will be meaningless, as the response is a function of the input and the present state. Without initialisation the two boards would start off in different states, and consequently respond differently. Complex initialisation sequences are necessary, and these systems do not always support this feature.

Fault diagnosis using this method is relatively straightforward but is very laborious and assumes a knowledge of the Device-Under-Test. A method of fault diagnosis would be the use of a Signature-Analyser.

2.4.3 SLOW-SPEED-FUNCTIONAL-TESTER

This method involves having a computer directly in control of a number of driver/sensor pins. Driving and sensing is done directly by the computer through some type of parallel port. The computer applies patterns and reads the Actual-Response.

This method while effective with testing combinational circuits, and some sequential circuits, suffers from a number of limitations related to speed. This method is generally slow in comparison to the normal operating speed of the digital Device-Under-Test.

The slow speed means that certain speed related faults will not be detected, and circuits with dynamic components, or internal clocks cannot be tested effectively.

This method has the advantage of being simple, but the slow speed limits the usefulness of this method. As with most Functional-Testers, a Signature-Analyser can be added to aid in tracing faults.

2.4.4 DYNAMIC-FUNCTIONAL-TESTING (HIGH SPEED)

This method is the one used in the Test-Hardware developed for this project. The Input-Stimulus and the Actual-Response can be applied and measured at speeds of typically up to 10 MHZ, as local memory is dedicated to driving and monitoring the driver/sensor pins. The typical memory size is 1K behind each pin. The main advantages of this method is the speed at which the Input-Stimulus is applied and the Actual-Response measured. This allows the digital circuits to be tested at normal switching speeds and thus speed related faults such as glitches and slow rise times can be detected.

The Test-Hardware has a Pattern-Output-Memory which is loaded with the Test-Pattern at the slow I/O rate of the Controlling-Computer. The Test-Pattern is then applied in a quick burst and the monitored Actual-Response is loaded into the Pattern-Returned-Memory. This Returned-Pattern is then read at the slow I/O rate of the computer and examined to see if it matches what was expected. If it does not match the Expected-Response then Dictionary-Of-Faults, if available, may be searched to track down the exact fault.

The Test-Hardware in no way restricts how the Output-Patterns are generated or how the Sensed-Patterns are analysed. The Output-Patterns could be generated by Automatic Test-Pattern Generation, Pseudo Random Pattern Generation, or generated manually by an Engineer. The Expected-Response can also be generated by a number of methods, namely by using a Logic-Simulator, a Known-Good-Board, or by the same engineer that generated the Test-Pattern.

With Dynamic-Functional-Testers it is also possible to use a Signature-Analyser to aid in tracing faults. The circuit is stimulated by the Test-Hardware and Signatures are read at various nodes and compared with expected Signatures.

This Hardware Method is very versatile which is the reason it was chosen.

2.5 IN-CIRCUIT-TESTING

In-Circuit-Testers work by applying and measuring signals internally to the circuit. This is either done by forcing the internal signals into a particular state, as with Back-Driving, or by replacing a component with a connection to the tester, as with In-Circuit-Emulation.

With In-Circuit-Testing the circuit is tested from an internal point of view and as such the testing is normally far simpler from an analysis point of view. With the increase in circuit complexity this method is becoming more and more attractive.

2.5.1 IN-CIRCUIT-EMULATION

This method is briefly described by Blyth (1982) and in more detail by Popky (1982). This method is used in the testing of microprocessor based systems. With circuit boards based on microprocessors it is very difficult to test the board properly from the edge connector, as the complexity of the board makes this task almost impossible.

In-Circuit-Emulation involves removing the microprocessor from the board and plugging in a connector from the Test-System in its place. A program to test the board is then either loaded into the memory of the Target-System or the memory of the Emulation-System. This program, which is written by an engineer, then tests the Target-System. If the Test-Program runs sucessfully then, as long as the actual microprocessor is working, the Device-Under-Test should work when the microprocessor is replaced.

Some Emulation-Systems have an extra feature which allows the actual microprocessor removed also to be tested. Emulation-Systems generally have a version of the actual microprocessor to be emulated contained in the emulator pod. These systems in effect instead of having this built into the pod have a socket for the user to supply the microprocessor. This allows the microprocessor and the Target-System to both be tested simultaneously.

In-Circuit-Emulators are probably the optimum method of checking microprocessor based systems, but they are generally fairly complex and are normally dedicated to the microprocessor which they are designed to emulate. The Test-Programs require a high degree of skill to write.

In-Circuit-Emulation can be used for both go/nogo Functional-Testing, and for fault isolation using a guided probe. This method is also compatible with the Signature-Analysis method of fault isolation.

The main advantages of this method is that, since it becomes the main controlling element, it can test all the functional elements. Testing can be done in real time and no special test fixtures are needed except that the microprocessor should be in a socket, or an extra socket should be available with a method of tri-stating the on-board microprocessor.

For microprocessor based systems this is generally by far the cheapest and most powerful method for automatically testing the circuits.

2.5.2 BACK-DRIVING

In this method the interfacing of the Device-Under-Test is via a Bed-Of-Nails Test Fixture. The connections on the baord are checked and each component is checked for presence, orientation, type or value and basic function. Assuming the design is correct then a board that passes the test has a high probability of working.

The board is first checked for correct connection. This is done at very low voltages and currents to prevent any further damage if something is wrong already. Once this is done the board is powered up for the next phase of the test.

The next phase of the test is called Back-Driving and involves forcing the inputs from their quiescent state to check the particular device on the board. The outputs of the device are measured to see if they are in the correct state. Forcing the inputs from their quiescent state is done for a very short period of time so that power dissipation in the output stages is not sufficient to produce a dangerous temperature rise.

One of the advantages of this method is that tests are written for each device on the board. Testing the Device-Under-Test is largely circuit independant as each component on the board is tested individually. Each component is normally fairly simple so writing the test programs is normally limited to giving a connection description.

One of the main disadvantages of this method is the cost of manufacture of the Bed-Of-Nails Test Fixture. This is however offset to a certain extent by savings in the programming costs.

2.6 ESTABLISHING EXPECTED-RESPONSE

It is very important to have an accurate Expected-Response which is needed to decide if Device-Under-Test is faulty or not. The Expected-Response is compared with the Actual-Response to verify correct operation of the circuit being tested.

The Expected-Response to certain fault conditions is also sometimes needed for storage in a Dictionary-Of-Faults which is used to analyse the faults detected in the Device-Under-Test. If the Actual-Response obtained does not match the Expected-Response then the Dictionary-Of-Faults is searched to see if the Actual-Response matches one of these.

Henckels (1982) compares the two methods of getting the Expected-Response, namely from a Known-Good-Board or from a Logic-Simulator. This article gives a good comparison of the two methods.

2.6.1 KNOWN-GOOD-BOARD

Using a Known-Good-Board is one method of obtaining the Expected-Response. Once the Test-Program has been developed the Known-Good-Board can be used to test the effectiveness of the Test-Program, to get the Expected-Response, and to generate a Dictionary-Of-Faults if necessary.

Firstly the Test-Program is run on the Known-Good-Board and the response is stored as the Expected-Response of a Good Board. A Fault Injector is then placed on each IC of the Known-Good-Board in turn and the Test-Program is run to see if the particular fault causes the output of the board to deviate from that of the Expected-Response. If this is the case then this means that the Test-Program will detect this particular fault. If the fault is not detected then this fact is recorded.

Once the whole board has been checked, statistics are generated to determine the percentage of faults detected. If the percentage is unsatisfactory then the Test-Program will be modified accordingly.

While the board is having faults injected, it is possible to store the responses in a Dictionary-Of-Faults which can be referenced during production testing to determine the actual fault on a faulty board.

There are a number of disadvantages of this method. One of the biggest problems is getting a Known-Good-Board. It is often necessary to have a special board built with all the IC's in sockets. If the Known-Good-Board functions incorrectly, the whole exercise could be meaningless.

Another problem is the fact that faults can be generated by the actual connecting of the Fault Generator Clip without even actually injecting a fault. This is often due to the capacitive loading the Clip places on the pins. With TTL this is not much of a problem, however with CMOS, NMOS and PMOS this can be a problem.

Another problem with this method is that it is the Test-Hardware is used while the Test-Program is checked and the Dictionary-Of-Faults is created. This means that production testing is suspended for a significant period of time.

2.6.2 LOGIC-SIMULATION

A Logic-Simulator is another method of obtaining the Expected-Response to a Test-Program. With a Logic-Simulator it is possible to automatically generate the faults and check the effectiveness of the Test-Program. As with the Known-Good-Board the responses to the various faults tested can

be stored in a *Dictionary-Of-Faults* so that actual faults can be detected when a faulty board is found during production testing.

One advantage of *Logic-Simulation* is that undefined states can be propogated out. In this way initialisation of the board can be tested.

It is necessary to enter a *description* of the circuit into the Test-System and then the *Test-Program* can be tested automatically. This however takes a lot of *computer time* but luckily can be done off-line without actually using the *Test-Hardware*.

One of the main disadvantages of *this method* is that it is necessary to have a software model of all the *IC's* on the board and with certain VLSI chips like Microprocessors, UARTS, DMA Controllers and Single-Chip-Computers this is very difficult. *These circuits* are generally very difficult to test, and are a problem for *conventional* testers. In-Circuit-Emulators used in conjunction with a *Known-Good-Board* are probably a better solution for this particular *type of circuit*.

Modelling the various logic *families* can be a *problem* with some simpler Logic-Simulators. More complex Simulators can *actually* warn the operator about loading problems, and probable race *conditions*.

2.7 CONCLUSION

The method chosen for this project is the method of *Dynamic-Functional-Testing* which for circuits of reasonable complexity *should be* adequate. In-Circuit-Emulation should be used for more complex *microprocessor* based circuits.

The hardware of Automatic-Test-Systems has a number of *different* tradeoffs in that the more expensive and complex the system, the *better* it will perform. The consideration of cost and mechanical *complexity* was the reason why a Bed-Of-Nails Test Fixture was discarded.

CHAPTER 3

FAULT MODELLING AND TEST-PATTERN GENERATION

3.1 INTRODUCTION

In Automatic-Digital-Circuit-Testing-*Equipment* the main function is to detect *faulty circuits* on the production line before they are sent out, unserviceable into the *field*, *since* the cost of repair in the *field* far *exceeds* the cost of fixing or replacing the *circuit* on the production line,or at a *service* establishment. Certain faults in the *field can* have disastrous results.

The earlier a fault *is* detected the cheaper it is to fix. With *large* systems it is better to test the *various* component circuits before assembling the whole system and thereafter testing the system as a whole.

The task is therefore to test a circuit on the production line or in the laboratory. *With* digital circuits of any complexity it *is* totally unfeasible to apply all possible *inputs* to the circuit. It is therefore *necessary* to confine the tests to finding certain *anticipated* faults.

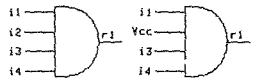
The Test-Program used was generated to test for certain expected faults. These *faults* are usually generated and the Test-*Program* is run to determine if the faults are detected. This was discussed earlier where Logic- Simulation was compared with the *method* of using a Known-Good-Board. *With* Logic-Simulation the faults are entered into the software model of the circuit and the Test-*Program* is used to apply the inputs and *measure* the outputs to see if the faults are *detected*. With the Known-Good-Board actual faults are injected either by short circuiting, or *open* circuiting, certain pins.

3.2 TYPES OF FAULTS

Most digital faults are generally modelled by the following three approaches.

3.2.1 STUCK-AT-FAULTS

This fault model assumes that a gate input, or output is stuck at a specific level. This could be caused by a faulty input, faulty output driver, open input, shorted input or output to ground or Vcc. The Figure below shows the test for a Stuck-at-1 (S-A-1) fault on input 2 (i2). This fault is usually designated as i2/1.



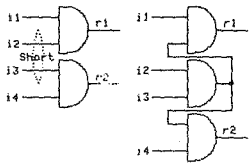
Good Model Faulted Model

Pattern Applied				Response	
i1	i2	i3	i4	r1	
1	0	1	1	0	Good Machine Model
1	0	1	1	1	Faulted Machine Model

Figure 4: Stuck-At-Fault

3.2.2 SHORTS

Shorts between signal lines of the network change the function of the circuit. Shorts to ground and Vcc are looked on rather as S-A-0 and S-A-1 faults, as described above. With the TTL logic family, an output sinks a lot more current than it sources. If two opposing outputs are shorted together the line will be pulled low by the stronger sinking current. With two shorted outputs of the same value the resultant level is not affected. This effect of the stronger sinking current causes the resultant level to be a logical 'AND' of the two shorted lines. This may however change for other logic families. The shorting of two outputs is demonstated in the Figure below where the shorting of inputs 2 and 3 (i2 and i3) cause the function of the circuit to change.



Good Model Faulted Model

Pattern Applied				Response		
i1	i2	i3	i4	r1	r2	
1	0	1	1	0	1	Good Machine Model
1	0	1	1	0	0	Faulted Machine Model

Figure 5: Short Fault

3.2.3 PARAMETRIC FAULTS

The two above faults are called DC faults as they are present from high frequencies right down to DC. Parametric Faults on the other hand are speed related and are usually only present at the normal switching speed of the digital circuit. These faults include slow gates, race conditions, slow rise time, and various types of loading faults where the load is too high and causes the signal to change slowly. The only way to detect these faults is to test the circuit at operational switching speed. The Test-Hardware employed is critical for detecting these types of faults.

To detect parametric faults it is necessary to apply Test-Patterns at the full operating speed of the Device-Under-Test. Dynamic-Function-Testers such as the one developed for this project should have enough speed to accomplish checking for parametric faults.

3.3 TYPES OF CIRCUITS

There are three basic types of digital circuits, namely Combinational, Synchronous Sequential Circuits, and Asynchronous Sequential Circuits. Automatic-Digital-Circuit-Testers can handle Combinational and Synchronous Sequential Circuits but Asynchronous Sequential Circuits are very difficult to handle and no generalised method of doing so has yet been devised. At design time it is best to design synchronous circuits as this makes testing so much easier. With a little thought it is possible to substitute synchronous circuits for asynchronous ones. An example of this is the circuit below where a one-shot (Typical Asynchronous Element) can be replaced by a synchronous circuit using two D-Flip Flops.

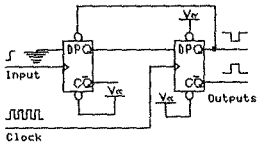


Figure 6 : Synchronous One-Shot

2.3.1 COMBINATIONAL CIRCUITS

With combinational circuits the outputs are solely a function of the inputs. The output is a direct function of the input. A typical example of this is a NAND gate. The general model of a Combinational Circuit is detailed below in Figure 7. Combinational Circuits are the simplest digital circuits and as such are the easiest to test. Sequential circuits whether Synchronous or Asynchronous always contain Combinational Circuits as sub units.

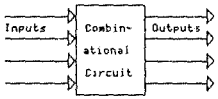


Figure 7 : Combinational Circuits

3.3.2 SYNCHRONOUS SEQUENTIAL CIRCUITS

Sequential Circuits differ from Combinational Circuits in that the outputs are a function of both the inputs and the previous state. A typical Sequential Circuit has a Latch where the previous state is retained. Synchronous Sequential Circuits are Sequential Circuits which change state at specific times. A clock usually controls the change of state. Sequential circuits can thus be simplified into a Combinational Circuit for each state, and thus are relatively easy to test using Automatic-Test-Equipment. The general Synchronous Sequential circuit is shown below in Figure 8.

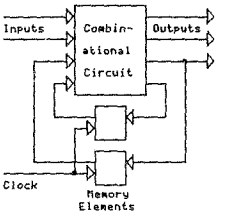


Figure 8 : Synchronous Sequential Circuits

3.3.3 ASYNCHRONOUS SEQUENTIAL CIRCUITS

Asynchronous Sequential Circuits are Sequential Circuits with a stored state and the output is a function of the previous state and the input. The difference from Synchronous Circuits is that the feedback loop does not necessarily occur at specific intervals and may involve some type of delayed wait. Typical elements in an Asynchronous Sequential Circuit are one-shots. The use of one-shots usually turns a Synchronous Sequential Circuit into an Asynchronous Sequential Circuit which makes it almost impossible to test on most Automatic-Test-Equipment as the Equipment samples the output at specific intervals normally linked to a clock within the Device-Under-Test. With Asynchronous circuits the problem

arises as to when the circuit should be sampled.

One-Shots in circuits cause difficulty in testing, and are also usually susceptible to spikes on the ground and Vcc pins which can cause spurious triggering.

The general case of an Asynchronous Sequential Circuit is detailed below in Figure 9. Asynchronous Sequential Circuits are almost impossible to test generally.

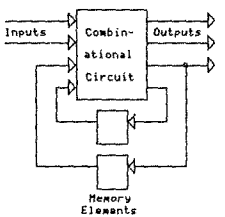


Figure 9 : Asynchronous Sequential Circuits

3.4 FAULT FOLDING

With digital networks many faults along a particular path may be indistinguishable. If two faults are indistinguishable they are called equivalent, and only one test is needed for detecting both equivalent faults. In this manner the number of faults to be tested for can be reduced considerably, making the Test-Program that much simpler.
The method of reducing the number of faults to be tested for falls into

the study of fault equivalence, dominance and folding. This is discussed by Muehldorf (1981) in an article and is also discussed in the book by Breuer (1976)

Fault folding basically boils down to a method of reducing the number of faults to be tested for. These faults are reduced to all single Stuck-At-Faults, on all primary inputs and all branches of fanout. The details are specified in the above references.

Detailed analysis of methods used in the analysis of faults, and the automatic generation of Test-Patterns will not be gone into in detail as the main aim of this project report is to discuss the broad aspects which may influence the Test-Hardware required.

For fault folding it is necessary to have a model of the circuit entered into the Test-System and some fairly complex software to do the fault folding. As with Logic-Simulators and automatic Test-Pattern generation programs the problems can be solved relatively easily with medium scale integration (MSI) components, but huge problems arise when one looks at very large scale integration (VLSI) components.

3.5 TEST-PATTERN GENERATION

Test-Pattern generation is concerned with what patterns should be applied to the Device-Under-Test in order to check whether it is operational or faulty. This discussion is concerned with the various methods of applying patterns to the connector or the pins directly, and is not concerned with In-Circuit-Emulators as these testers differ completely in concept. In-Circuit-Emulators drive tests from the microprocessor and are very closely related to the whole bus structure of microcomputers and are more of a programming exercise for the Device-Under-Test. The testers we are concerned with have their signals applied externally to the board, or externally to the individual integrated circuits populating the boards.

3.5.1 MANUAL METHOD

In this method an engineer programs the test hardware to apply various Test-Patterns to the Device-Under-Test. The engineer in question needs to know the functions of the board to be tested and the Test-Hardware to be used. Thus a large degree of skill is needed to generate the Test-Programs.

However cumbersome this method may sound it is still widely used in industry as it is fairly simple in concept and will always work to a certain degree. With some automatic methods they may suddenly come across a circuit that cannot be analysed, such as a new VLSI microprocessor chip, and then the method is completely useless. This method should always work to a certain degree even if the percentage of faults actually checked is relatively small.

The usefullness of this method varies depending on the engineer concerned and the amount of time allocated to writing and checking the Test-Programs.

3.5.2 RANDOM METHOD

This method involves spraying the inputs with a pseudo random pattern and detecting any deviation from the expected output. The expected output as explained earlier could be established from a Logic-Simulator or a Known-Good-Board. With this method the Known-Good-Board would usually be the method used as the complexity of having a simulation of the circuit would usually mean a more analytic method of generating the Test-Patterns would be used.

This method can have fairly good results as long as the circuit starts off in a known state. In order for the circuit to start of in a known state certain initialisation of the Device-Under-Test needs to be undertaken. This can be a problem with certain sequential circuits, and the hardware used to generate random patterns is not normally suitable for long involved initialisation sequences.

Another problem could be with tri-state lines where the lines are both

3.5.1 MANUAL METHOD

In this method an engineer programs the test hardware to apply various Test-Patterns to the Device-Under-Test. The engineer in question needs to know the functions of the board to be tested and the Test-Hardware to be used. Thus a large degree of skill is needed to generate the Test-Programs.

However cumbersome this method may sound it is still widely used in industry as it is fairly simple in concept and will always work to a certain degree. With some automatic methods they may suddenly come across a circuit that cannot be analysed, such as a new VLSI microprocessor chip, and then the method is completely useless. This method should always work to a certain degree even if the percentage of faults actually checked is relatively small.

The usefullness of this method varies depending on the engineer concerned and the amount of time allocated to writing and checking the Test-Programs.

3.5.2 RANDOM METHOD

This method involves spraying the inputs with a pseudo random pattern and detecting any deviation from the expected output. The expected output as explained earlier could be established from a Logic-Simulator or a Known-Good-Board. With this method the Known-Good-Board would usually be the method used as the complexity of having a simulation of the circuit would usually mean a more analytic method of generating the Test-Patterns would be used.

This method can have fairly good results as long as the circuit starts off in a known state. In order for the circuit to start of in a known state certain initialisation of the Device-Under-Test needs to be undertaken. This can be a problem with certain sequential circuits, and the hardware used to generate random patterns is not normally suitable for long involved initialisation sequences.

Another problem could be with tri-state lines where the lines are both

inputs and outputs at different times during the operation of the circuit. Randomly applying patterns may cause the Test-Hardware to clash with the Device-Under-Test when they both try and drive the same line.

3.5.3 AUTOMATIC TEST-PATTERN GENERATION

This method is the most complex from the Test-System's point of view. A description of the Circuit is entered into the Test-System in much the same way as for a simulator. The Test-System needs fairly powerful computing capabilities, however this may be done on some external computer and the Test-Patterns loaded down to the Test-Hardware once the computing is completed. The fully automated Test-System would do the Fault Folding, the Automatic Test-Pattern Generation, and finally would use a Logic-Simulator to establish the Expected-Response and a Dictionary-Of-Faults.

This method sounds very attractive as once the circuit description has been entered then nearly everything else is automatic. However there is one big downfall with this method, which is that a library needs to be kept of all components used in the circuits. This works reasonably for MSI where gate level descriptions of the various chips are available, but has become a big problem with the widespread use of VLSI. Gate level descriptions of Microprocessors, Serial Controllers, DMA Controllers etc are not available and are very difficult to design.

This method works very well as long as the circuit is simple enough to be handled. A number of methods are used. Some methods rely on an algebraic description of the circuit, namely the Algebraic Expression Method and the Boolean Difference Method. The Boolean Difference method involves differentiating the algebraic expression of the circuit to see how the output differs to a fault condition. Other methods rely on a topological description of the circuit, namely ath Sensitisation Method and D-Algorithm method. These methods work on applying a pattern which will propogate a fault to one of the outputs. The various methods are described in greater detail in Muehldorf (1981) and in Breuer (1976).

3.6 CONCLUSION

Automatic Fault Folding and Automatic Test-Pattern Generation, are very attractive as long as they work properly. The main disadvantages of these methods is that a large amount of computing power is needed and the introduction of VLSI components has made it very difficult to generate library parts.

For small companies it is probably unfeasible even to consider trying to develop their own library parts of VLSI components. It may be possible to rent some very sophisticated packages at a considerable cost. The amount of work to keep such systems up to date with current VLSI components is huge.

Many Test-Patterns are still generated manually by an Engineer as this method is fairly simple in concept, and works for most circuits. Even though the resultant Test-Programs might not be very exhaustive at least they work for a certain percentage of possible faults.

Logic-Simulators use the same software description of the Device-Under-Test as used for Automatic Test-Pattern Generators. Logic-Simulators will be discussed in detail in the next chapter.

CHAPTER 4

LOGIC-SIMULATION

4.1 INTRODUCTION

A Logic-Simulator is a program which runs on a host computer. The purpose of a Simulator is to model the behavior of a given digital circuit. It is necessary to have a significant amount of computing power to run some of the more complex Simulators. Test-Systems which include automatic fault folding and Test-Pattern generation would normally have a Logic-Simulator, as the principles involved are closely allied in many ways and the circuit description need only be entered once. This can then be used by all three packages, namely : Automatic Fault Folding, Automatic Test-Pattern Generation, and Logic-Simulation with automatic generation of a Dictionary-Of-Faults and Expected-Response, from the Test-Pattern put through the Simulator.

Simulators are not only found in Automatic-Test-Systems. Often sophisticated and some not so sophisticated Computer Aided Design (CAD) Systems have Simulators as part of the whole design package. Pure circuit layout packages have library parts for the pinouts and physical measurements of the various components used. These library parts are then connected together in the design. All the connection information of the circuit is available to the CAD package. With Simulators the electrical characteristics are added to the library parts to help with the Simulation.

Having a Simulation available of the circuit design can be invaluable in determining if the design will actually work when the printed circuit board is manufactured. If prototypes are first built and then the design is transferred to the CAD System then the Simulator can be used to check

that there were no faults in copying the design to the CAD System. The Simulator can also check for certain design flaws, depending on how sophisticated it is. With a very good Simulator a number of prototyping phases may be skipped in bringing a circuit to manufacturing level.

Simulators can be very useful from the simplest up to the most complex. Obviously the more features needed the more complex becomes the actual Simulator. One of the main downfalls of Simulators has been the use of VLSI where the complexity of the components makes it almost impossible to generate some of the library primitives. Microprocessors, DMA controllers and Serial Controllers are some examples of difficult to Simulate VLSI circuits.

For this project a very simple Simulator written in BASIC was used to test some of the features of Simulators. It was also used to Simulate some of the timing circuitry used in the Test-Hardware. The listing and output of this Simulator can be found in the Appendix J and Appendix L. The Simulator was a modified version of the one developed by McDermott (1983). It is a Table-Driven-Simulator with Event-Directed-Simulation, Named Nodes, Macrocircuit Capability, Independent Inputs and Sampled Outputs. The features mentioned here are explained below in greater detail.

4.2 BASIC SIMULATOR STRUCTURE

There are two basic types of Simulators, namely Compiler-Driven-Simulators and Table-Driven-Simulators. These two have many similarities to High Level Languages which are Compiled or Interpreted. The description of the circuit in a way is like a computer program which is translated and executed by the computer.

4.2.1 COMPILER-DRIVEN-SIMULATION

In Compiler-Driven-Simulation the circuit description is translated into machine executable code, such as Assembly Language. Before the code generation can take place the circuit must first be transformed into the standard format of a synchronous sequential circuit. This can be seen below in figure 10. The combinational part is then relatively easy to translate into Assembly Language.

This method assumes zero delay through the gates and as such will only work with synchronous circuits which can be translated into the standard format. As such this method is very restrictive and is not as general purpose as Tab e-Driven-Simulators.

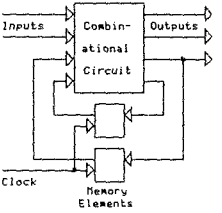


Figure 10 : Syncrhronous Standard Format

4.2.2 TABLE-DRIVEN-SIMULATION

Table-Driven-Simulators can handle circuit delay and many circuit characteristics that cannot be handled by Compiler-Driven-Simulators. The main disadvantage of Table-Driven-Simulators is that the execution time is necessarily slower than Compiler-Driven-Simulators.

Table-Driven-Simulators in principle are relatively simple. Each circuit element has inputs and outputs. Outputs from one gate go to the inputs of another gate. Each signal line between gates, called a node, is given an

entry in an array of node values. There are two identical arrays with all the same nodes except the one is for inputs and the outher for outputs from the various gates. During a particular execution phase the input values from the input array are applied to the various models of the gates and the outputs from the gates are written to the output array. After the execution phase is complete the output array is copied into the input array for the next execution phase. A basic diagram of this can be seen below in Figure 11.

This basic function in its simplest form assumes a constant delay through each gate. This problem is easily solved by assuming some type of time granularity and then each gate needs a certain number of time grains before the inputs changing affects the outputs. In this way time delays through gates can be easily modelled.

Table-Driven-Simulators are fairly simple in concept. However, very complex Simulators can be built from this simple structure. Table-Driven-Simulators have two phases of operation. The initial phase involves generating symbol tables, setting up the arrays of nodes, setting up the gate models and expanding any macros. The next phase is the execution phase where the Simulation takes place and the outputs are displayed.

The basic execution phase of a Table-Driven-Simulator is contained in the figure below :-

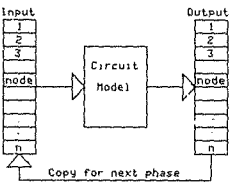


Figure 11 : Table-Driven-Simulator

4.2.3 EVENT-DIRECTED-SIMULATION

With Table-Driven-Simulators a small percentage of inputs may change from one state to the next. With gates the outputs do not change if the inputs do not. Therefore it is not necessary to Simulate each gate every time.

It is only necessary to Simulate the gate if the inputs change from one state to the next. In this way a large amount of time can be saved by not Simulating gates that are not going to change. This optimisation can be very important with Table-Driven-Simulators as they are typically slow.

4.2.4 NAMED NODES

An input to a gate and an output from a gate are called nodes. Some very simple Simulators insist that you number the nodes and then these numbers are usually element positions in the input and output arrays. However things can be made much more user friendly by taking a tip from the use of Assembly Language and allowing named nodes. A symbol table is built up in much the same way as an Assembler. Each symbol is placed in the symbol table along with its array entry number. When a new symbol is read from the circuit description the symbol table is searched and if an entry is not found then a new entry is made. The symbol table also contains information as to where the symbol is defined from the output of a gate. If at the end of the compiling of the symbol table a symbol is not defined as an output anywhere then an error is flagged.

4.2.5 MACROCIRCUIT CAPABILITY

Macrocircuit capability is the ability to create your own macros from the primitive library parts. This is very useful if you want to define a component with primitive constituents. This component can then be used in the circuit a number of times without having to define the primitive constituents every time.

This feature is another one that comes from Assembly Language as did the use of named nodes. In many ways the initial table building phase, and macro expansion resemble the operation of an Assembler.

4.2.6 INDEPENDENT INPUTS

The circuit to be Simulated may have a number of external inputs, driven from outside the circuit. These inputs in a real circuit would be driven by external circuit components of the system, which have different functions in the overall System. In a System based around a computer bus these would be the assorted circuit cards, with different functions to the circuit in question. There are two basic types of inputs, namely periodic and aperiodic inputs.

The Periodic input resembles a clock input and the frequency and duty cycle of this signal should be definable. Once defined these signals will keep oscillating at the defined frequency.

Aperiodic inputs are those where the actual value at each particular point in time must be defined. This is done by defining the starting value and the time of each signal change.

4.2.7 SAMPLED OUTPUTS

The outputs of the Logic-Simulator should resemble those of a Logic-Analyser where a certain number can be displayed. The sampling freqency and the signal to be sampled should be specified. This gives a trace very similar to that of a Logic-Analyser.

The sampling frequency can be easily adjusted by only sampling after a specified number of time grains. All output changes in the interim are ignored.

Effectively it is possible to specify the names of the signals to be displayed along with the sampling frequency. This in effect is identical to the features available on a Logic Analyser.

4.3 CIRCUIT DELAY

There are a number of delays found in digital circuits and for an accurate Simulation it is necessary to take these into account.

4.3.1 TRANSPORT DELAY

This delay is through a circuit element and through the wire connecting the various circuit elements. This delay from the input to a component, can be viewed as a delay in the output driver. As such this delay is added to the output delay of the circuit element driving the signal.

4.3.2 [illegible] DELAY

Most circu[illegible]vices do not have an exact delay. Delay is [illegible]ually specified as a maximim and mininum value and varies from chip to [illegible]p, and also changes with temperature. It is necessary to have a maximum delay and a minimum delay and the Simulator should check for any race conditions that may be influenced by a variation in delay.

4.3.3 RISE-FALL DELAY

When the output from a gate changes there is a particular time it takes to rise or fall depending on factors like capacitance and driving capacity of the gate. With MOS devices these delays can differ considerably depending on the load. This delay should also be taken into account in the Simulator.

4.3.4 INERTIAL DELAY

Inertial delay must be considered where an incoming signal has a duration too short to actually switch the gate. Many gates have a minimum pulse width that will be recognised and this should be taken into account and reported as a possible hazardous condition. The simulated gates which receive a pulse of too short a duration, should not switch.

4.4 MULTI VALUED LOGIC

With Simulators a number of problems can be solved by having Multi-Valued logic. In Multi-Valued Logic each node has more than one bit to define its state. An extra bit could be used for tri-state, undefined state etc.

4.4.1 INITIALISATION PROBLEM

With Simulators it is possible to detect initialisation problems that may arise in circuits. With real circuits they are either high or low and the circuit may power up differently each time. An extra bit is added to the value of each node and defines whether the signal is defined or not.

Certain laws of boolean logic are applied. An unknown signal 'AND'ed with a low will give a low out. An unknown signal 'AND'ed with a high will give an unknown signal out. This law can be derived from the following boolean formulas (X.0 = 0) and (X.1 = X). This process can be applied to all gates and unknown signals can be reported in the output by a specific symbol like a question mark.

4.4.2 HAZARD DETECTION

In detecting hazards it is necessary to take into account the fact that digital signals do not change instantaneously. A signal takes some time in changing from low to high or vice versa.

Two extra states are added to the normal states, by the addition of an extra bit, namely a 0 to 1 transition state, and a 1 to 0 transition state. These states can be fed into the various logic elements and in this way hazards can be detected more easily as the information about the changing state of a signal is available. When two inputs to a gate are in transition then the gate may enter an unknown hazardous state. This condition is dependant on the type of gate in question.

4.5 FAULT SIMULATION

With Simulators it is possible to change the circuit to Simulate various types of faults, from shorts, stuck nodes, to delay problems.

4.5.1 DICTIONARY-OF-FAULTS

With a Simulator it is possible to Simulate the circuit with a various common faults introduced. The Test-Program is used as input for the circuit and the resultant output of the circuit is stored for each fault introduced. The output must differ from the expected output of a good circuit for the Test-Program to detect the fault. These outputs from the circuits having injected faults are stored in what is called a 'Dictionary-Of-Faults'. The fault is stored along with the Output-Pattern obtained with that fault.

During production testing if the output differs from the Expected-Response then the Dictionary-Of-Faults is searched to see if the output matches any in the Dictionary. If a match is found then the actual fault stored with the pattern in the Dictionary can be reported.

4.6 CONCLUSION

Certain circuits may be too complex to Simulate on a Logic Level. , this aspect Logic-Simulation software is similar to that used for Automatic Fault Folding and Automatic Test-Pattern Generation. If the circuit to be modelled is not too complex these methods can be very useful.

The main downfall with Simulators is with certain VLSI components the circuit model is very difficult to Simulate. Another disadvantage is that Logic-Simulators are normally slow and need significant computing power.

This brings to a close the general discussion of Test-Systems. The specific Test-System developed for this project will now be discussed.

CHAPTER 5

FUNCTIONAL DESCRIPTION OF THE TEST-SYSTEM DEVELOPED

5.1 INTRODUCTION

This chapter describes the Automatic-Test-Hardware that was developed for the purposes of this project, from a functional point of view. The software needed to drive the Test-Hardware is described in this chapter, and the next chapter deals with the Test-System from a hardware point of view.

The hardware developed falls into the category of Dynamic-Functional-Testers where there is dedicated memory behind the test pins which allow the patterns to be driven ans sensed at high speeds. The basic operation of Dynamic-Functional-Testers is that there are two banks of memory, one is the Pattern-Output-Memory for the Driven-Patterns, and the other is the Pattern-Returned-Memory for the Sensed-Patterns.

The Main-Control-Computer gives the patterns to be applied to the 8086 Single-Board-Computer in the Test-Hardware. This Single-Board-Computer then loads the patterns into the Pattern-Output-Memory at normal bus speed. Once this is finished the Test-Hardware is activated and in a quick burst outputs the patterns at the same time reading the inputs into the Pattern-Returned-Memory. This burst is sent at the switching speed of the Device-Under-Test. Once this is finished the Test-Hardware is de-activated and the patterns are read from the Test-Hardware at bus speed and sent up to the Main-Control-Computer where they are analysed.

The main function of the Test-Hardware is the ability to output and input patterns at very high speeds, in this case up to 16 MHz.

5.2 OVERALL SYSTEM DESCRIPTION

The Test-System basically consists of a Main-Control-Computer connected via a serial link to the 8086 Single-Board-Computer which in turn controls the Dedicated Test-Hardware over MULTIBUS. The Test-Hardware in turn is connected to the Device-Under-Test. This basic layout can be seen in the figure below :-

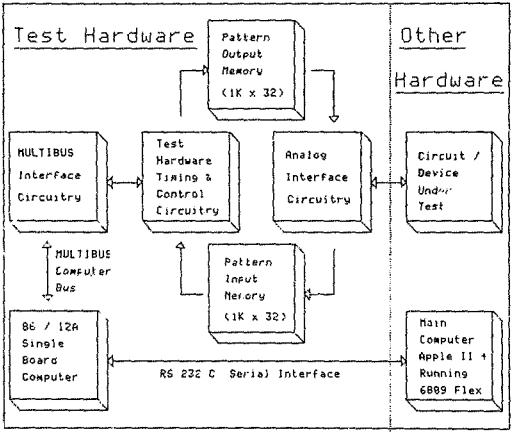


Figure 12 : Basic Block Diagram

The Test-System used to test the Device-Under-Test consists of the Main Computer and the Test Hardware.

The Main Computer in this case was an Apple II + compatible computer. This choice was made mainly due to the availability of the particular computer system. This computer could be replaced by any mini-computer or micro-computer with the required processing power and storage capacity.

The function of the Main Computer is to act as the Data-Base for all the patterns and programs loaded down to the 86/12A Computer via the RS-232C Serial Link. Automatic Test Pattern Generation, simulation and detailed analysis would be done on this Computer.

The 86/12A Single-Board-Computer was chosen as a fast computer was needed to control the Dedicated-Test-Hardware developed. The availability of an 86/12A Single-Board-Computer and an 8086 Emulator at the University, were important factors used in choosing the 86/12A Single-Board-Computer.

The Dedicated-Test-Hardware which is connected to the 86/12A Single-Board-Computer on MULTIBUS, has the main function of driving and sensing patterns. The application of patterns to the Device-Under-Test, and the sensing of the response, needs to be executed at speeds up to 16 MHz. This speed is needed to test Digital Circuits at operational speeds.

To obtain these high speeds it is necessary to have dedicated memory behind the electronics of the driver / sensor pins. The Pattern-Output-Memory is loaded from the 86/12A Single-Board-Computer at normal bus speeds. Once this is completed the Dedicated-Test-Hardware is then placed in test mode, where the patterns are driven out at high speed from the Pattern-Output-Memory. At the same time the sensed patterns are clocked into the Pattern-Input-Memory.

The Analog-Interface-Circuitry is needed to interface the seperate digital inputs and outputs to the testers driver / sensor pins which drive and sense tri-state signals.

The main functions of the various constituents will now be descussed in greater detail.

5.2.1 MAIN-CONTROL-COMPUTER

The Main-Control-Computer which is connected to the Test-Hardware via a serial link is the main computing power in the Test-System. This computer in the fully developed Automatic-Test-System would contain all the software necessary to perform functions like Automatic Fault Folding, Automatic Test-Pattern Generation, and Logic-Simulation. The Dictionary-Of-Faults would also be stored here along with any complex software function necessary. This computer in a fully fledged Automatic-Test-System would be either a micro-computer or a mini-computer, with peripheral storage for the application programs.

For the purposes of this project an APPLE II + compatible computer was used along with a custom built serial card. The reason for this choice was the fact that the software developed had the prime function of verifying that the Test-Hardware was operational. Another factor was the fact that with Microcomputers it is a lot easier to get into the Operating-System at a low level and control the hardware directly. With mini-computers with multiple users the Operating-System has to protect users from one another and consequently makes it difficult to achieve certain low level functions.

The APPLE II + computer needed to communicate with the 8086 Single-Board-Computer over the Serial Link and for this purpose a custom serial card was built. The reason for this is that the standard APPLE II + serial card uses software rather than a UART to shift the data in and out. This means that processing of the data can only take place during the stop bits which limits the speed of data transmission considerably. The serial card built uses a UART and transmission speeds up to 4800 baud could be used. With the original serial card 150 baud was the fastest that could be attained. The custom serial card increased the possible transmission speed by a factor of 32 which is considerable. The actual design of the Serial Card developed is contained in Appendix F and Appendix G.

For the purposes of checking the Test-Hardware it was necessary to communicate with the Intel monitor on the 8086 Single-Board-Computer. This usually involved typing long sequences of instructions which became

extremely tedious. A serial communication program was developed to aid in this aspect. The serial communication program was written under the 6809 Flex Operating-System on the APPLE II + compatible. This program normally operated like a Terminal-Emulator but had a number of special options. It was possible to send sequences to the 8086 Single-Board-Computer from a file stored under Flex. Another feature was that it was possible to capture all typing sequences in a file, and another option was the ability to capture all outputs from the Monitor Program running on the 8086 Single-Board-Computer.

A listing of this program is contained in Appendix H, and all tests of the Test-Hardware were done using this serial communication program. The listings of the various tests in the Appendix, namely Appendix N and Appendix O, are actual terminal sessions captured by the serial communication program into a file and printed later.

5.2.2 8086 SINGLE-BOARD-COMPUTER

The 8086 Single-Board-Computer was actually an 86/12A SBC manufactured by Intel. This Single-Board-Computer is a MULTIBUS compatible computer with an 8086 microprocessor as the main processing element. The board contains 32K of dual ported RAM and four EPROM sockets which can hold a variety of EEPROMS with the most dense being 2732 EPROMS which gives the board a maximum capacity of 16K of EPROM. The board also contains an 8251A (Programmable Communications Interface), an 8253 PIT (Programmable Interval Timer), an 8255A PPI (Programmable Peripheral Interface), and an 8259A PIC (Programmable Interrupt Controller). For a more detailed description see the iSBC 86/12A Single-Board-Computer Hardware Reference Manual by Intel (1979).

The board was fitted with a Monitor Program which communicated with the Main-Control-Computer over the Serial Link. The monitor description is contained in the iSBC 957A Intellec - iSBC 86/12A Interface and Execution Package User's Guide by Intel (1980). This is a fairly standard monitor program which allows memory modification, display, single stepping etc.

5.2.3 TEST-HARDWARE

The Test-Hardware has the main function of applying Test-Patterns to the Device-Under-Test and measuring the Actual-Response at the same time. Each pin is both a driver and a sensor at the same time. The actual level is measured all the time and the pin can be driven high, low or into tri-state during application of the Test-Pattern. This means that a particular pin can be changed from driving to sensing, and vice versa, during a test sequence. This means that systems employing tri-state can be tested. Dual thresholds are used and an optional load pulls the line to the middle of the illegal region of TTL to distinguish tri-state from the high and low levels.

The Test-Hardware excluding the 8086 Single-Board-Computer consists of three MULTIBUS compatible cards. The functions of these three cards is as follows :-

Card 1) Multibus Interface, clock generation and control circuitry.
Card 2) Pattern-Input and Output memory, latches and multiplexers.
Card 3) Analog Interface circuitry.

Card no (1) contains the interface to the MULTIBUS. The card is configured as a slave card and appears to the 8086 Single-Board-Computer as memory and I/O ports. The clock generation is used to control the whole timing of the Test-Hardware as the sampling and pattern output must be synchronised with the Device-Under-Test. The Sampling clock can be generated internally in the Test-Hardware or can be obtained from the Device-Under-Test. The control circuitry controls the Operational Modes and has a Control-Register for this purpose. A Status-Register is also available to read the present state of the Test-Hardware. Incrementing through memory and many other functions are handled by the control circuitry.

Card no (2) contains the Pattern-Input and Output memory along with latches and multiplexers which are controlled by the control circuitry on card no (1). This card also contains the digital comparators used in detecting the Trigger-Pattern used in the Logic-Analyser Mode.

Card no (3) is the analog interface and takes the purely digital signals from card no (2) and interfaces with the Device-Under-Test. This card consists mainly of Analog Comparators, Tri-State drivers and voltage generation circuitry to control the thresholds. Dual Thresholds are used in the sensing of tri-state lines. Two digital inputs and two digital outputs from card (2) are combined in this card into one general purpose I/O pin for driving and sensing high, low and tri-state. The thresholds can be changed for different logic families as needed.

The functions of the Test-Hardware are detailed in the figure below :-

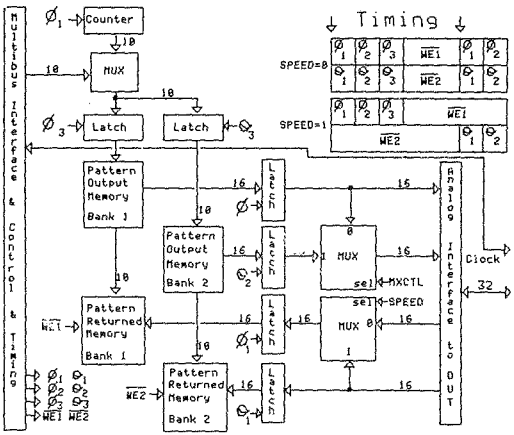


Figure 13 : Test-Hardware Functional Diagram

5.3 SYSTEM CONTROL

The various Operational and Configuration modes will now be discussed followed by a discussion of the Control-Register, Status-Register, the mapping of the Pattern-Output-Memory and Pattern-Input-Memory, followed by a description of the Trigger-Pattern.

This gives a programmers viewpoint of the Test-System.

5.3.1 MODES OF OPERATION

There are three basic Operational Modes of the Test-Hardware and these are as follows :-

1) Digital-Tester Mode

The Test-Hardware goes from the Start to the End of the dedicated I/O memory driving the the Test-Patterns and sensing the Returned-Patterns at the same time. This mode does a single pass of the I/O memory. This mode is the normal mode of operation in testing a digital circuit.

2) Logic-Analyser Mode

The Test-Hardware keeps Looping repetitively through the dedicated I/O memory going from the Start to the End and then repeating this. The Test-Patterns are driven and the Returned-Patterns are sensed the whole time. A Trigger-Pattern is set up on the Trigger-Switches and when the Returned-Pattern matches the Trigger-Pattern the address of the trigger is captured in the Status-Register and the Test-Hardware keeps on operating for half the memory so that half the patterns captured are before, and half after the trigger point. In a pure Logic-Analyser all patterns driven would be purely tri-state, making all pins purely sensors. This mode allows the Test-Hardware to operate as a Logic-Analyser.

3) **Triggered Digital-Tester Mode**

This mode is similar to the Digital-Tester Mode in that a single pass is made of the dedicated I/O memory. The main difference here is that the single pass is only started once a Pattern-Input matches the Trigger-Switches. This mode would be used to apply the Test-Stimulus and capture the Actual-Response once a certain state is reached.

The analog interface board in this case was designed for the testing of TTL circuits with the option of using tri-state detection. The Test-Hardware is connected to the analog interface via a connector and it would be relatively easy to design a number of different interfaces with different characteristics for testing digital circuits. One option considered was to use flip-flops for the detection of glitches.

The Test-Hardware was required to operate at speeds of up to 16 MHz and for this purpose it was necessary to do memory multiplexing. With memory multiplexing speeds of up to 16 MHz can be reached and without memory multiplexing speeds up to 8 MHz can be reached.

The multiplexing and non-multiplexing modes combined with the different ways that the analog card can be set up give 4 different Configuration Modes. These Configuration modes can be seen in the following table :-

	Memory/Pin	No.Pins	Max Speed	Detect State
1)	1K	32	8 MHz	Bi - State
2)	1K	16	8 MHz	Tri- State
3)	2K	16	16 MHz	Bi - State
4)	2K	8	16 MHz	Tri- State

Table 2 : Different Configuration Modes

The Test-Hardware has a Control/Status-Register and Dedicated I/O Memory. The Control-Register and the Status-Register are at the same I/O Port Address. However the Control-Register is written and the Status-Register is read. The dedicated I/O memory consists of Pattern-Output-Memory and Pattern-Returned-Memory and these two memory banks are similarly at the same location except the one is written to and the other is read from. Both memory banks are in turn split up into low and high banks and these two banks have different operations depending on whether memory multiplexing is chosen or not. If multiplexing is not chosen then each bank supplies 16 bit wide digital I/O giving a total of 32 input and 32 output bits. Under memory multiplexing the high and low banks are alternated to give increased speed and consequently the number of input and output bits is halved giving 16 inputs and 16 outputs. However under memory multiplexing the amount of memory behind each pin increases from 1K to 2K.

5.3.2 CONTROL-REGISTER

The Control-Register is used to control the Operational Modes of the Test-Hardware as well as whether memory multiplexing is used or not. The various control bits are discribed in the Table below :-

Bit 15	STONTRIG	0 = Normal Start	Start without Trigger-Pattern matching input. Undefined for CLEAR=0.
		1 = Start on Trigger	Start when Trigger-Pattern matches input.
Bit 14	CLEAR	0 = Clear	Clear Test-Hardware.
		1 = Go	Releases Clear.
Bit 13	NORMLA	0 = Normal Single Scan	Normal single scan mode, and reset Logic-Analyser Mode.
		1 = Logic-Analyser On	Logic-Analyser mode for looping.

Table 3 : Control-Register (Continued ...)

Bit 12	EPOS	0 = Disable Pos Edge	*No positive edge clocking.*
		1 = Enable Pos Edge	Clock on *positive edges.*
Bit 11	ENEG	0 = *Disable* Neg Edge	No negative edge clocking.
		1 = *Enable Neg Edge*	Clock on negative edges.
Bit 10	EXINT	0 = External Access	*External* access of memory from 8086 SBC. Disables clocking as well.
		1 = Internal Access	Internal Access for running *Test-Hardware.*
Bit 9	SPEED	0 = Normal Speed	Non-multiplexed mem access.
		1 = Full Speed	Multiplexed memory acess.
Bit 8	G2	0 = Enable Clock2	*Drive Auxiliary Clock 2.*
		1 = Disable Clock 2	*Tri-State Auxiliary Clock 2.*
Bit 7-5	C2,B2,A2	0,0,0 = Single Step	Speed of Clock 2.
		0,0,1 = 00.25 MHz	(Auxiliary Clock).
		0,1,0 = 00.50 MHz	
		0,1,1 = 01.00 MHz	
		1,0,0 = 02.00 MHz	
		1,0,1 = 04.00 MHz	
		1,1,0 = 08.00 MHz	
		1,1,1 = 16.00 MHz	
Bit 4	G1	0 = Enable Clock1	Drive Main Clock 1.
		1 = Disable Clock 1	Tri-State Main Clock 1.
Bit 3-1	C1,B1,A1	0,0,0 = Single Step	Speed of Clock 1.
		0,0,1 = 00.25 MHz	(Main Clock).
		0,1,0 = 00.50 MHz	
		0,1,1 = 01.00 MHz	
		1,0,0 = 02.00 MHz	
		1,0,1 = 04.00 MHz	
		1,1,0 = 08.00 MHz	
		1,1,1 = 16.00 MHz	
Bit 0	SSTEP	0 = Step Low	Clock Low Under Single Step.
		1 = Step High	Clock *High Under* Single Step

...e 3 : Control-Register

5.3.3 STATUS-REGISTER

The Status-Register contains the present state of the Test-Hardware as well as the Trigger-Address when the machine is triggered under the Logic-Analyser mode. The various bits are detailed in the Table below :-

Bit 15	TRIG STARTED	0 = Start FF Off	Under triggered start the Test-Hardware has not yet triggered.
		1 = Start FF On	The Hardware has triggered or is permanently enabled.
Bit 14	LA TRIGGERED	0 = No Trigger	Logic-Analyser not triggered
		1 = Triggered	Logic-Analyser has been triggered.
Bit 13	LA FINISHED	0 = LA Finished	Logic-Analyser has finished counting after being triggered.
		1 = LA Busy	Logic-Analyser is waiting for trigger or is counting after being triggered.
Bit 12	LA CLOCKING	0 = LA Stopped	The Logic-Analyser Clock is stopped either because finished or not in LA Mode.
		1 = LA Clocking	The Logic-Analyser is operating.
Bit 11	CLOCKING	0 = Not Clocking	The main counters which clock through memory are stopped after single pass or because disabled.
		1 = Clocking	The main counters are busy clocking in single pass or are counting during LA Mode before trigger.

Table 4 : Status-Register (Continued ...)

Bit 10	MXCTL	0 = High Memory	Accessing high memory during multiplexing mode.
		1 = Low Memory	Accessing Low memory during multiplexing mode, or not in multiplexing mode at all.
Bit 9-0	ADDRESS	Trigger Address	Address of trigger under LA mode. Once triggered address is held in these bits. The address is address of 16 bit word where trigger occurred.

Table 4 : Status-Register

5.3.4 PATTERN-OUTPUT-MEMORY

There are two bit output memory one at address offset 800H above the memory base , and the second at address offset 880H. In the non-multiplexed memory access mode the low bits are taken from the bank at address offset 800H and the high order bits from the bank at address offset 880H. In the multiplexed memory access mode only the high bits are driven and they are taken from the low bank at offset 800H and the high bank at address offset 880H in turn, before incrementing the count to the next location.

Two bits are used to control the value of the outgoing signal. One bit controls whether the signal is driven or in tri-state and the second bit is used to control whether the signal is driven high or low.

Under the multiplexed mode there are 8 different drivers and 16 under the non-multiplexed mode. Due to the way in which memory is accessed it is necessary when writing to and reading from the dedicated I/O memory to always read and write to bit words on even boundaries. As with the way Intel stores the words the least significant bits are stored in the low memory location.

The drivers are controlled by writing words to the Pattern-Output-Memory using the format detailed in the Table below :-

Tn = Tri-State Control Bit n	0 = Put Pin in Tri-State 1 = Drive pin according to Vn
Vn = Drive Control Bit n	0 = Drive Pin Low 1 = Drive Pin High

Table 5 : Output Memory Bit Definitions

The structure of Pattern-Output-Memory can be seen in the Table below :-

Speed = 0 (Non - Multiplexed Mode)

At offset 000H (default 0:0000)

Low Address								High Address							
T4	V4	T3	V3	T2	V2	T1	V1	T8	V8	T7	V7	T6	V6	T5	V5

At offset 800H (default 0:0000)

Low Address								High Address							
T12	V12	T11	V11	T10	V10	T9	V9	T16	V16	T15	V15	T14	V14	T13	V13

Speed = 1 (Multiplexed Mode)

At offset 000H (default 0:0000) and offset 800H (default 0:0000)

Low Address								High Address							
T4	V4	T3	V3	T2	V2	T1	V1	T8	V8	T7	V7	T6	V6	T5	V5

Table 6 : Pattern-Output-Memory Structure

5.3.5 PATTERN-RETURNED-MEMORY

There are two banks of intput memory one at address offset 000H above the memory base address, and the second at address offset 800H. In the non-multiplexed memory access mode the low bits are read into the bank at address offset 000H and the high order bits to the bank at address offset 800H. In the multiplexed memory access mode only the high bits are read and they are put into the low bank at address offset 000H and the high bank at address offset 800H in turn, before incrementing the count to the next location.

The sensed signal is transformed into two bits of input. In the Tri-State detection mode there are two different thresholds used. The High threshold would typically be set at 2.4 Volts and low threshold typically at 0.8 Volts. The sensed signal is then pulled via a resistor network to 1.6 Volts when in tri-state. The high input would be the one using the high threshold and the low input the one using the low threshold. The three states of the sensed signal are determined by the following Table :

Low Signal -	Input High = 0 Input Low = 0
Tri-State Signal -	Input High = 0 Input Low = 1
High Signal -	Input High = 1 Input Low = 1

Hn = Input High Bit n	0 = Below High Threshold 1 = Above High Threshold
Ln = Input Low Bit n	0 = Below Low Threshold 1 = Above Low Threshold

Table 7 : Input Memory Bit Definitions

The state where Input High = 1 and Input Low = 0 is Illegal. This state should never occur in practice. When the Test-System is used to detect Bi-State then switches on the Analog Interface Board Shorting the two inputs are opened and the Thresholds are set to the same value of about 1.6 Volts. This then gives you effectively twice the number of inputs.

The outputs still work in pairs so one must be careful in using the Bi-state detection mode as the mode is not as fully supported as the Tri-State detection mode.

Under the multiplexed mode there are 8 different inputs and 16 under the non-multiplexed mode. Due to the way in which memory is accessed it is necessary when writing to and reading from the dedicated I/O memory to always read and write 16 bit words on even boundaries. As with the way Intel stores the words the least significant bits are stored in the low memory location.

The inputs are read in using the structure in the Table below :-

Speed = 0 (Non - Multiplexed Mode)

At offset 000H (default 0:8000)

Low Address								High Address							
H4	L4	H3	L3	H2	L2	H1	L1	H8	L8	H7	L7	H6	L6	H5	L5

At offset 800H (default 0:8800)

Low Address								High Address							
H12	L12	H11	L11	H10	L10	H9	L9	H16	L16	H15	L15	H14	L14	H13	L13

Speed = 1 (Multiplexed Mode)

At offset 000H (default 0:8000) and offset 800H (default 0:8800)

Low Address								High Address							
H4	L4	H3	L3	H2	L2	H1	L1	H8	L8	H7	L7	H6	L6	H5	L5

Table 8 : Pattern-Returned-Memory Structure

5.3.6 TRIGGER-PATTERN

The Trigger-Pattern is set up using 32 three position switches. The three positions are namely, high, don't care, and low. The 32 switches correspond closely to the sensors input to the Pattern-Returned-Memory.

In the Non-Multiplexed mode (Speed = 0) all 32 switches can be used in selecting the pattern. However in the Multiplexed mode (Speed = 1) only the higher 16 switches are used. In this mode (Speed = 1) the lower 16 switches must be placed in the middle position, for don't care.

NOTE In Multiplexed Mode (Speed = 1) the lower 16 switches must be put in the don't care state.

The Trigger-Pattern set up on the switches is only operational in the Logic-Analyser mode and the Triggered Digital-Test Mode. In the normal Digital-Test Mode these switches are not used at all.

The switches come in pairs like the Pattern-Returned-Memory bits where in Tri-State detection the Upper bit uses the upper threshold, and the Lower bit uses the lower threshold. In Bi-State detection each switch represents a bit.

The bit definitions are detailed in the Table below :-

Hn = Pattern High Bit n	0 = Below High Threshold 1 = Above High Threshold
Ln = Pattern Low Bit n	0 = Below Low Threshold 1 = Above Low Threshold

Table 9 : Trigger-Pattern Bit Definitions

The structure of the switches is contained in the table below. The switches are listed from left to right just as they appear on the Trigger-Pattern switch box.

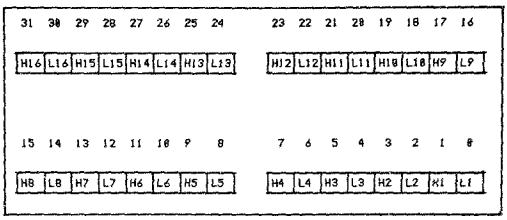


Table 10 : Trigger-Pattern Structure

5.3.7 CONTROL-REGISTER PROGRAMMING CODES

This section deals with the codes sent to the Control-Register to control the Test-Hardware. These codes should make it easier to program the Test-Hardware without having to work out each bit in the Control-Register.

EXTERNAL ACCESS BY SBC OF TEST-HARDWARE 0110H

CLEAR SEQUENCE OF TEST-HARDWARE 1500H
1501H
0110H

SETTING LOGIC-ANALYSER OFF

			1/4MHz	1/2MHz	1MHz	2MHz	...	16MHz
Pos	Edge	Speed = 0	7422H	7444H	7466H	7488H		74EEH
Neg	Edge	Speed = 0	6C22H	6C44H	6C66H	6C88H		6CEEH
Both	Edges	Speed = 0	7C22H	7C44H	7C66H	7C88H		7CEEH
Pos	Edge	Speed = 1	7622H	7644H	7666H	7688H		76EEH
Neg	Edge	Speed = 1	6E22H	6E44H	6E66H	6E88H		6EEEH
Both	Edges	Speed = 1	7E22H	7E44H	7E66H	7E88H		7EEEH

Before the Logic-Analyser has Triggered get A???H from Status-Register.
After the Logic-Analyser has Triggered get C???H from Status-Register.

SETTING DIGITAL-TESTER MODE OFF

	1/4MHz	1/2MHz	1MHz	2MHz ...	16MHz
Pos Edge Speed = 0	5422H	5444H	5466H	5488H	54EEH
Neg Edge Speed = 0	4C22H	4C44H	4C66H	4C88H	4CEEH
Both Edges Speed = 0	5C22H	5C44H	5C66H	5C88H	5CEEH
Pos Edge Speed = 1	5622H	5644H	5666H	5688H	56EEH
Neg Edge Speed = 1	4E22H	4E44H	4E66H	4E88H	4EEEH
Both Edges Speed = 1	5E22H	5E44H	5E66H	5E88H	5EEEH

SETTING TRIGGERED DIGITAL-TESTER MODE OFF

	1/4MHz	1/2MHz	1MHz	2MHz ...	16MHz
Pos Edge Speed = 0	D422H	D444H	D466H	D488H	D4EEH
Neg Edge Speed = 0	CC22H	CC44H	CC66H	CC88H	CCEEH
Both Edges Speed = 0	DC22H	DC44H	DC66H	DC88H	DCEEH
Pos Edge Speed = 1	D622H	D644H	D666H	D688H	D6EEH
Neg Edge Speed = 1	CE2.	4H	CE66H	CE88H	CEEEH
Both Edges Speed = 1	DE22H	44H	DE66H	DE88H	DEEEH

5.4 CONCLUSION

The functional description given in this chapter should contain sufficient information to enable programming the Test-System assuming one knows how to use the 86/12A Single-Board-Computer. See Appendix N and Appendix O for examples of the programming.

In the examples in the Appendix the Control & Status-Register were set up at I/O port number 188H. Both registers occupy the same word location except that the one is written to and the other is read from. The Control-Register is written and the Status-Register is read.

The default setup for the Dedicated I/O Memory is from location 0000:8000H to 0000:8FFF. The lower bank is from 0000:8000 to 0000:87FF and the upper bank is from 0000:8800 to 0000:8FFF. In writing and reading from the dedicated I/O memory two things must be kept in mind. Firstly bit 10 (EXINT) of the Control-Register must be set low for external access and information must be written and read using word transfers on even boundaries.

Further details about the hardware functions of the Test-System developed are contained in the next chapter.

CHAPTER 6

TECHNICAL DESCRIPTION OF THE TEST-HARDWARE

6.1 INTRODUCTION

The hardware will be discussed in detail in this chapter. The basic physical layout of the 3 boards and 3 connectors is in the Figure below :

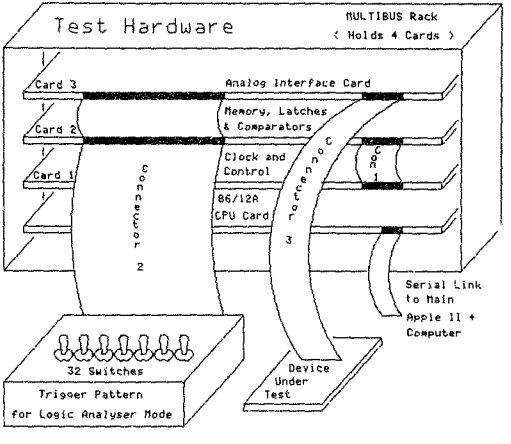


Figure 14 : Basic Physical Layout

The Test-Hardware excluding the 8086 Single-Board-Computer consists of three MULTIBUS compatible cards. The functions of these three cards is as follows :-

Card 1) Multibus Interface, clock generation and control circuitry.
Card 2) Pattern Input and Output memory, latches and multiplexers.
Card 3) Analog Interface circuitry.

There are three connectors used to join the cards, excluding the MULTIBUS connector. The functions of these three connectors is as follows :-

Connector 1) Timing and Control Connector.
Connector 2) Digital I/O Connector.
Connector 3) Analog I/O Connector.

These cards and connectors will now be described in greater detail in this Chapter. The detailed circuit diagrams are contained in Appendix A, and will not be found in this Chapter. The discussion will not describe the function of each and every gate as this can be obtained from the circuit diagram.

6.2 FUNCTIONS OF THE TEST-HARDWARE

The Test-Hardware consists of four multibus size boards, of which one is an 8086 Single-Board-Computer whose function it is to control the other boards which are dedicated boards to the Test-Hardware. The basic function of the Test-System as described earlier is to output and input patterns at high speed. The Test-System is controlled by use of a Control-Register and Status is returned via a Status-Register. The Dedicated I/O memory looks like memory to the Single-Board-Computer.

The functional diagram of the Test-Hardware is in the Figure below :-

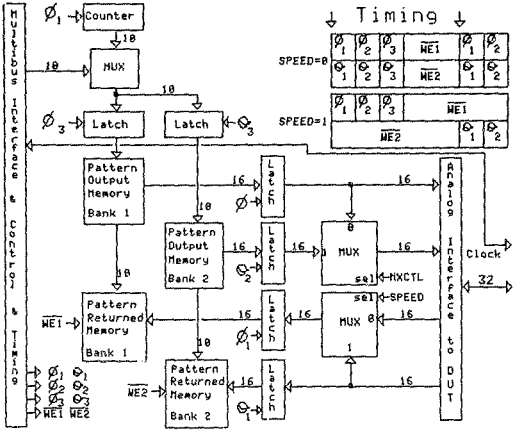


Figure 15 : Test-Hardware Functional Diagram

In the figure above the signal called "Clock" is the signal used to control when the Test-Patterns should be applied and the Returned-Patterns read. This "Clock" signal can be driven by the Device-Under-Test or by the Test-Hardware itself. It is possible to output and sample on the Positive Edge, the Negative Edge, or Both Positive and Negative Edges.

The Timing shown in the upper right corner is used to specify the order in which events take place. The case of speed = 8 is the normal non multiplexed memory access mode, for lower speeds. The case of speed = 1 is the multiplexed memory access mode, for very high speeds. The order

of operations is very important to the function of the Test-System and this timing is controlled by the clock generation circuitry.

The various phases will now be described in greater detail :-

ϕ_1

This clock pulse Latches the incoming data in for Bank 1 and increments the Counter used to count through the dedicated I/O memory in the Test-Hardware. The Returned-Patterns are latched in before the outgoing patterns are changed to avoid the problem of reading a changing signal.

ϕ_2

This clock latches the outgoing patterns from the Pattern-Output-Memory of Bank 1 so that the values are passed along to the Analog Interface Card.

ϕ_3

This clock latches the count through to Bank 1 Pattern-Output-Memory and Pattern-Returned-Memory once it has been incremented.

$\overline{WE1}$

This signal goes low to write the Returned-Pattern into the Bank 1 of Pattern-Returned-Memory. The Returned-Pattern has already been latched.

ϕ_1'

This clock pulse Latches the incoming data in for Bank 2. The Returned-Patterns are latched in before the outgoing patterns are changed to stop the problem of reading a changing signal.

ϕ_2'

This clock latches the outgoing patterns from the Pattern-Output-Memory of Bank 2 so that the values are passed along to the Analog Interface Card.

3

This clock latches the count through to *Bank 2* Pattern-Output-Memory and *Pattern-Returned-Memory* once it has been incremented.

$\overline{WE2}$

This signal goes low to write the Returned-Pattern into the *Bank 2* of Pattern-Returned-Memory. The *Returned*-Pattern has already been latched.

The dedicated Test-Hardware is built on three seperate boards and the functions of these three boards will now be described in the following sections.

6.3 MULTIBUS INTERFACE AND CONTROL BOARD (Card 1)

Card no (1) contains the interface to the MULTIBUS. The card is configured as a slave card and appears to the 8086 Single-Board-Computer as memory and I/O ports. The clock generation is used to control the whole timing of the Test-Hardware as the sampling and pattern output must be synchronised with the *Device-Under-Test*. Sampling-Clock can be generated internally in the Test-Hardware or can be obtained from the Device-Under-Test. The control circuitry controls the Operational Modes and has a Control-Register for this purpose. A Status-Register is also available to read the present state of the Test-Hardware. Incrementing through memory and many other functions are handled by the control circuitry.

6.3.1 MULTIBUS INTERFACE

The MULTIBUS interface circuitry does the decoding and interfacing to the Test-Hardware on the three boards. The other two boards only take power

off MULTIBUS and do *not* interface to the bus as such. The MULTIBUS interface makes the Test-Hardware look like a SLAVE board with a bank of memory and a Control/Status-Register at a port address.

The Tran.. Acknowledge signal /XACK is generated by the MULTIBUS interface and the waiting time is switch selectable. The base address of the Control/Register and the dedicated I/O memory are also switch selectable.

The switches are contained in three 8 switch dual in line packages at locations 34, 23 and 12 on the board. These 24 switches control the base addresses and the Transfer acknowledge wait.

The switches are numbered according to their position on the card as shown in the table below :-

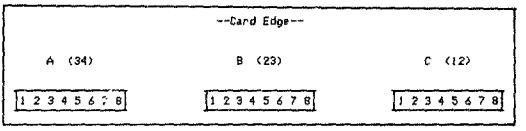


Table 11 : Switch Numbering

The default switch positions for the memory at 0:B800H -) 0:8FFF and the Control/Status-Register at 180H is in the Table below :-

A (34)	B (23)	C (12)
0 0 0 0 0 0 0 0	1 1 1 0 0 0 0 1	0 0 0 0 1 0 0 0

0 = Closed
1 = Open

Table 12 : Default Switch Settings

Memory Base Address

The *Dedicated I/O Memory* base address can be set up for a 16 address bit option or a 20 address *bit option*. For 16 address bit option switch B4 must be open, and closed for a 20 addesss bit option. With a 16 address *bit* option the upper four address bits are ignored.

When addressing the *Dedicated I/O memory* this must be done by addressing 16 bit words at even boundaries.
The base addess is set up according to the *following Table :-*

B4 = open for 16 address bits = closed for 20 address bits	
/ADR13 -> C1 /ADR12 -> C2 /ADR11 -> C3 /ADR10 -> C4	0 = Closed 1 = Open for /ADR
/ADR F -> C5 /ADR E -> C6 /ADR D -> C7 /ADR C -> C8	0 = Open 1 = Closed for ADR
/ADR is inverted ADR as *it appears* on the bus.	

Table *13 : Dedicated* I/O Memory Base Address

Control/Status-Register Address

The Control/Status-Register address *can* be set up for an 8 bit port address option or a 12 bit port address option. *For* an 8 *bit port* address option switch A1 must be open and closed for a 12 bit port address option. With an 8 *bit port* address option the upper four address bits are ignored.

The Control/Status-Register is a 16 bit register and as such is located at an even address. The low order byte is at the even address and the high order byte at the odd address.

The address is set up according to the following Table :-

A1 = open for 8 address bits = closed for 12 address bits	
/ADR 8 -> B5 /ADR A -> B6 /ADR 9 -> B7 /ADR 8 -> B8 /ADR 7 -> A2 /ADR 6 -> A3 /ADR 5 -> A4 /ADR 4 -> A5 /ADR 3 -> A6 /ADR 2 -> A7 /ADR 1 -> A8	0 = Closed 1 = Open for /ADR 0 = Open 1 = Closed for ADR
/ADR is inverted ADR as it appears on the bus.	

Table 14 : Control/Status-Register Address

Transfer Acknowledge Wait

The Transfer Acknowledge /XACK wait is switch selectable. The wait is from the time that the Slave board is addressed to the time that it is ready for the transfer to take place. This allows boards of varying speeds to access the bus. The wait state may be changed from 1 Clock cycle to 8 clock cycles on this interface. The clock cycles referred to are the /CCLK signal on the MULTIBUS.

The wait states may be set up according to the following Table :-

P1	B2	B3	Wait
Closed	Closed	Closed	1 Clock Cycle
Closed	Closed	Open	2 Clock Cycles
Closed	Open	Closed	3 Clock Cycles
Closed	Open	Open	4 Clock Cycles
Open	Closed	Closed	5 Clock Cycles
Open	Closed	Open	6 Clock Cycles
Open	Open	Closed	7 Clock Cycles
Open	Open	Open	8 Clock Cycles

Table 15 : Wait State Switch Selection

This concludes the discussion on the MULTIBUS interface. For more information see the circuit diagram in Appendix A and the Intel Literature on MULTIBUS.

6.3.2 CLOCK AND TIMING CONTROL

The clock and timing control circuitry has the function of timing all the circuits in the Test-Hardware. It is necessary to Output from the Pattern-Output-Memory and Input into the Pattern-Returned-Memory at particular intervals. This sampling is controlled by the Main-Sampling-Clock signal which can either be input from the Device-Under-Test or it can be driven by the Test-Hardware at speeds from 0.25 MHz up to 16MHz.

The driving and sensing of patterns can take place on the Positive Edge, on the Negative Edge, or on Both Edges of this Clock. These factors are set by the Control-Register which is detailed in the previous chapter.

There is a Main-Sampling-Clock, that controls the sampling, and an Auxiliary-Sampling-Clock which can be used for various purposes. The Auxiliary-Sampling-Clock is seperately programmible from the Main-

Sampling-Clock and can also run from 8.25 MHz up to 16 MHz. The two clocks are however multiples of one another, as they are derived from the same source, This fact may be very useful if you want a faster clock to drive the Device-Under-Test and only want to sample at say one sixteenth of this clock speed.

To drive and sense at 16MHz, internal timing was controlled by a 67MHz clock, which was obtained by using a phase locked loop and a Crystal oscillator.

There are two basic modes of operation of the Test-Hardware namely multiplexed and non multiplexed memory access. Under multiplexed memory access the access to the memory banks is interlaced and this results in far higher speeds. An offshoot of this is that the number of I/O lines is halved and the memory behind each I/O pin is doubled. Under non multiplexed access both banks work together and the number of I/O lines is double that of the multiplexed access.

Speed = 0 (Non-Multiplexed)

This is the non multiplexed access mode. The number of I/O pins is double that of the multiplexed mode and the memory behind each pin is half. Under this mode both banks 1 and 2 operate together. The multiplexer control signal MXCTL remains high and the clock signals generated are in sync for the two banks. This mode is used for normal speeds of up to about 8 MHz.

For this mode SPEED remains low and MXCTL remains high.

The timing diagram for this mode can be seen in the Figure below :-

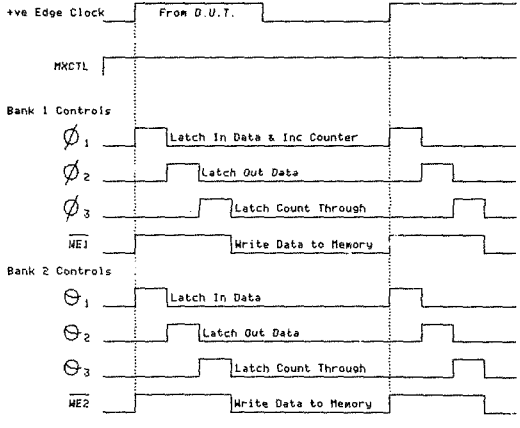


Figure 16 : Speed 0 Timing Diagram (Non-Multiplexed)

Speed = 1 (Multiplexed)

This is the multiplexed access mode. The number of I/O pins is half that of the non-multiplexed mode and the memory behind each pin is double. Under this mode both banks 1 and 2 operate alternatively on each sampling edge so that each has sufficient time to write and read from memory. The multiplexer control signal MXCTL oscillates depending on which bank is being used for each sample and the clock signals generated are out of sync for the two banks. This mode is used for high speeds of up to about 16 MHz.

The timing diagram for this mode can be seen in the Figure below :-

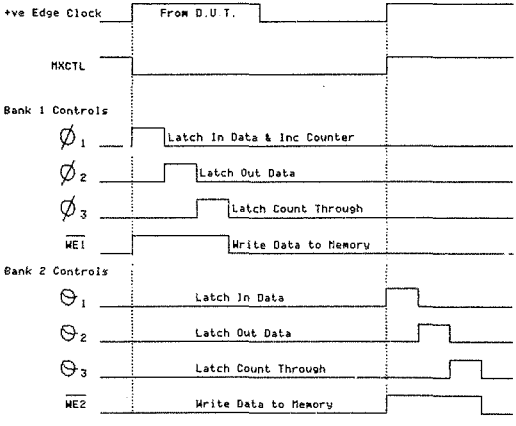


Figure 17 : Speed 1 Timing Diagram (Multiplexed)

6.3.3 ADDRESSING AND MODE CONTROL

This section of the circuitry is used to control the Test-Hardware and address the dedicated I/O memory. There is a Control-Register and a Status-Register in this circuitry. The functions of these registers is described in the previous chapter.

The main counter in this circuitry is used to address the dedicated I/O memory and this counter is incremented while the Test-Hardware is operational so that each pattern is fetched from a new memory location and the Returned-Patterns stored in a new memory location. The address from this counter is also fed through a multiplexer which is used to select whether the address should come from the counter or from MULTIBUS.

There is another counter which is used in the Logic-Analyser mode when the Trigger-Pattern matches the present address is latched in the Status-Register and this counter is started. Once this counter has counted to half the available memory it disables the main counter. This effectively gives half the memory before the trigger point and half after the trigger point.

The main counter usually terminates at terminal count during normal operation mode. In Logic-Analyser mode the main counter keeps counting until stopped by the Logic-Analyser counter.

Another feature of this circuitry is the ability to support the triggered test mode which means normal testing commences when the Trigger-Pattern matches the Returned-Pattern.

There is support for a manual trigger so that it is possible to trigger the Logic-Analyser or start the Tester under Triggered Tester Mode without the Trigger-Pattern matching.

6.4 MEMORY, LATCH, MULTIPLEXER AND COMPARATOR BOARD (Card 2)

Card no (2) contains the Pattern-Input-Memory and Pattern-Output-Memory along with latches and multiplexers which are controlled by the control circuitry on card no (1). This card also contains the digital comparators used in detecting the Trigger-Pattern used in the Logic-Analyser Mode.

The timing is all handled by card 1 and has been discussed earlier. There are four banks of memory as shown in Figure 15. These four banks, namely Pattern-Output-Memory bank 1 and 2, and Pattern-Returned-Memory bank 1 and 2 are on this card along with the multiplexers used in the memory multiplexing.

Tri-State drivers are also on this card and connect the memory to the MULTIBUS interface so that the Single-Board-Computer can write the Output-Patterns and read the Returned-Patterns.

The trigger generation circuitry on this card consists of four 8 bit cascaded digital comparators and this circuitry is totally asynchronous so that the trigger matching can have a very short matching time, relative to the sampling time.

The RAM used on this card is high speed static RAM with an access time of less than 100ns.

6.5 ANALOG INTERFACE BOARD (Card 3)

Card no (3) is the analog interface and takes the purely digital signals from card no (2) and interfaces with the Device-Under-Test. This card consists mainly of Analog Comparators, Tri-State drivers and voltage generation circuitry to control the thresholds. Dual Thresholds are used in the measurement of tri-state lines. Two digital inputs and two digital outputs from card (2) are combined in this card into one general purpose driver/sensor pin for driving and sensing high, low and tri-state. The thresholds could be changed for different logic families.

High speed analog comparators are used to compare the incoming signal with the threshold and generate the digital signals. The digital signals are output via tri-state drivers.

6.5.1 BI-STATE MODE

In this mode the switches are opened and the incoming lines are not loaded at all. The two thresholds are set at 1.6 Volts to detect high and low. The two incoming lines are basically buffered through the High Speed Analog Comparators.

The output is a more complex in this mode. The output lines are in pairs and they can only be driven together. The two outgoing digital lines are used to control tri-state and level of the two lines.

If the lines are driven into tri-state then there are effectively two input lines, however if the lines are driven high or low then two lines are both driven together and there is effectively an identical level on both lines.

There are 32 Bi-State Sensor pins or 16 Tri-State Driver pins supported by this card. For more details on this mode see the circuit diagram of the Analog Interface Card. For programming information see the previous chapter.

6.5.2 TRI-STATE MODE

In this mode the two switches associated with the line are closed. This effectively shorts the two lines and connects the resistor network in, which effectively pulls the line towards 1.6 volts. Under no load the line will be pulled to 1.6 volts and tri-state can be detected. Tri-State is detected by using dual thresholds one at 0.8 Volts and the other at 2.4 Volts. If the signal is above or below both then it is a normal high or low. However if the incoming signal is above 0.8 Volts and below 2.4 Volts then it is in tri-state.

The driver is designed particularly for this mode where the two digital output lines are used to control tri-state and level. The actual programming is detailed in the previous chapter.

The Driver/Sensor pin is effectively driven by two output lines and the state is sensed by two other lines. Two lines from the Pattern-Output-Memory drive the line, and the state of the line is sensed by two lines of the Pattern-Returned-Memory.

There are 16 Tri-State Driver/Sensor pins supported by this card.

6.5.3 CUSTOM ANALOG INTERFACES

Due to the fact that this board has a modular function, it is possible to design a custom analog interface for a particular application in mind. There are effectively 32 input and 32 output lines from this board to the Test-Hardware. Any analog board could be designed to interface between the Test-Hardware and the Device-Under-Test.

It is not necessary to use analog components. A simple board would be to just buffer the signals and effectively give 32 inputs and 32 outputs.

Another possibility would be to have flip-flops on this board which can be used for glitch detection. The possibilities are endless.

6.6 TIMING AND CONTROL CONNECTOR (Connector 1)

This connector is between the multibus interface and control board *(Card 1) and the Memory*,Latch, Multiplexer and Comparator Board (Card 2), and sends contol signals from the one board to the other.

Component Side (Top)		Solder Side (Bottom)	
1	Data1	2	Data0
3	Data3	4	Data2
5	Data5	6	Data4
7	Data7	8	Data6
9	Data9	10	Data8
11	DataB	12	DataA
13	DataD	14	DataC
15	DataF	16	DataE
17	ϕ1	18	Addr1
19	ϕ2	20	Addr2
21	ϕ3	22	Addr3
23	/WE1	24	Addr4
25	θ1	26	Addr5
27	θ2	28	Addr6
29	θ3	30	Addr7
31	/WE2	32	Addr8
33	~	34	Addr9
35	~	36	AddrA
37	~	38	AddrB
39	External Trigger	40	/MRD Memory Read
41	~	42	/MWT Memory Write
43	~	44	/EXINT External Internal
45	~	46	MXCTL Multiplexer Control
47	/TRIGGER	48	SPEED *Multiplexed or Not*
49	CLOCK1 Main Clock	50	CLOCK2 Auxiliary Clock

Table 16 : Timing and Control Connector

4.7 DIGITAL I/O CONNECTOR (Connector 2)

This Connector joins the Memory, Latch, Multiplexer and Comparator Board (Card 2) with the Analog Interface Board (Card 3) and the Trigger-Pattern-Switches. OUT n are outputs and IN n and PAT n are inputs. The trigger matches when for all n, IN n equals PAT n.

Component Side		Solder Side	
1	CLOCK1	2	CLOCK2
3	*Manual Trig*	4	-
5	OUT 25	6	OUT 17
7	OUT 26	8	OUT 18
9	OUT 27	10	OUT 19
11	OUT 28	12	OUT 20
13	OUT 29	14	OUT 21
15	OUT 30	16	OUT 22
17	OUT 31	18	OUT 23
19	OUT 32	20	OUT 24
21	OUT 9	22	OUT 1
23	OUT 10	24	OUT 2
25	OUT 11	26	OUT 3
27	OUT 12	28	OUT 4
29	OUT 13	30	OUT 5
31	OUT 14	32	OUT 6
33	OUT 15	34	OUT 7
35	OUT 16	36	OUT 8
37	PAT 18	38	IN 17
39	IN 18	40	PAT 17
41	PAT 20	42	IN 19
43	IN 20	44	PAT 19
45	PAT 22	46	IN 21
47	IN 22	48	PAT 21
49	PAT 24	50	IN 23

Component Side		Solder Side	
51	IN 24	52	PAT 23
53	PAT 26	54	IN 25
55	IN 26	56	PAT 25
57	PAT 28	58	IN 27
59	IN 28	60	PAT 27
61	PAT 30	62	IN 29
63	IN 30	64	PAT 29
65	PAT 32	66	IN 31
67	IN 32	68	PAT 31
69	PAT 2	70	IN 1
71	IN 2	72	PAT 1
73	PAT 4	74	IN 3
75	IN 4	76	PAT 3
77	PAT 6	78	IN 5
79	IN 6	80	PAT 5
81	PAT 8	82	IN 7
83	IN 8	84	PAT 7
85	PAT 10	86	IN 9
87	IN 10	88	PAT 9
89	PAT 12	90	IN 11
91	IN 12	92	PAT 11
93	PAT 14	94	IN 13
95	IN 14	96	PAT 13
97	PAT 16	98	IN 15
99	IN 16	100	PAT 15

Table 17 : Digital I/O Connector

6.8 ANALOG I/O CONNECTOR (Connector 3)

This connector is from the Analog Interface Board (Card 3) to the Device-Under-Test. The Voltage Signals are to set the Thresholds of the 4 banks.

Component Side (Top)				Solder Side (Bottom)			
1	I/O 1	A		2	I/O 2	B	
3	I/O 3	A		4	I/O 4	B	
5	I/O 5	A		6	I/O 6	B	
7	I/O 7	A		8	I/O 8	B	
9	I/O 9	A		10	I/O 10	B	
11	I/O 11	A		12	I/O 12	B	
13	I/O 13	A		14	I/O 14	B	
15	I/O 15	A		16	I/O 16	B	
17	I/O 17	C		18	I/O 18	D	
19	I/O 19	C		20	I/O 20	D	
21	I/O 21	C		22	I/O 22	D	
23	I/O 23	C	Double	24	I/O 24	D	Double
25	I/O 25	C	Speed	26	I/O 26	D	Speed
27	I/O 27	C		28	I/O 28	D	
29	I/O 29	C		30	I/O 30	D	
31	I/O 31	C		32	I/O 32	D	
33	-			34	-		
35	-			36	-		
37	Voltage A			38	Voltage A		
39	Voltage B			40	Voltage B		
41	Voltage C			42	Voltage C		
43	Voltage D			44	Voltage D		
45	-			46	-		
47	-			48	-		
49	CLOCK1 Main Clock			50	CLOCK2 Auxiliary Clock		

Table 18 : Analog I/O Connector

6.9 TRIGGER-PATTERN SWITCHES

There are 32 switches on the Trigger-Pattern-Switch-Box and they are connected to the DATA-Input and PATTERN-Input lines of connector 2. When the PATTERN-Input equals the DATA-Pattern then a trigger signal is generated.

Each switch has three positions: 1) High
2) Don't Care
3) Low

In Position 1 (High) the PATTERN-Input is connected high.
In Position 2 (Don't Care) the PATTERN-Input is connected to DATA-Input.
In Position 3 (Low) the PATTERN-Input is connected low.

There is also a manual trigger on the Trigger-Pattern-Switch-Box and this is for if the operator wants to trigger the Logic-Analyser Mode or the Triggered Tester Mode without the Trigger-Pattern matching the Returned-Pattern.

6.10 CONCLUSION

A description of the hardware developed for this project has now been covered. The detailed operation of the hardware can be obtained by examining the Circuit Diagrams, however this Chapter and the previous Chapter should be sufficient to allow the Test-Hardware to be used in a Test-System.

For using the Test-Hardware the previous chapter describes the programming information. The pinouts of the Analog I/O connector are also needed for connecting the Test-System to the Device-Under-Test.

The Test-Hardware was tested and debugged and the only problem was at the higher speeds the probes attached to the Device-Under-Test were of very poor quality and sometimes did not switch fast enough due to the capacitive loading. Otherwise the Test-Hardware worked and was demonstrated successfully.

Actual test results on the Test-Hardware are contained in Appendix N and Appendix O where the Test-Hardware was used to test examples of Devices-Under-Test. Appendix P contains the Hardware Bugs that were found and methods used to get around these problems.

Possible uses for the hardware developed and extensions will be discussed in the next chapter.

CHAPTER 7

POTENTIAL USES OF THE TEST-SYSTEM

7.1 INTRODUCTION

The Test-Hardware developed for this project was designed to be as general purpose as posssible. There are therefore a number of uses that the hardware can be used for, with the addition of suitable software or hardware.

With some of the uses a host computer is needed and this was available at the University in the form of the DOP 11 Mini Computer, or an Apple II + Micro Computer. Most uses need a terminal attached to the serial link on the 8086 Single-Board-Computer.

A Signature-Analyser probe is the simplest and probably the most useful hardware addition to the Test-System developed. This probe would be very useful in nearly all the methods described in this Chapter.

Three of the most obvious uses will be discussed here. There are possibly a lot more uses for the hardware, depending on the application at hand.

7.2 STAND ALONE LOGIC-ANALYSER

By connecting a Terminal to the serial link of the 8086 Single-Board-Computer, and writing suitable software in the PROM on the 8086 Single-Board-Computer it is easy to Configure the Test-Hardware as a Logic-Analyser.

The Test-Hardware provides all the functions to operate as a Logic-Analyser. The Pattern-Output-Memory is loaded with the pattern for driving the outputs into tri-state, and the Test-Hardware is placed into Logic-Analyser mode to stop when the Trigger-Pattern matches the Returned-Pattern.

Appendix D gives an example of the hardware being used as a Logic-Analyser. For ease of use, software can be developed to make the interface between man and machine more user friendly. This software would make the Test-System easy to use and very useful in the laboratory as a general purpose Logic-Analyser. For use as a Logic-Analyser, a good display of the timing diagrams needs a graphics terminal, or one with either a programmable or special character set.

In the University environment Logic-Analysers are always needed and it will be useful to develop the necessary software, and put on probes of sufficiently high quality to use the Test-System as a Logic-Analyser.

7.3 STAND ALONE DYNAMIC-FUNCTIONAL-TESTER

The Test-Hardware can be programmed to operate as a Stand Alone Dynamic-Functional-Tester, with the addition of suitable software. It would however be necessary to add some type of non-volatile peripheral storage to the Test-System, in the form of a Floppy Disk Drive or a Tape Drive. Another necessary addition is the addition of a terminal to the serial link on the 8086 Single-Board-Computer.

The peripheral storage is necessary to store the Test-Pattern, and the Expected and Returned-Results. The Terminal is needed to interface to the operator.

The addition of a Signature-Analyser would be very useful in the tracing of faults. This addition would allow the Test-System to operate as a Signature-Analysis System *where the Test-Hardware Developed would be the* pattern application part of the System.

7.3.1 TEST-PATTERN GENERATION

The Test-Program consists of applying a Test-Pattern to the Device-Under-*Test and measuring the Actual-Response*. The Actual-Response is then compared with the Expected-Response to see if they match.

The Test-Pattern can be generated by an Engineer writing a program on the Test-System. Software could be written on the Test-System to aid in this task.

Another possibility is that of having a pseudo random pattern generator generate the patterns. With this method it is necessary to be able to initialise the Device-Under-Test to a known state and this initialisation could be done by writing a program similar to the manual method, where the whole Test-Procedure is written by an engineer. However initialisation may be simple, so this method would make writing the Test-Program easier.

Lastly it may be possible to load the Test-Pattern down from some host computer, via the serial link, or by transferring the patterns on Floppy Disk, or whatever non volatile media is used.

As a stand alone Test-System it must be able to store the Test-Pattern and the Expected-Response so that the validity of boards can be checked in the Stand Alone Mode.

7.3.2 EXPECTED-RESPONSES

The Expected-Response can be obtained by loading the Expected-Response down from a host computer, like the Test-Pattern. The Host Computer would generate the Expected-Response using a Logic-Simulator or similar software package. The Dictionary-Of-Faults could also be loaded down to the Stand Alone Test-System.

Another method would be to use a Known-Good-Board to generate the Expected-Response. This Known-Good-Board can also be used to generate a Dictionary-Of-Faults, by injecting various faults and storing the Actual-Response to these faults.

7.4 SLAVE DYNAMIC-FUNCTIONAL-TESTER

The Test-Hardware can be operated as a Slave Dynamic-Functional-Tester by the addition of suitable communications software. The actual intelligence would remain in the Host Computer and the Slave Tester would only act on command from the Host. The Test-Hardware can be connected to the host by a serial or parallel link on the 8086 Single-Board-Computer.

No intelligence need reside in the Test-System and all processing can be carried out by the host. However the more intelligence that resides in the Slave the less traffic there will be between the Host and the Slave. Too little intelligence in the slave may mean that operations will be slowed down by the need to transfer vast amounts of information. It would be better for instance if the slave compared the Expected-Response with the Actual-Response and reported any differences, rather than sending up the Actual-Response in its entirety.

As with the previous method a Signature-Analyser would be a valuable addition to the Hardware.

7.4.1 TEST-PATTERN GENERATION

There are no restrictions to the method of Test-Pattern generation. It is possible to use nearly any method depending on the software available.

Automatic Test-Pattern generation could be done here with the addition of suitable software on the Host Computer. This would be very useful however Automatic Test-Pattern Generation gets very difficult with VLSI components as the library parts are extremely difficult to generate.

The Manual method of an engineer programming the Test-Program is another possibility. This method has its advantages and disadvantages as discussed in chapter 4.

Lastly it is possible to have a pseudo random pattern generator to

generate the Test-Patterns. However with this method the initialisation problem with complex sequential circuits has to be dealt with.

Nearly any method can be used in the generation of the Test-Patterns.

7.4.2 EXPECTED-RESPONSES

The Expected-Response and the Dictionary-Of-Faults can be generated either by a Logic-Simulator or a Known-Good-Board. A third possibility is to manually generate the Expected-Response, however this is very tedious and error prone.

The pros and cons of Logic-Simulation versus the Known-Good-Board are discussed in chapter 3. One thing that must be kept in mind is that Logic-Simulation can be done as a background task without tying up the Test-Hardware in any way.

7.5 CONCLUSION

The hardware developed is general purpose and can be used in a number of different test applications. Most applications are possible by the addition of suitable software.

The addition of a Signature-Analyser to the hardware would increase the flexibility and usefulness of the Test-System considerably.

If the hardware were ever to be put into production, a number of modifications would need to be made. One main modification would be the use of multi-layer printed circuit boards. It was necessary to have numerous jumpers to overcome the routing problem found on a double layer board without through hole plating.

The addition of proper high speed probes would also be needed for this Test-System.

CHAPTER 8

CONCLUSION

8.1 INTRODUCTION

The field of Automatic-Test-Equipment is very competitive and complex field with vast sums being spent on development and very costly systems being developed. To compete with the industry leaders in Automatic-Test-Equipment was not the aim. The aim was to produce the Hardware framework of an Automatic-Test-System which would be relatively cheap and easy to produce, while at the same time being as flexible in use and application as possible. The previous chapter describes three possible applications of the Test-Hardware.

The Test-Hardware developed for this project was demonstrated successfully. Appendix N and Appendix O are actual examples of tests run on the Test-System.

This project report contains information on the reasons for developing the Test-Hardware as well as a description from a functional software point of view as well as a detailed hardware description of the Test-System.

8.2 IMPROVEMENTS

There are a number of improvements that could be made to the Test-System but only the more obvious ones will be mentioned here. All systems can be improved so these points should not be looked on as limitations, but rather as possible paths of improvement.

8.2.1 SIGNATURE-ANALYSER ADDITION

As mentioned in the previous chapter the most useful hardware addition would probably be that of a Signature-Analyser probe. This probe is relatively simple and can be added easily to the present hardware.

The addition of a Signature-Analyser would make the tracing of faults very much easier. Fault tracing with a Signature-Analyser is a lot simpler than without one.

8.2.2 CASCADING HARDWARE

Increasing the number of Driver/Sensor Pins can be done by the addition of cascading signals to the Test-Hardware. The number of pins can be doubled by duplicating the hardware and cascading two Test-Systems together.

The number of Driver/Sensor pins could be increased in this way to whatever number was needed by the addition of a suitable number of Test-Systems.

8.3 HARDWARE BUGS

After the circuit was designed and tested it was noted that there were two small bugs. Neither of the problems actually caused difficulties in the working of the Test-Hardware as they were very easy to circumvent.

8.3.1 PULL-UP RESISTOR BUG

Component No 24 (74LS266) on the Interface Control Board which is an open collector gate had the Pull-Up resistors omitted. This component was in the addressing circuitry of the MULTIBUS interface. It was possible to get around this problem by delaying the Transfer Acknowledge (XACK) signal. The delay is switch selectable. If pullup resistors were added then the Transfer Acknowledge could be made faster.

8.3.2 WORD ADDRESSING

It is necessary to address the Control/Status-Register and the Dedicated Memory as words on even boundaries due to a fault in the design of the internal addressing. This was not a problem as such but improved versions of the board should have the ability to access the memory at byte addresses.

8.4 CONCLUSION

Up to the point that it was developed, the Test-Hardware Developed worked successfully. The main factors which were kept in mind when developing the hardware were low cost and flexibility.

When developing or buying an Automatic-Test-System it must be realised that generally the more functions required the more it will cost. The Test-System developed for this project hopefully gives one as many functions as possible for the cost incurred.

REFERENCES

[1] BEDARD, A. R. (Pylon Company, USA)
Bed-of-nails test fixtures
Pulse August 1982, Pages 82-83

[2] BLYTH, G. (Marconi Instruments, UK)
Automatic testing of digital p.c.b. assemblies
Pulse August 1982, Pages 87-98

[3] BREUER, M. A. and FRIEDMAN, A. D. (Univ. Southern California)
Diagnosis & Reliable Design of Digital Systems
Pitman Publishing Limited 1976

[4] HENCKELS, L. P. and HASS, R. M. (GenRad, USA)
The Known-good-board approach versus software simulation
Pulse August 1982, Pages 92-94,97

[5] *INTEL*
iSBC 957A Intellec - iSBC86/12A Interface and Execution Package User's Guide
Manual Order Number 142849-001 (1980)

[6] INTEL
iSBC 86/12A Single Board Computer Hardware Reference Manual
Manual Order Number *9803874-01 (1979)*

[7] McDERMOTT, R. (International Telephone & Telegraph)
Simulation of Simple Digital Logic through a Computer-Aided Design System
Byte Publications, Volume 8 No 1, January 1983, Pages 396-414

[8] McDERMOTT, R. (International Telephone & Telegraph)
The Design of an Advanced Logic Simulator
Byte Publications, Volume 8 No 4, April 1983, Pages 398-438

[9] MUEHLDORF, E. I. (Senior member IEEE) and SAVKAR, A. D.
LSI Logic Testing - An Overview
IEEE Transactions on Computers, Vol C30, January 1981, Pages 1-16

[10] POPKY, A. (Millenium Systems)
Function testing by in-circuit emulation
Pulse August 1982, Pages 98-106

BIBLIOGRAPHY

[1] BEDARD, A. R. (Pylon Company, USA)
Bed-of-nails test fixtures
Pulse August 1982, Pages 82-83

This article covers the basic mechanical considerations that must be taken into account with this type of test fixture.

[2] BELT, E. and KOLE, R. (Fairchild, USA)
Testing microprocessors. A comparison of four techniques.
Pulse August 1982, Pages 110-115

This article covers four different methods of testing microprocessor based systems, namely: In-System-Testing, Comparison-Testing, Augmented Comparison-Testing, and General-Purpose-Testing.

[3] BENNETTS, R. G.
The Diagnosis of Logical Faults
Wireless World, July 1971

This article discusses the basic differences between Combinational and Sequential Circuits and discusses the method of Path Sensitisation.

[4] BENNETTS, R. G.
The Diagnosis of Logical Faults
Wireless World, August 1971

This brief article discusses Boolean Difference and Partitioning Techniques. It is the conclusion to the previous article above.

[5] BLUESTONE, A. (Adar Associates, Massachusetts)
Logical Environment Comparison Testing Handles Complex LSI Devices
Computer Design, April 1979

This article gives an overview of Digital-Testing and covers *in detail the comparison method where* the Device-Under-Test is compared with a Known-Good-Board.

[6] BLYTH, G. (Marconi Instruments, UK)
Automatic testing of digital p.c.b. assemblies
Pulse August 1982, Pages 87-98

This is one of the best articles on Automatic-Digital-Circuit-Testing. It gives a good summary of the various methods used in testing digital circuits. A good discussion of the different types of Test-Hardware along with the pros and cons is given.

[7] BOURICIUS, W. G.(IBM), HSIEH, E. P.(IBM), PUTZOLU, G. R.(IBM), ROTH, J. P.(IBM), SCHNEIDER, P. R.(IBM), TAN, C.(IBM)
Algorithms for Detection of Faults in Logic Circuits
IEEE Transactions on Computers, Vol C-28, November 1971, Pages 1258-1264

This article presents the D-Algorithm for generating Test-Patterns for combinational circuits. This is a fairly technical article.

[8] BREUER, M. A. and FRIEDMAN, A. D. (Univ. Southern California)
Diagnosis & Reliable Design of Digital Systems
Pitman Publishing Limited 1976

This is a very useful book in Digital Circuit Testing. It covers the theory involved in generating Test-Patterns for Combinational and Sequential Circuits. Logic-Simulation and *Reliable Design Theory* are also discussed. This book looks mainly at the theory involved and does not cover any actual Test-Hardware examples.

Methods of test generation for combinational circuits discussed are : Boolean Difference, Critical Path, Path Sensitisation and the D-algorithm. Fault Equivalence, *Dominance and Collapsing* as well as Test Reduction and Minimisation are also covered for Combinational Circuits.

For Sequential circuits the D-algorithm and Critical Path Algorithm are extended. Synchronous and Asynchronous sequential circuits are discussed. Under sequential circuits RAM tests are covered in depth.

Logic-Simulators are discussed in detail. Circuit delays and Multi-valued logic are discussed. Fault Simulation and Simulation oscillation are also discussed.

Under the heading of Reliable Design, Self-Checking Circuits, Fault-Tolerant Design and designs to *simplify testing* are discussed.

[9] BROCK, G. (Gould Inc, *USA*)
Logic analyser trace functions
Pulse November 1982, Pages 84-88

Interesting article about an advanced Logic-Analyser. Useful since Test-Hardware is very similar to that of a Logic-*Analyser*.

[10] CHAN, A. Y. (Hewlett Packard)
Easy-to-Use Signature Analyser Accurately Troubleshoots Complex Logic Circuits
Hewlett Packard Journal, Pages 9-14

This article discusses the Electrical Details of a Signature-Analyser. Very good article on the subject.

[11] DONNELLY, B. (Electronics Industry, London)
Spotlight on automatic board testing
Pulse August 1982, Pages 76-88

This article is basically a summary of the features of all the various board testing equipment available. All sorts of technical terms and capabilities are quoted with very little explanation as to what they actually means.

[12] FROHWERK, R. A. (Hewlett Packard)
Signature Analysis: A New Digital Field Service Method
Hewlett Packard Journal, Pages 2-8

This article covers the theory of Signature-Analysis. The author did the theoretical work on Signature-Analysis.

[13] GROVES, A. G. (Hewlett Packard)
Rapid Digital Fault Isolation with *FASTRACE*
Hewlett Packard Journal, March 1979, Pages 8-13

This article discusses the basic operation of a particular tester. It does not go into *how Test-Programs are generated* but basically describes what the tester can do on the Test-Program has been generated.

[14] HARVEY, K. P. (Fluke SA)
Fluke automated test systems in South Africa
Pulse August 1982, Pages 104-107

This article is basically an advertisement for Fluke. The *criticisms* of other methods is unconvincing and none of the obvious shortcomings of the Fluke method are mentioned. Seems to skip over obvious problems in their methods.

[15] HENCKELS, L. P. and HASS, R. H. (GenRad, USA)
The Known-good-board approach versus software simulation
Pulse August 1982, Pages 92-94,97

Good article which first describes how each method is used in detail and then discusses the pros and cons of each method. The article is not biased towards any particular method.

[16] INTEL
iSBC 957A Intellec - iSBC86/12A Interface and Execution Package User's Guide
Manual Order Number 142849-001 (1980)

This manual covers the monitor used in the 86/12A Single-Board-Computer.

[17] INTEL
iSBC 86/12A Single Board Computer Hardware Reference Manual
Manual Order Number 9803874-01 (1979)

This manual covers the use of the 8086 based Single-Board-Computer used in the Test-Hardware.

[18] [illegible]WITT, R. (International Telephone & Telegraph)
Simulation of Simple Digital Logic through a Computer-Aided Design System
Byte Publications, Volume 8 No 1, January 1983, Pages 396-414

This article covers the actual implimentation of a simple Table Driven Logic-Simulator. It covers the implimentation in detail and actually includes the listing in BASIC. Very good article to actually get to grips with what is involved in Logic-Simulators.

[19] McDERMOTT, R. (International Telephone & Telegraph)
The Design of an Advanced Logic Simulator
Byte Publications, Volume 8 No 4, April 1983, Pages 398-438

This article covers the design of an Advanced Logic-Simulator which has Named Nodes, Macrocircuit Capability, Error Checking, Selective Trace and Sampled Output. Primitive elements include all the basic combinational gates, and the sequential gates like the D and J-K type Flip Flops, an N Bit Counter, an N Bit Shift-Register, and a Parallel load Shift-Register. Inputs may be periodic or aperiodic. This article actually contains the listing of this program in BASIC and this is the one that was implemented on my Nascom 1 with various improvements. Very usful article about Table Driven Simulators.

[28] MOON, D. L. (University of Dayton)
Bipolar Logic Test Pattern Generation
Computer Design, March 1971, Pages 63-73

This article basically covers the method of Path Sensitisation for pattern generation and fault detection and verification.

[21] MUEHLDORF, E. I. (Senior member IEEE) and SAVKAR, A. D.
LSI Logic Testing - An Overview
IEEE Transactions on Computers, Vol C30, January 1981, Pages 1-16

Very useful overview of the theory involved in fault folding and automatic Test-Pattern generation using the various methods. Some of the examples have very terse explanations and are difficult to follow but on the whole an excellent article.

[22] NADIG, H. J. (Hewlett Packard)
Signature Analysis - Concepts, Examples and Guidelines
Hewlett Packard Journal, Pages 15-21

Useful overview on the use of Signature-Analysis. This article describes all that is needed for the use of Signature-Analysis. An example is included in this article.

[23] PARKER, K. P. (Hewlett Packard)
Software Simulator Speeds Digital Board Test Generation
Hewlett Packard Journal, March 1979, Pages 13-19

This article basically describes the features of a particular Digital-Simulator. This is basically a Table-Driven-Simulator which has various features like Fault Simulation, and the ability to detect races and hazards. This article mentions the differences between Compiled code and Table-Driven-Simulators.

[24] POPKY, A. (Millenium Systems)
Function testing by in-circuit emulation
Pulse August 1982, Pages 98-100

This is a good article on the advantages of using In-Circuit-Emulation for testing microprocessor systems or boards that plug into a microprocessor bus. Some interesting facts are put forward.

[25] RAYMOND, D. W. (Plantronics/Zehntel, California)
Component-By-Component Testing of Digital Circuit Boards
Computer Design, April 1988, Pages 129-137

This article covers functional testing, in-circuit testing, and Signature-Analysis. It covers the disadvantages and advantages of these methods, and is a very interesting article.

[26] ROTH, J. P. (IBM)
Diagnosis of Automata Failures: A Calculus and a Method
IBM Journal, July 1966, Pages 278-291

This article precisely describes the D-algorithm used to compute tests and analyse faults in a digital circuit. Fairly comprehensive article on this method, with examples.

[27] ROTH, J.P.(IBM), BOURICIUS, W.G.(IBM) and SCHNEIDER, P.R.(IBM)
Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits
IEEE Transactions on *Electronic Computers*, Vol EC-16, October *1967*, *Pages* 567-580

This article presents two algorithms for Test-Pattern generation and fault diagnosis, namely the *DALG-II* and *TEST-DETECT* which are both based on the calculus of D-cubes used in the D-*algorithm*.

[28] SCHNURMANN, H. D.(Mem IEEE), LINDBLOOM, E. and CARPENTER, R.G.
The Weighted Random Test-Pattern Generator
IEEE Transactions on Computers, Vol C-24, July 1975, Pages 695-700

This article covers the Random Test-Pattern *Generator*. It covers the *purely random pattern*, the weighted adaptive *patterns*, and the dynamic adaptive patterns. Pattern reduction techniques are also discussed.

[29] SCHOFIELD, M. (Wayne Kerr, UK)
Automatic digital diagnostics extend bench-top ATE capability
Pulse August 1982, Pages 95-97

This article basically describes the Wayne Kerr A8880 System which stimulates the board under test and uses Signature-Analysis to detect faults. The Test-Program is generated using a good board and by remembering the various signatures.

[30] SELLERS, F. F. (Jr mem IEEE), HSIAO. M. Y, and BEARNSON, L. W.
Analyzing Errors with the Boolean Difference
IEEE Transactions on Computers, Vol C-17, July 1968, Pages 676-683

This article covers how the boolean difference method is used to analyse the effect of errors on the output. Fairly technical article.

[31] SU, S. Y. H. (Member IEEE) and CHO, Y. (Student member IEEE)
A New Approach to the Fault Location of Combinational Circuits
IEEE Transactions on Computers, Vol C-21, January 1972, Pages 21-38

This article covers the D-algorithm of generating Test-Patterns and diagnosing faults. Fairly comprehensive article on this method.

[32] SUSSKIND, A. K. (Lehigh University)
Diagnostics for logic networks
IEEE Spectrum, October 1973, Pages 48-47

This article is a bit outdated as it covers the Algebraic Expression method, which is only good for networks of modest size, typically hundreds of gates.

[33] TAYLOR, E. (Specom)
Evaluating board testers
Pulse August 1982, Page 188

Short fair article on the economic and other factors one must take into account when buying an board tester.

[34] THOMAS, J. J. (LTV Aerospace Corporation, Dallas, Texas)
Automated Diagnostic Test Program for Digital Networks
Computer Design, August 1971, Pages 63-67

This article describes a total Testing-System and is mainly about the software in the system. This is a good overview of a system.

[35] THOMAS, J. J. (Digitest Corporation, Dallas, Texas)
Common Misconceptions in Digital Test Generation
Computer Design, January 1977, Pages 89-94

This is a useful review of the capabilities of Automatic-Testers at that time. Rather pessimistic view for testers.

[36] TO, K. (Western Electric, USA)
Fault Folding for Irredundant and Redundant Combinational Circuits
IEEE Transactions on Computers, Vol C-22, November 1973, Pages 1008-1015

This article is very mathematical article about fault folding. It is definately not light reading and contains proofs and theorems.

[37] YAU, S. S. (Senior Mem IEEE) and TANG, Y. (Student Mem IEEE)
An Efficient Algorithm for Generating Complex Test Sets for Combinational Logic Circuits
IEEE Transactions on Computers, Vol C-20, November 1971, Pages 1245-1251

This article basically covers the theory of Boolean *Difference* for generating Test-Patterns. This is a *very* technical article.

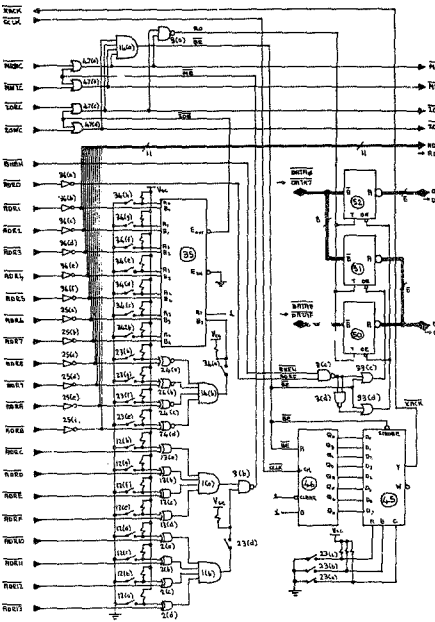
APPENDIX A

TEST-HARDWARE PRINTED CIRCUIT DIAGRAMS

MULTIBUS INTERFACE AND CONTROL BOARD (Card 1)

This card contains the MULTIBUS Interface, the Clock and Timing Control and the Addressing and Mode Control Circuitry. The list of components appears below followed by the circuit diagrams.

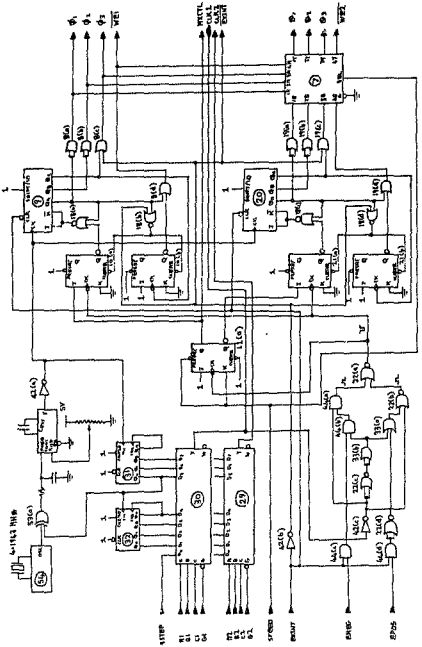
Number	Component	Description
3	74LS00	Quadruple 2-Input Positive NAND Gate
18,22	74S02	Quadruple 2-Input Positive NOR Gate
42	74S04	Hex Inverters
25,36	74LS04	Hex Inverters
44	74S08	Quadruple 2-Input Positive AND Gate
16	74LS08	Quadruple 2-Input Positive AND Gate
1,14	74LS21	Dual 4-Input Positive AND Gate
8,19,33	74S32	Quad 2-Input Positive OR Gate
47	74LS32	Quad 2-Input Positive OR date
53	74S86	Quadruple 2-Input EXCLUSIVE-OR Gate
2,13	74LS86	Quadruple 2-Input EXCLUSIVE-OR Gate
10,11,15,21	74S112	Dual J-K Negative Edge Flip-Flops
43	74S124	Dual Voltage Controlled Oscillators
7,26,37,48	74S157	Quad 2- to 1-Line Data selectors / MUX's
4,5,6,27,38,49	74S163	Binary 4 Bit Counter with Synchronous Clear
46	74164	8 Bit Parallel Output Serial Shift Register
9,28	74S195	4-Bit Parallel Access Shift Registers
31,32	74S197	Binary Presettable Asynchronous Counter
26	74LS244	Octal Buffer Line Drivers / Receivers
45	74S251	8- to 1-Line Data Selector / MUX
29,30	74LS251	8- to 1-Line Data Selector / MUX
24	74LS266	Quadruple 2-Input EXCLUSIVE-NOR Gates (OC)
17,39	74LS373	Octal D-Type Latches with Tri-State Outputs
40	74S374	Octal D-Type Flip-Flops
41	74LS374	Octal D-Type Flip-Flops
54	i8224	Crystal Clock Driver
50,51,52	i8287	Inverting Octal Bus Transceiver
35	25LS2521	8 Bit Digital Comparator
12,23,34	SWITCH	8 Dual In line switches



MULTIBUS INTERFACE

CARD 1

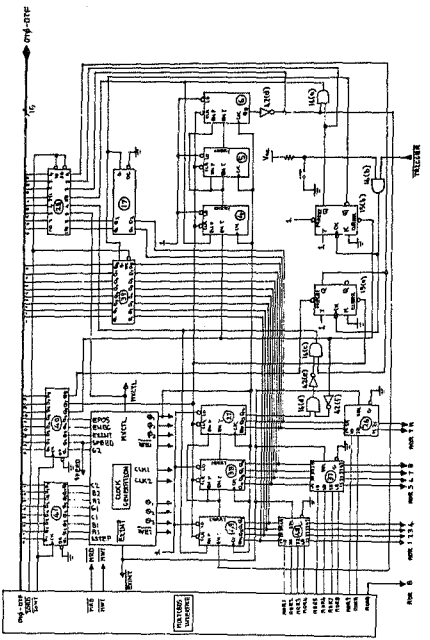
MULTIBUS INTERFACE AND CONTROL BOARD



CLOCK GENERATION

CARD 1

MULTIBUS INTERFACE AND CONTROL BOARD



CONTROL

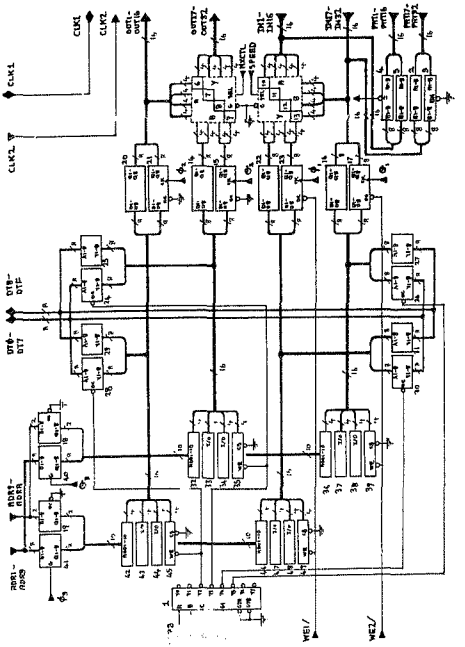
CARD 1

MULTIBUS INTERFACE AND CONTROL BOARD

MEMORY, LATCH, MULTIPLEXER AND COMPARATOR BOARD (Card 2)

This card contains the 4 banks of memory along with the Input and Output Latches, the Multiplexers used in multiplexed memory access, the digital comparators used for the Trigger-Pattern and Tri-State drivers to connect the memory to the bus interface. The list of components appears below followed by the circuit diagram.

Number	Component	Description
1	74LS138	3-To-8 Line Decoders / Multiplexers
6 -> 13	74S157	Quad 2- To 1-Line Data Selectors / MUX's
24 -> 31	74LS244	Octal Buffers / Line Drivers/Line Receivers
18,19,40,41	74S373	Octal D-Type Latches
14 -> 17,28 -> 23	74S374	Octal D-Type Flip-Flops
2 -> 5	2*LS2521	8 Bit Digital Comparator
32 -> 39,42 -> 49	HM6148-LP	1024-word X 4-bit High Speed CMOS RAM



CHAU 2.

MEMORY, LATCH, MULTIPLEXER AND COMPARATOR BOARD

ANALOG INTERFACE BOARD (Card 3)

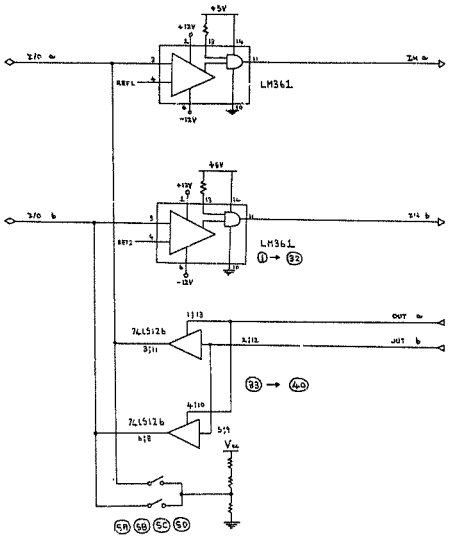
This card contains the high speed analog comparators, the tri-state drivers, the switches to select bi-state or tri-state, and four variable voltage sources for the thresholds. The numbering of the analog comparators on this board refers to the bit that they input. The list of components appears below followed by the circuit diagram.

Number	Component	Description
33 -) 48	74LS126	Quadruple Buffers with Tri-State Outputs
1 -) 32	LM361	High Speed Differential Comparator
41 -) 44	LM1088	Programmible Power Op Amp
SA,SB,SC,SD	SWITCHES	8 Dual In Line switches

ANALOG INTERFACE BOARD (Card 3)

This card contains the high speed analog comparators, the tri-state drivers, the switches to select bi-state or tri-state, and four variable voltage sources for the thresholds. The numbering of the analog comparators on this board refers to the bit that they input. The list of components appears below followed by the circuit diagram.

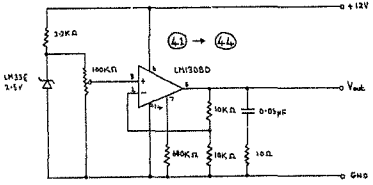
Number	Component	Description
33 -) 48	74LS126	Quadruple Buffers with Tri-State Outputs
1 -) 32	LM361	High Speed Differential Comparator
41 -) 44	LM3080	Programmible Power Op Amp
SA,SB,SC,SD	SWITCHES	8 Dual In Line switches



1 BIT TRI-STATE I/O

CARD 3

ANALOG INTERFACE BOARD



VOLTAGE REFERENCES

CARD 3

ANALOG INTERFACE BOARD

APPENDIX B

TEST-HARDWARE BOARD LAYOUTS

MULTIBUS INTERFACE AND CONTROL BOARD (Card 1)

This card contains the MULTIBUS Interface, the Clock and Timing Control and the Addressing and Mode Control Circuitry.

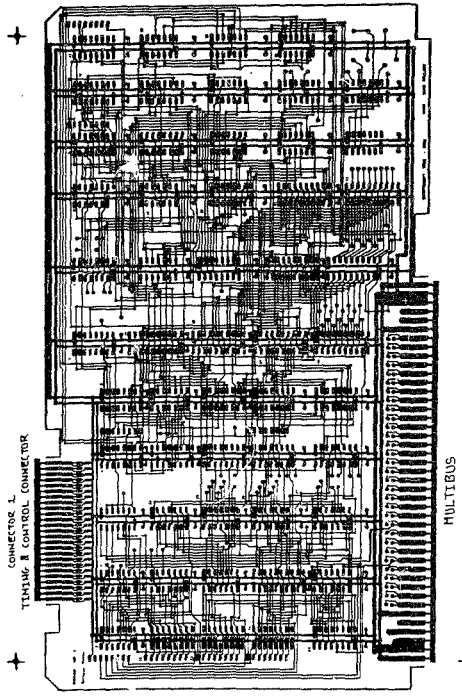
The actal layouts of this card are now given. All the layouts given are looking down at the card from the component side.

The three layouts given are as follows :-

1) Component Side (Contains Component Numbering)
2) Solder Side
3) Both Component and Solder Side Superimposed

MULTIBUS INTERFACE AND CONTROL BOARD

MULTIBUS INTERFACE AND CONTROL BOARD



BOTH SIDES CARD 1

MULTIBUS INTERFACE AND CONTROL BOARD

MEMORY, LATCH, MULTIPLEXER AND COMPARATOR BOARD (Card 2)

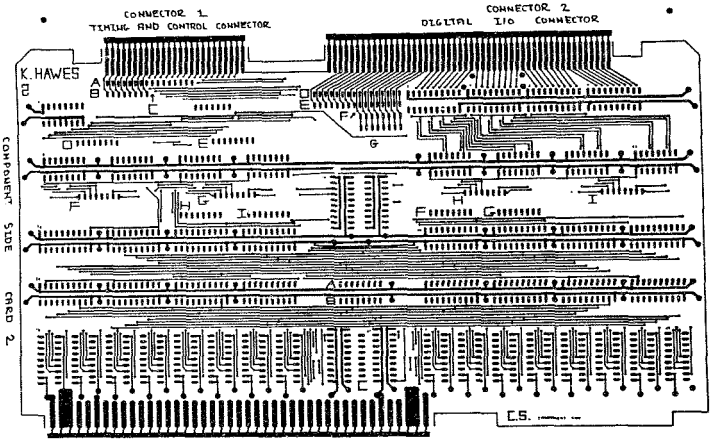
This card contains the 4 banks of memory along with the Input and Output Latches, the Multiplexers used in multiplexed memory access, the digital comparators used for the Trigger-Pattern and Tri-State drivers to connect the memory to the bus interface.

The actal layouts of this card are now given. All the layouts given are looking down at the card from the component side.

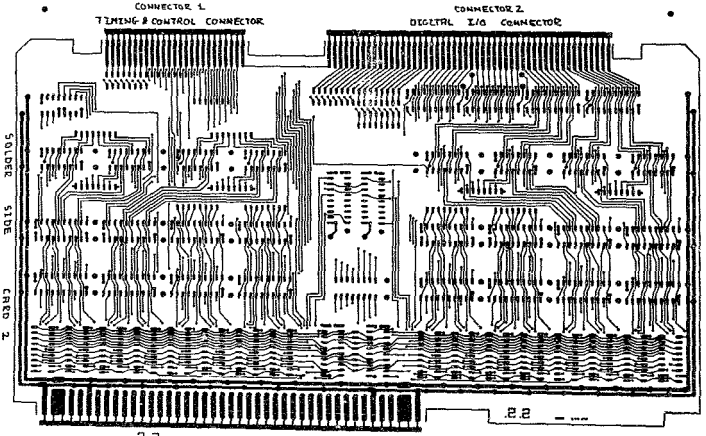
The three layouts given are as follows :-

1) Component Side (Contains Component Numbering)
2) Solder Side
3) Both Component and Solder Side Superimposed

MEMORY, LATCH, MULTIPLEXER AND COMPARITOR BOARD

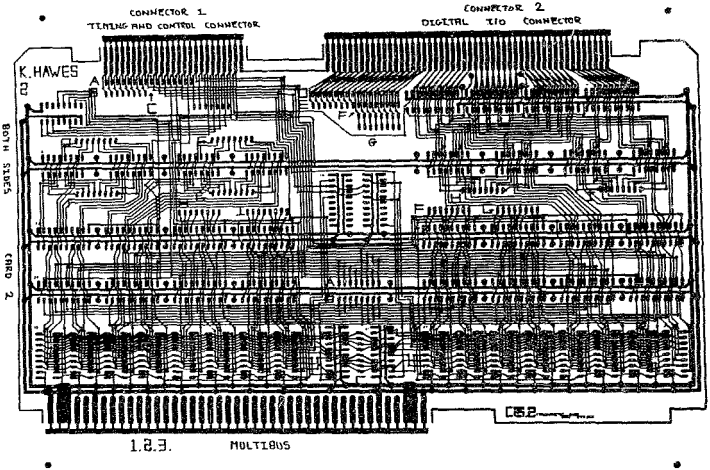


1.2. MULTIBUS


CONNECTOR 1.
TIMING & CONTROL CONNECTOR
CONNECTOR 2.
DIGITAL I/O CONNECTOR
SOLDER SIDE
CARD 2.
2.2

2.3. MULTIBUS

MEMORY LATCH MULTIPLEXER AND COMPARATOR BOARD



ANALOG INTERFACE BOARD (Card 3)

This card contains the high speed analog comparators, the tri-state drivers, the switches to select bi-state or tri-state, and four variable voltage sources for the thresholds. The numbering of the analog comparators on this board refers to the bit that they input.

The actal layouts of this card are now given. All the layouts given are looking down at the card from the component side.

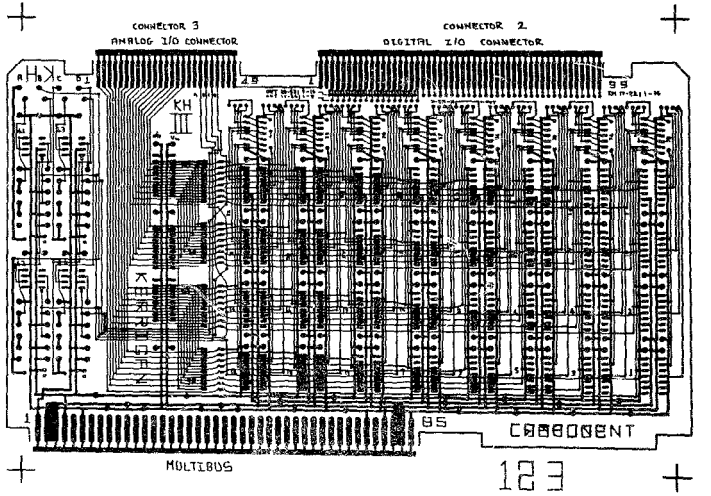
The three layouts given are as follows :-

1) Component Side (Contains Component Numbering)
2) Solder Side
3) Both Component and Solder Side Superimposed

ANALOG INTERFACE BOARD

ANALOG INTERFACE BOARD

ANALOG INTERFACE BOARD



APPENDIX C

CONNECTOR PINOUTS

Timing and Control Connector

Component Side (Top)		Solder Side (Bottom)		
1	Data1	2	Data0	
3	Data3	4	Data2	
5	Data5	6	Data4	
7	Data7	8	Data6	
9	Data9	10	Data8	
11	DataB	12	DataA	
13	DataD	14	DataC	
15	DataF	16	DataE	
17	Ø1	18	Addr1	
19	Ø2	20	Addr2	
21	Ø3	22	Addr3	
23	/WE1	24	Addr4	
25	Ø1	26	Addr5	
27	Ø2	28	Addr6	
29	Ø3	30	Addr7	
31	/WE2	32	Addr8	
33	-	34	Addr9	
35	-	36	AddrA	
37	-	38	AddrB	
39	External Trigger	40	/MRD	Memory Read
41	-	42	/MWT	Memory Write
43	-	44	/EXINT	External Internal
45	-	46	MXCTL	Multiplexer Control
47	/TRIGGER	48	SPEED	Multiplexed or Not
49	CLOCK1 Main Clock	50	CLOCK2	Auxiliary Clock

Digital I/O Connector

Component Side		Solder Side			Component Side		Solder Side	
1	CLOCK1	2	CLOCK2		51	IN 24	52	PAT 23
3	Manual Trig	4	-		53	PAT 26	54	IN 25
5	OUT 25	6	OUT 17		55	IN 26	56	PAT 25
7	OUT 26	8	OUT 18		57	PAT 28	58	IN 27
9	OUT 27	10	OUT 19		59	IN 28	60	PAT 27
11	OUT 28	12	OUT 20		61	PAT 30	62	IN 29
13	OUT 29	14	OUT 21		63	IN 30	64	PAT 29
15	OUT 30	16	OUT 22		65	PAT 32	66	IN 31
17	OUT 31	18	OUT 23		67	IN 32	68	PAT 31
19	OUT 32	20	OUT 24		69	PAT 2	70	IN 1
21	OUT 9	22	OUT 1		71	IN 2	72	PAT 1
23	OUT 10	24	OUT 2		73	PAT 4	74	IN 3
25	OUT 11	26	OUT 3		75	IN 4	76	PAT 3
27	OUT 12	28	OUT 4		77	PAT 6	78	IN 5
29	OUT 13	30	OUT 5		79	IN 6	80	PAT 5
31	OUT 14	32	OUT 6		81	PAT 8	82	IN 7
33	OUT 15	34	OUT 7		83	IN 8	84	PAT 7
35	OUT 16	36	OUT 8		85	PAT 10	86	IN 9
37	PAT 18	38	IN 17		87	IN 10	88	PAT 9
39	IN 18	40	PAT 17		89	PAT 12	90	IN 11
41	PAT 20	42	IN 19		91	IN 12	92	PAT 11
43	IN 20	44	PAT 19		93	PAT 14	94	IN 13
45	PAT 22	46	IN 21		95	IN 14	96	PAT 13
47	IN 22	48	PAT 21		97	PAT 16	98	IN 15
49	PAT 24	50	IN 23		99	IN 16	100	PAT 15

Analog I/O Connector

Component Side (Top)				Solder Side (Bottom)			
1	I/O 1	A		2	I/O 2	B	
3	I/O 3	A		4	I/O 4	B	
5	I/O 5	A		6	I/O 6	B	
7	I/O 7	A		8	I/O 8	B	
9	I/O 9	A		10	I/O 10	B	
11	I/O 11	A		12	I/O 12	B	
13	I/O 13	A		14	I/O 14	B	
15	I/O 15	A		16	I/O 16	B	
17	I/O 17	C		18	I/O 18	D	
19	I/O 19	C		20	I/O 20	D	
21	I/O 21	C		22	I/O 22	D	
23	I/O 23	C	Double	24	I/O 24	D	Double
25	I/O 25	C	Speed	26	I/O 26	D	Speed
27	I/O 27	C		28	I/O 28	D	
29	I/O 29	C		30	I/O 30	D	
31	I/O 31	C		32	I/O 32	D	
33	-			34	-		
35	-			36	-		
37	Voltage A			38	Voltage A		
39	Voltage B			40	Voltage B		
41	Voltage C			42	Voltage C		
43	Voltage D			44	Voltage D		
45	-			46	-		
47	-			48	-		
49	CLOCK1 Main Clock			50	CLOCK2 Auxiliary Clock		

APPENDIX D

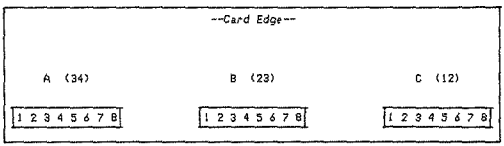
MULTIBUS INTERFACE SWITCH SELECTION / MEMORY MAP

Default Memory Map

Dedicated I/O Memory : BANK 1 0:8000H -> 0:87FFH
BANK 2 0:8800H -> 0:8FFFH

Control/Status-Register Address : 100H -> 101H

Switch Numbering



Default Switch Settings

A (34)	B (23)	C (12)
0 0 0 0 0 0 0 0	1 1 1 0 0 0 0 1	0 0 0 0 1 0 0 0

0 = Closed
1 = Open

Dedicated I/O Memory Base Address

B4 = open for 16 address bits
= closed for 20 address bits

/ADR13 -) C1	0 = Closed
/ADR12 -) C2	1 = Open
/ADR11 -) C3	for /ADR
/ADR10 -) C4	
/ADR F -) C5	0 = Open
/ADR E -) C6	1 = Closed
/ADR D -) C7	for ADR
/ADR C -) C8	

/ADR is inverted ADR as it appears on the bus.

Control / Status-Register Address

A1 = open for 8 address bits
= closed for 12 address bits

/ADR B -) B5	0 = Closed
/ADR A -) B6	1 = Open
/ADR 9 -) B7	for /ADR
/ADR 8 -) B8	
/ADR 7 -) A2	0 = Open
/ADR 6 -) A3	1 = Closed
/ADR 5 -) A4	for ADR
/ADR 4 -) A5	
/ADR 3 -) A6	
/ADR 2 -) A7	
/ADR 1 -) A8	

/ADR is inverted ADR as it appears on the bus.

Wait States Switch Selection

B1	B2	B3	Wait
Closed	Closed	Closed	1 Clock Cycle
Closed	Closed	Open	2 Clock Cycles
Closed	Open	Closed	3 Clock Cycles
Closed	Open	Open	4 Clock Cycles
Open	Closed	Closed	5 Clock Cycles
Open	Closed	Open	6 Clock Cycles
Open	Open	Closed	7 Clock Cycles
Open	Open	Open	8 Clock Cycles

APPENDIX E

CONTROL / STATUS-REGISTERS AND I/O MEMORY STRUCTURE

CONTROL-REGISTER

Bit 15	STONTRIG	0 = Normal Start 1 = Start on Trigger
Bit 14	CLEAR	0 = Clear 1 = Go
Bit 13	NORMLA	0 = Normal Single Scan 1 = Logic-Analyser On
Bit 12	EPOS	0 = Disable Pos Edge 1 = Enable Pos Edge
Bit 11	ENEG	0 = Disable Neg Edge 1 = Enable Neg Edge
Bit 10	EXINT	0 = External Access 1 = Internal Access
Bit 9	SPEED	0 = Normal Speed 1 = Full Speed
Bit 8	G2	0 = Enable Clock2 1 = Disable Clock 2
Bit 7-5	C2,B2,A2	0,0,0 = Single Step 0,0,1 = 00.25 MHz 0,1,0 = 00.50 MHz 0,1,1 = 01.00 MHz 1,0,0 = 02.00 MHz 1,0,1 = 04.00 MHz 1,1,0 = 08.00 MHz 1,1,1 = 16.00 MHz
Bit 4	G1	0 = Enable Clock1 1 = Disable Clock 1
Bit 3-1	C1,B1,A1	0,0,0 = Single Step 0,0,1 = 00.25 MHz 0,1,0 = 00.50 MHz 0,1,1 = 01.00 MHz 1,0,0 = 02.00 MHz 1,0,1 = 04.00 MHz 1,1,0 = 08.00 MHz 1,1,1 = 16.00 MHz
Bit 0	SSTEP	0 = Step Low 1 = Step High

STATUS-REGISTER

Bit 15	TRIG STARTED	0 = Start FF Off 1 = Start FF On
Bit 14	LA TRIGGERED	0 = No Trigger 1 = Triggered
Bit 13	LA FINISHED	0 = LA Finished 1 = LA Busy
Bit 12	LA CLOCKING	0 = LA Stopped 1 = LA Clocking
Bit 11	CLOCKING	0 = Not Clocking 1 = Clocking
Bit 10	MXCTL	0 = High Memory 1 = Low Memory
Bit 9-0	ADDRESS	Trigger Address

PATTERN-OUTPUT-MEMORY

Tn = Tri-State Control Bit n	0 = Put Pin in Tri-State 1 = Drive pin according to Vn
Vn = Drive Control Bit n	0 = Drive Pin Low 1 = Drive Pin High

Speed = 0 (Non - Multiplexed Mode)

At offset 000H (default 0:0000)

Low Address								High Address							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
T4	V4	T3	V3	T2	V2	T1	V1	T8	V8	T7	V7	T6	V6	T5	V5

At offset 800H (default 0:0000)

Low Address								High Address							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
T12	V12	T11	V11	T10	V10	T9	V9	T16	V16	T15	V15	T14	V14	T13	V13

Speed = 1 (Multiplexed Mode)

At offset 000H (default 0:0000) and offset 800H (default 0:0000)

Low Address								High Address							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
T4	V4	T3	V3	T2	V2	T1	V1	T8	V8	T7	V7	T6	V6	T5	V5

PATTERN-INPUT-MEMORY

Hn = Input High Bit n	0 = Below High Threshold 1 = Above High Threshold
Ln = Input Low Bit n	0 = Below Low Threshold 1 = Above Low Threshold

Speed = 0 (Non - Multiplexed Mode)

At offset 800H (default 0:8000)

Low Address								High Address							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
H4	L4	H3	L3	H2	L2	H1	L1	H8	L8	H7	L7	H6	L6	H5	L5

At offset 808H (default 0:8800)

Low Address								High Address							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
H12	L12	H11	L11	H10	L10	H9	L9	H16	L16	H15	L15	H14	L14	H13	L13

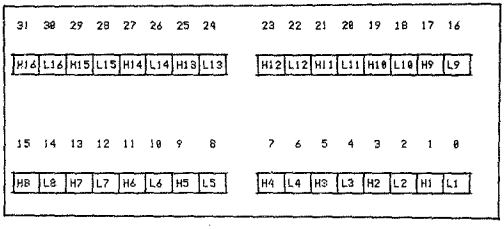
Speed = 1 (Multiplexed Mode)

At offset 800H (default 0:8000) and offset 808H (default 0:8800)

Low Address								High Address							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
H4	L4	H3	L3	H2	L2	H1	L1	H8	L8	H7	L7	H6	L6	H5	L5

TRIGGER-PATTERN

Hn = Pattern High Bit n	0 = Below High Threshold 1 = Above High Threshold
Ln = Pattern Low Bit n	0 = Below Low Threshold 1 = Above Low Threshold



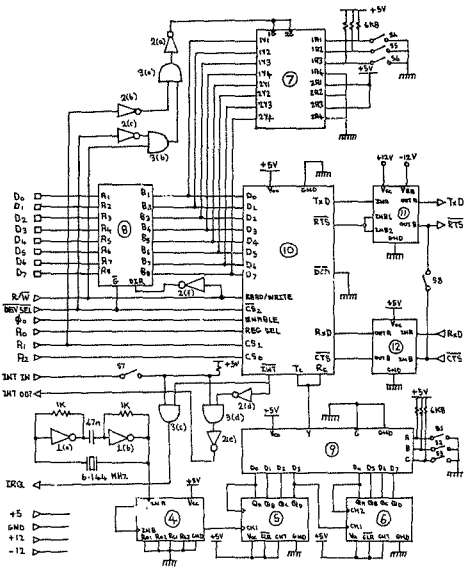
APPENDIX F

SERIAL COMMUNICATION BOARD CIRCUIT DIAGRAM

SERIAL COMMUNICATION BOARD

This card impliments the serial link on the Apple II + compatible computer. This card is basically a UART with baud rate switches and a Parameter-Register that can be used for inputting information to the terminal driver software. The list of components appears below followed by the circuit diagram.

Number	Component	Description
1	7404	Hex Inverters
2	74LS04	Hex Inverters
3	74LS08	Quadruple 2-Input Positive AND Gates
4	74LS90	Decade Counter
5,6	74LS197	Presettable Binary Counter
7	74LS244	Octal Buffers / Tri-State Line Drivers
8	75LS245	Octal Bus Transceivers
9	74LS251	1 of 8 Data Selector / Multiplexer
10	MC6850	Asynchronous Communications Interface
11	MC1488	Dual RS 232 C Line Driver
12	MC1489	Dual RS 232 C Line Receiver



SERIAL COMMUNICATION BOARD

APPENDIX G

SERIAL COMMUNICATION BOARD CONTROL

Baud Rates (with /16 Clock)

	Switch 1	Switch 2	Switch 3		Baud Rate
0)	Open	Open	Open	-)	150 baud
1)	Open	Open	Closed	-)	300 baud
2)	Open	Closed	Open	-)	600 baud
3)	Open	Closed	Closed	-)	1200 baud
4)	Closed	Open	Open	-)	2400 baud
5)	Closed	Open	Closed	-)	4800 baud
6)	Closed	Closed	Open	-)	9600 baud
7)	Closed	Closed	Closed	-)	19200 baud

Parameter Switches

Switch 4	Switch 5	Switch 6	
Bit 0	Bit 1	Bit 2	of Parameter Address

Open = Logic 1
Closed = Logic 0

Interrupt Control Switch

Switch 7 -	Open =	Ignore	Interrupt daisy Chain Line (INT IN)
-	Closed =	Use	Interrupt daisy Chain Line (INT IN)

RTS - CTS Control

Switch 8 -	Open =	Use RTS (Request to Send) & CTS (Clear to Send) lines
-	Closed =	Short RTS & CTS lines, Ignore hardware handshaking

Memory Map

Data-Register	- C0n7H / C0nFH
Control / Status-Register	- C0n6H / C0nEH
Parameter-Register	- C0n8H / C0n1H / C0n4H / C0n5H C0n8H / C0n9H / C0nCH / C0nDH

n = Slot Number + 8

Parameter-Register Structure

Bit Numbers	7	6	5	4	3	2	1	0
Contents	0	1	0	1	0	S6	S5	S4

Upper Bits Fixed and lower bits switch selectable.

Control-Register Programming Information

Reset UART -> 03H
Set UART to Default -> 11H

Status-Register Programming Information

Bit 0	- Transmit Data Register Empty	0 = Not Empty 1 = Empty (Send next Char)
Bit 1	- Receive Data Register Full	0 = Not Full 1 = Full (Fetch Char)

For more information see the Motorolla documentation on the MC6850 Asynchronous Communications Interface Adapter (ACIA)

APPENDIX H

SERIAL COMMUNICATION PROGRAM LISTING

INTRODUCTION

This program contained in the listing is written in 6809 Assembly Language and runs under the FLEX Operating-System. The program is a menu driven Terminal emulation program with special features.

Special features include the ability to be able to transmit commands directy from a file rather than from the Keyboard. Another feature is the ability to be able to capture all keystrokes in a file for later transmission, and the ability to capture all that is received on the serial link in a file.

Running the program is easy as it is menu driven, so the various functions will not be discussed in detail here.

```
                        * Serial Card Utility Program
                        * ---------------------------
                        *
                        * By M. K. Hawes
                        * -------------
                        *
                        * Last Updated 27 February 1984
                        * ----------------------------
                        *
                        ************************************************
                        *                                              *
                        *    This Utility is used to communicate via   *
                        *    the serial link with monitor programs in  *
                        *    computers which expect an external terminal. *
                        *    This Utility allows you to send information  *
                        *    from a text file, and capture any output  *
                        *    from the computer in a file stored under  *
                        *    Flex. Another feature is that you can     *
                        *    capture all you type in a file and then   *
                        *    send it later without retyping it.        *
                        *                                              *
                        *    This Utility was written for my M.Sc.     *
                        *    project where it was necessary to         *
                        *    communicate with a monitor on an 8086     *
                        *    single board computer.                    *
                        *                                              *
                        *    The maximum possible speed of operation   *
                        *    is about 1200 Baud as characters are lost *
                        *    at speeds exceeding this.                 *
                        *                                              *
                        ************************************************
                        *
                        * Dos Equates
                        * -----------
                        *
CD00  COLDS   EQU   $CD00     Cold Start Entry Point
CD03  WARMS   EQU   $CD03     Warm Start Entry Point
CD06  RENTER  EQU   $CD06     DOS Main Loop Re-entry Pt.
CD09  INCH    EQU   $CD09     Input Character
CD0C  INCH2   EQU   $CD0C     Input Character
CD0F  OUTCH   EQU   $CD0F     Output Character
CD12  OUTCH2  EQU   $CD12     Output Character
CD15  GETCHR  EQU   $CD15     Get Character
CD18  PUTCHR  EQU   $CD18     Put Character
CD1B  INBUFF  EQU   $CD1B     Input into Line Buffer
CD1E  PSTRNG  EQU   $CD1E     Print String
CD21  CLASS   EQU   $CD21     Classify Character
CD24  PCRLF   EQU   $CD24     Print CR and LF
CD27  NXTCH   EQU   $CD27     Get Next Buffer Character
CD2A  RSTRIO  EQU   $CD2A     Restore I/O Vectors
CD2D  GETFIL  EQU   $CD2D     Get File Specification
CD30  LOAD    EQU   $CD30     File Loader
CD33  SETEXT  EQU   $CD33     Set Extension
```

```
                CD36  ADDBX   EQU   $CD36    Add B-reg to X-reg
                CD39  OUTDEC  EQU   $CD39    Output Decimal Number
                CD3C  OUTHEX  EQU   $CD3C    Output Hexadecimal Number
                CD3F  RPTERR  EQU   $CD3F    Report Error
                CD42  GETHEX  EQU   $CD42    Get Hexadecimal Number
                CD45  OUTADR  EQU   $CD45    Output Hexadecimal Address
                CD48  INDEC   EQU   $CD48    Input Decimal Number
                CD4B  DOCMND  EQU   $CD4B    Call DOS as a Subroutine
                CD4E  STAT    EQU   $CD4E    Check Terminal Input Status
                      *
                      * FMS Equates
                      * ----------
                      *
                D400  FMSINT  EQU   $D400    FMS Initialisation
                D403  FMSCLS  EQU   $D403    FMS Close
                D406  FMS     EQU   $D406    FMS Call
                      *
                      * FMS Functions
                      * -------------
                      *
                0000  RDWR    EQU   $0       Read/Write Next Byte/Char
                0001  OPENRD  EQU   $1       Open for Read
                0002  OPENWR  EQU   $2       Open for Write
                0003  OPENUP  EQU   $3       Open for Update
                0004  CLOSE   EQU   $4       Close File
                0005  REWIND  EQU   $5       Rewind File
                0006  OPNDIR  EQU   $6       Open Directory
                0007  GETINF  EQU   $7       Get Information Record
                0008  PUTINF  EQU   $8       Put Information Record
                0009  GETSEC  EQU   $9       Read Single Sector
                000A  PUTSEC  EQU   $A       Write Single Sector
                000B  RES1    EQU   $B       Reserved for Future
                000C  DELETE  EQU   $C       Delete File
                000D  RENAME  EQU   $D       Rename File
                000E  RES2    EQU   $E       Reserved for Future
                000F  NXTSEQ  EQU   $F       Next Sequential Sector
                0010  OSIREC  EQU   $10      Open System Info Record
                0011  GETRND  EQU   $11      Get Random Byte from Sector
                0012  PUTRND  EQU   $12      Put Random Byte in Sector
                0013  RES3    EQU   $13      Reserved for Future
                0014  NXTDRV  EQU   $14      Find Next Drive
                0015  REGNON  EQU   $15      Position to Record N
                0016  BACREC  EQU   $16      Backup One Record
                      *
                      * File Control Block
                      * ------------------
                0000  FUNCTN  EQU   0        Function Code
                0001  ERSTAT  EQU   1        Error Status Byte
                0002  ACTVST  EQU   2        Activity Status
                0003  DRVNUM  EQU   3        Drive Number
                0004  FNAME   EQU   4        -11   File Name
                000C  FEXTEN  EQU   12       -14   File Extension
                000F  FATTRB  EQU   15       File Attributes
                0010  RES4    EQU   16       Reserved for Future
```

```
                0011  BEGTRK  EQU   17        Starting Track of File
                0012  BEGSEC  EQU   18        Starting Sector of File
                0013  ENDTRK  EQU   19        Ending Track of File
                0014  ENDSEC  EQU   20        Ending Sector of File
                0015  FILSIZ  EQU   21        -22   File Size
                0017  FSECMI  EQU   23        File Sector Map Indicator
                0018  RES5    EQU   24        Reserved for Future
                0019  FCDATE  EQU   25        -27   File Creation Date
                001C  FCBLPT  EQU   28        -29   FCB List Pointer
                001E  TRACK   EQU   30        Current Track
                001F  SECTOR  EQU   31        Current Sector
                0020  RECORD  EQU   32        -33   Current Record No.
                0022  DATNDX  EQU   34        Data Index
                0023  RNDNDX  EQU   35        Random Index
                0024  NAMEBF  EQU   36        -46   Name Work Buffer
                002F  DIRADD  EQU   47        -49   Current Dir. Add
                0032  FDELDP  EQU   50        -52   First Del Dir Ptr.
                0035  SCRTCH  EQU   53        -63   Scratch Bytes
                003B  COMPRS  EQU   59        Space Compression Flag
                0040  BUFFER  EQU   64        -319  Sector Buffer (256)
                      *
                      * Special Values
                      * --------------
                CC2B  MEMEND  EQU   $CC2B     End User Mem For Flex
                CC16  ESKIRC  EQU   $CC16     Escape Return Register
                0060  DELAY   EQU   $0060     Delay Parameter
                0013  CTRLS   EQU   $13       Wait Character
                      *
                      * File Control Blocks
                      * -------------------
                C100  FCB1    EQU   $C100     Utility Area
                C240  FCB2    EQU   $C240     Utility Area
                C380  FCB0    EQU   $C380     Utility Area
                      *
                      * Macro Definitions
                      * -----------------
                      SUBIN   MACRO
                              PSHS    A,B,X,Y,CC
                              ENDM
                      *
                      SUBOUT  MACRO
                              PULS    A,B,X,Y,CC,PC
                              ENDM
                      *
                      * Start of Program
                      * ----------------
0000                          ORG     $0000
                      *
0000 20  01   TERM    BRA     TERM1     Get Around Temps
0002 01       VN      FCB     1         Version Number
                      *
0003 8D  09   TERM1   BSR     INIT      Initialise
                      *
                      * Main Loop
```

```
                0011  BEGTRK  EQU   17        Starting Track of File
                0012  BEGSEC  EQU   18        Starting Sector of File
                0013  ENDTRK  EQU   19        Ending Track of File
                0014  ENDSEC  EQU   20        Ending Sector of File
                0015  FILSIZ  EQU   21        ~22   File Size
                0017  FSECMI  EQU   23        File Sector Map Indicator
                0018  RES5    EQU   24        Reserved for Future
                0019  FCDATE  EQU   25        ~27   File Creation Date
                001C  FCBLPT  EQU   28        ~29   FCB List Pointer
                001E  TRACK   EQU   30        Current Track
                001F  SECTOR  EQU   31        Current Sector
                0020  RECORD  EQU   32        ~33   Current Record No.
                0022  DATNDX  EQU   34        Data Index
                0023  RNDNDX  EQU   35        Random Index
                0024  NAMEBF  EQU   36        ~46   Name Work Buffer
                002F  DIRADD  EQU   47        ~49   Current Dir. Add
                0032  FDELDP  EQU   50        ~52   First Del Dir Ptr.
                0035  SCRTCH  EQU   53        ~63   Scratch Bytes
                003B  COMPRS  EQU   59        Space Compression Flag
                0040  BUFFER  EQU   64        ~319  Sector Buffer (256)
                      *
                      * Special Values
                      * --------------
                CC2B  MEMEND  EQU   $CC2B     End User Mem For Flex
                CC16  ESRTRG  EQU   $CC16     Escape Return Register
                0060  DELAY   EQU   $0060     Delay Parameter
                0013  CTRLS   EQU   $13       Wait Character
                      *
                      * File Control Blocks
                      * ------------------
                C100  FCB1    EQU   $C100     Utility Area
                C240  FCB2    EQU   $C240     Utility Area
                C380  FCB0    EQU   $C380     Utility Area
                      *
                      * Macro Definitions
                      * -----------------
                      SUBIN   MACRO
                              PSHS  A,B,X,Y,CC
                              ENDM
                      *
                      SUBOUT  MACRO
                              PULS  A,B,X,Y,CC,PC
                              ENDM
                      *
                      * Start of Program
                      * ----------------
0000                          ORG   $0000
                      *
0000 20  01           TERM    BRA   TERM1     Get Around Temps
0002 01               VN      FCB   1         Version Number
                      *
0003 8D  09           TERM1   BSR   INIT      Initialise
                      *
                      * Main Loop
```

```
                                * ---------
0005 BD  41          LOOP   BSR    KBDCHK    Check Keyboard
0007 17  0354               LBSR   FILCHK    Check For File Input
000A 8D  7B                 BSR    URTCHK    Check Uart
000C 20  F7                 BRA    LOOP      Loop Always
                                *
                                * Initialise Subroutine
                                * ====================
000E                 INIT   SUBIN  Subroutine Beginning
0010 7F  0475               CLR    HALFDUP   Clear Half Dup
0013 7F  0476               CLR    NBSTRL    Clear No BS Trl
0016 7F  0477               CLR    F1FLAG    Clear File 1 Flag
0019 7F  0478               CLR    F2FLAG    Clear File 2 Flag
001C 7F  0479               CLR    FOFLAG    Clear File Out Flag
001F 7F  047A               CLR    WAIT      Clear Wait Flag
                                *
0022 CC  0060               LDD    #DELAY    Ininial Delay
0025 FD  0472               STD    SPEED     Store in Speed
0028 7F  0474               CLR    FILTYP    Clear File Type
                                *
002B CC  0AD4               LDD    #ENDPGM   End of Program
002E FD  047D               STD    BUFSP     Start of Buffer
0031 FD  04B1               STD    FFTR      File Pointer
0034 FC  CC2B               LDD    MEMEND    End of Memory
0037 FD  047F               STD    BUFEP     End of Buffer
                                *
003A CC  043A               LDD    #EXIT     Patch Escape Ret Reg
003D FD  CC16               STD    ESRTRG    Store Address
                                *
0040 17  010B               LBSR   SETSTT    Set Status Of Utility
0043 17  0A55               LBSR   UINIT     Initialise Uart
0046                        SUBOUT Subroutine End
                                *
                                * Check keyboard Subroutine
                                * ========================
0048                 KBDCHK SUBIN  Subroutine Beginning
004A BD  CD4E               JSR    STAT      Check Keyboard Status
004D 27  26                 BEQ    KBDEXT    Exit If No Char
                                *
004F 17  0A79               LBSR   KEYIN     Get Char No Echo
                                *
0052 81  1B                 CMPA   #ESC      Check For Escape
0054 26  05                 BNE    NESC      Not Escape
0056 17  00F5               LBSR   SETSTT    Set Status
0059 20  1A                 BRA    KBDEXT    Exit From Subroutine
                                *
005B F6  0476        NESC   LDB    NBSTRL    Check If Translate BS
005E 5D                     TSTB
005F 26  06                 BNE    NCTQ      If Not Then Don't
                                *
0061 81  08                 CMPA   #8        Check for Back Space
0063 26  02                 BNE    NCTQ      Not Back Space
0065 86  7F                 LDA    #$7F      Change to Del
                                *
```

```
0067 81   13     NCTQ    CMPA   #CTRLS   Check For Ctrl-S
0069 26   05             BNE    NCTS     Not Ctrl-S
006B 73   047A           COM    WAIT     Compliment Wait Flag
006E 20   05             BRA    KBDEXT   Consume Character
                 *
0070 17   02D7   NCTS    LBSR   LOGIN    Write Character
0073 8D   02             BSR    CHOUT    Character Output
                 *
0075             KBDEXT  SUBOUT Subroutine End
                 *
                 * Output Char From Keyboard or File
                 * ================================
0077             CHOUT   SUBIN  Subroutine Beginning
0079 F6   0475           LDB    HALFDUP  Check Duplex
007C 5D                  TSTB            Set Flags
007D 27   03             BEQ    CHKCHR   No Echo Full Duplex
007F BD   CD18           JSR    PUTCHR   Console Output
                 *
0082 17   0A33   CHKCHR  LBSR   UOUT     Output to Uart
                 *
0085                     SUBOUT Subroutine End
                 *
                 * Uart Check Subroutine
                 * =====================
0087             URTCHK  SUBIN  Subroutine Beginning
0089 17   0A1E           LBSR   USTAT    Check Uart Status
008C 27   16             BEQ    URTEXT   Exit if no Character
008E 17   0A23           LBSR   UIN      Get Uart Character
0091 7D   0474           TST    FILTYP   Check File Type
0094 26   02             BNE    NSTRIP   Do Not Strip
0096 84   7F             ANDA   #$7F     Strip Most Significant Bit
0098 17   028E   NSTRIP  LBSR   PIFILE   Write Character
009B BD   CD18           JSR    PUTCHR   Console Output
009E CC   0000           LDD    #0       Set Counter to 0
00A1 FD   0485           STD    DELCTR   Store in Delay Ctr
                 *
00A4             URTEXT  SUBOUT Subroutine End
                 *
                 * Print Free Memory
                 * =================
00A6             PRFREE  SUBIN  Subroutine Beginning
00A8 8E   04EF           LDX    #FREEM   Free Memory Message
00AB BD   CD1E           JSR    PSTRNG   Print
00AE FC   047F           LDD    BUFEP    Buffer End
00B1 B3   0481           SUBD   FPTR     Get Free Memory
00B4 8D   1E             BSR    PRINTD   Print D
                 *
00B6 8E   051D           LDX    #USEDM   Used Memory Message
00B9 BD   CD1E           JSR    PSTRNG   Print
00BC FC   0481           LDD    FPTR     Get Pointer
00BF B3   047D           SUBD   BUFSP    Get Used Memory
00C2 8D   10             BSR    PRINTD   Print D
                 *
00C4 8E   054B           LDX    #ALLMM   Total Memory Message
```

```
 273  00C7 BD  CD1E          JSR    PSTRNG   Print
 274  00CA FC  047F          LDD    BUFEP    End Buffer
 275  00CD B3  047D          SUBD   BUFSP    Get Buffer Size
 276  00D0 8D  02            BSR    PRINTD   Print D
 277                  *
      00D2                   SUBOUT Subroutine End
 279                  *
 280                  * Print D
 281                  * =======
      00D4            PRINTD SUBIN  Subroutine Beginning
 283  00D6 FD  0487          STD    NUMSTR   Store Number
 284  00D9 8E  0487          LDX    #NUMSTR  Point to Number
 285  00DC BD  CD45          JSR    OUTADR   Print Number
      00DF                   SUBOUT Subroutine End
 287                  *
 288                  * Print File Name Subroutine
 289                  * ==========================
      00E1            FNAME  SUBIN  Subroutine Beginning
 291  00E3 108E 0721         LDY    #SPACER  Point To Spacer
 292  00E7 C6  05            LDB    #5       Length of Spacer
 293  00E9 8D  15            BSR    PNNDT    Print Spacer
 294  00EB A6  03            LDA    DRVNUM,X Get Drive Number
 295  00ED 8B  30            ADDA   #$30     Change to Character
 296  00EF BD  CD18          JSR    PUTCHR   Output Character
 297  00F2 31  04            LEAY   FNAME,X  Point to File Name
 298  00F4 C6  08            LDB    #8       Number of Characters
 299  00F6 8D  0E            BSR    FNAME    Print Name
 300  00F8 31  0C            LEAY   FEXTEN,X Point to File Extension
 301  00FA C6  03            LDB    #3       Number of Characters
 302  00FC 8D  08            BSR    FNAME    Print Name
      00FE                   SUBOUT Subroutine End
 304                  *
 305                  * Print Name
 306                  * ==========
      0100            PNNDT  SUBIN  Subroutine Beginning
 308  0102 1F  21            TFR    Y,X      Put Y -> X
 309  0104 20  09            BRA    PNLOOP   Entry No '.'
 310                  *
      0106            FNAME  SUBIN  Subroutine Beginning
 312  0108 1F  21            TFR    Y,X      Put Y -> X
 313  010A 86  2E            LDA    #'.      Print '.
 314  010C BD  CD18          JSR    PUTCHR   Output Character
 315  010F A6  80    PNLOOP  LDA    ,X+      Get Character
 316  0111 BD  CD18          JSR    PUTCHR   Output Character
 317  0114 5A                DECB            Decrement Counter
 318  0115 26  F8            BNE    PNLOOP   Loop Until Last Char
 319                  *
      0117                   SUBOUT Subroutine End
 321                  *
 322                  * Print Speed Subroutine
 323                  * ======================
      0119            PSPEED SUBIN  Subroutine Beginning
 325  011B 8E  08C0          LDX    #SPEEDM  Speed Message
 326  011E BD  CD1E          JSR    PSTRNG   Print
```

```
 327  0121 8E  0472          LDX   #SPEED    Point to Speed
 328  0124 BD  CD45          JSR   OUTADR    Output Number
 329  0127 86  20            LDA   #'        Print Blank
 330  0129 BD  CD18          JSR   PUTCHR
 331  012C 86  29            LDA   #')       Print Bracket
 332  012E BD  CD18          JSR   PUTCHR
      0131                   SUBOUT Subroutine End
 334                    *
 335                    * Set Speed
 336                    * =========
      0132              SSPEED SUBIN Subroutine Beginning
 338  0135 8E  074F    SSSAGN LDX   #ISPEDM   Input Speed Message
 339  0138 BD  CD1E          JSR   PSTRNG    Print
 340  013B BD  CD1B          JSR   INBUFF    Input Buffer
 341  013E BD  CD42          JSR   GETHEX    Get Hex Number
 342  0141 25  F2            BCS   SSSAGN    Try Again
 343  0143 5D                TSTB            Is There a No
 344  0144 26                BNE   ISANO     There is a no
 345  0146 8E                LDX   #DELAY    Default for return
 346  0149 BF  ISANO   STX   SPEED   Store Speed
      014C                   SUBOUT Subroutine End
 348                    *
 349                    * Set Status Subroutine
 350                    * ====================
      014E              SETSTT SUBIN Subroutine Beginning
 352  0150 8E  0489    SSAGN  LDX   #STATM1   Status Message
 353  0153 BD  CD1E          JSR   PSTRNG    Print
 354  0156 17  FF4D          LBSR  PRFREE    Print Free Space
 355  0159 8E  0579          LDX   #t ATM2   Status Message
 356  015C BD  CD1E          JSR   PSTRNG    Print
 357  015F B6  0475          LDA   HALFDUP   Look at Duplex Flag
 358  0162 4D                TSTA            Set Flags
 359  0163 26  05            BNE   SHFDUP    Half Duplex Jump
 360  0165 8E  0776          LDX   #MFDUP    Full Duplex Message
 361  0168 20  03            BRA   BSTCHK    Now Check Backspace
 362  016A 8E  07AD    SHFDUP LDX   #MHDUP    Half Duplex Message
 363  016D BD  CD1E    BSTCHK JSR   PSTRNG    Print
 364  0170 B6  0476          LDA   NBSTRL    Look at Backspace Flag
 365  0173 4D                TSTA            Set Flags
 366  0174 26  05            BNE   SNBST     No Backspace T Jump
 367  0176 8E  07E4          LDX   #MBST     Backspace Trans Mess
 368  0179 20  03            BRA   LOGPS     Now Print Log State
 369  017B 8E  081B    SNBST  LDX   #MBSNT    No Backspace Trans
 370  017E BD  CD1E    LOGPS  JSR   PSTRNG    Print
 371  0181 B6  0474          LDA   FILTYP    Check File Type
 372  0184 4D                TSTA            Set Flags
 373  0185 26  05            BNE   NSCOM     No Space Compress
 374  0187 8E  0889          LDX   #HSPCM    Has Space Compress
 375  018A 20  03            BRA   SPDCHK    Speed Check
 376  018C 8E  0852    NSCON  LDX   #NSPCM    No Space Compress
 377  018F BD  CD1E    SPDCHK JSR   PSTRNG    Print
 378  0192 8D  85            BSR   PSPEED    Print Speed
 379  0194 B6  0477          LDA   FIFLAG    Log File Flag
 380  0197 4D                TSTA            Set Flags
```

```
381  0198 26 08          BNE   F1O    File 1 Open Jump
382  019A 8E 68F1        LDX   #F1CM   File 1 Closed Mess
383  019D BD CD1E        JSR   PSTRNG  Print
384  01A0 20 0C          BRA   F2PS    File 2 Print State
385  01A2 8E 0928  F1O   LDX   #F1OM   File 1 Open Mess
386  01A5 BD CD1E        JSR   PSTRNG  Print
387  01A8 8E C100        LDX   #FCB1   Get FCB Address
388  01AB 17 FF33        LBSR  PFNAME  Print File Name
389  01AE B6 0478  F2PS  LDA   F2FLAG  Storage File Flag
390  01B1 4D             TSTA          Set Flags
391  01B2 26 08          BNE   F2O     File 2 Open Jump
392  01B4 8E 095F        LDX   #F2CM   File 2 Closed Mess
393  01B7 BD CD1E        JSR   PSTRNG  Print
394  01BA 20 0C          BRA   FOPS    File Out Print State
395  01BC 8E 0996  F2O   LDX   #F2OM   File 2 Open Mess
396  01BF BD CD1E        JSR   PSTRNG  Print
397  01C2 8E C240        LDX   #FCB2   Get FCB Address
398  01C5 17 FF19        LBSR  PFNAME  Print File Name
399  01C8 B6 0479  FOPS  LDA   FOFLAG  Output File Flag
400  01CB 4D             TSTA          Set Flags
401  01CC 26 08          BNE   FOO     File Output Open Jump
402  01CE 8E 09CD        LDX   #FOCM   File Output Closed Mess
403  01D1 BD CD1E        JSR   PSTRNG  Print
404  01D4 20 0C          BRA   NOWSS   Now Set Status
405  01D6 8E 0A04  FOO   LDX   #FOOM   File Output Open Mess
406  01D9 BD CD1E        JSR   PSTRNG  Print
407  01DC 8E C380        LDX   #FCB0   Get FCB Address
408  01DF 17 FEFF        LBSR  PFNAME  Print File Name
409  01E2 8E 0A3B  NOWSS LDX   #MENU   Menu Message
410  01E5 BD CD1E        JSR   PSTRNG  Print
411                 *
412  01EB BD CD15        JSR   GETCHR  Input Character
413                 *
414  01EB 81 0D          CMPA  #13     Carriage Ret
415  01ED 27 5F          BEQ   SEXIT   Subroutine Exit
416  01EF 81 31    TR1   CMPA  #'1
417  01F1 26 06          BNE   TR2     Try 2
418  01F3 73 0475        COM   HALFDUP Invert Duplex
419  01F6 16 FF57        LBRA  SSAGN   Do Again
420  01F9 81 32    TR2   CMPA  #'2
421  01FB 26 06          BNE   TR3     Try 3
422  01FD 73 0476        COM   NBSTRL  Invert BS Translation
423  0200 16 FF4D        LBRA  SSAGN   Do Again
424  0203 81 33    TR3   CMPA  #'3
425  0205 26 06          BNE   TR4     Try 4
426  0207 73 0474        COM   FILTYP  Invert File Type
427  020A 16 FF43        LBRA  SSAGN   Do Again
428  020D 81 34    TR4   CMPA  #'4
429  020F 26 06          BNE   TR      Try 5
430  0211 17 FF1F        LBSR  SSPEED  Set Speed
431  0214 16 FF39        LBRA  SSAGN   Do Again
432  0217 81 35    TR5   CMPA  #'5
433  0219 26 05          BNE   TR6     Try 6
434  021B 8D 39          BSR   FUN5    Function 5 - Log File
```

```
                021D 16  FF30          LBRA   SSAGN    Do Again
                0220 81  36    TR6     CMPA   #'6
                0222 26  05            BNE    TR7      Try 7
                0224 8D  43            BSR    FUN6     Function 6 - Storage File
                0226 16  FF27          LBRA   SSAGN    Do Again
                0229 81  37    TR7     CMPA   #'7
                022B 26  05            BNE    TR8      Try 8
                022D 8D  53            BSR    FUN7     Function 7 - Output File
                022F 16  FF1E          LBRA   SSAGN    Do Again
                0232 81  38    TR8     CMPA   #'8
                0234 26  03            BNE    TR9      Try 9
                0236 16  0201          LBRA   EXIT     Function 8 - Exit
                0239 81  39    TR9     CMPA   #'9
                023B 26  03            BNE    CERRR    Character Error
                023D 16  0216          LBRA   ABORT    Function 9 - Abort
                                *
                0240 86  07    CERRR   LDA    #07      Beep
                0242 BD  CD18          JSR    PUTCHR   Output
                0245 BD  CD18          JSR    PUTCHR   Output
                0248 BD  CD18          JSR    PUTCHR   Output
                024B 16  FF02          LBRA   SSAGN    Do Again
                                *
                024E 8E  0635  SEXIT   LDX    #RETM    Return Message
                0251 BD  CD1E          JSR    PSTRNG   Print
                0254                   SUBOUT Subroutine End
                                *
                                * Open / Close Log File
                                * =====================
                0256           FUN5    SUBIN  Subroutine Beginning
                0258 8D  3C            BSR    FILE1
                025A B6  0477          LDA    F1FLAG
                025D 4D                TSTA
                025E 26  04            BNE    FUN5B
                0260 8D  64            BSR    OPENF
                0262 20  03            BRA    FUN5E
                                *
                0264 17  0172  FUN5B   LBSR   CLOUT
                0267           FUN5E   SUBOUT Subroutine End
                                *
                                * Open / Close Storage File
                                * =========================
                0269           FUN6    SUBIN  Subroutine Beginning
                026B 8D  39            BSR    FILE2
                026D B6  0478          LDA    F2FLAG
                0270 4D                TSTA
                0271 26  04            BNE    FUN6B
                0273 8D  51            BSR    OPENF
                0275 20  03            BRA    FUN6E
                                *
                0277 17  0083  FUN6B   LBSR   CLOSEF
                027A FC  047D  FUN6E   LDD    BUF5P
                027D FD  0481          STD    FPTR
                0280                   SUBOUT Subroutine End
                                *
```

```
                        * Open / Close Output File
                        * ********** ***********
      02B2              FUN7    SUBIN   Subroutine Beginning
0492  02B4 8D   30              BSR     FILE0
0493  02B6 B6   0479            LDA     FOFLAG
0494  02B9 4D                   TSTA
0495  02BA 26   05              BNE     FUN7B
0496  02BC 17   0112            LBSR    OPNOUT
0497  02BF 20   03              BRA     FUN7E
0498                    *
0499  0291 17   0145    FUN7B   LBSR    CLOUT
      0294              FUN7E   SUBOUT  Subroutine End
0501                    *
0502                    * Set File 1 (Log File)
0503                    * ********************
      0296              FILE1   SUBIN   Subroutine Beginning
0505  0298 CC   C100            LDD     #FCB1       Store FCB
0506  029B FD   047B            STD     FCB
0507  029E CC   0477            LDD     #F1FLAG     File 1 Flag
0508  02A1 FD   0483            STD     FFLAG
      02A4              SUBOUT  Subroutine End
0510                    *
0511                    * Set File 2 (Storage File)
0512                    * ********************
      02A6              FILE2   SUBIN   Subroutine Beginning
0514  02A8 CC   C240            LDD     #FCB2       Store FCB
0515  02AB FD   047B            STD     FCB
0516  02AE CC   0478            LDD     #F2FLAG     File 2 Flag
0517  02B1 FD   0483            STD     FFLAG
      02B4              SUBOUT  Subroutine End
0519                    *
0520                    * Set File 0 ( Output File )
0521                    * ********************
      02B6              FILE0   SUBIN   Subroutine Beginning
0523  02B8 CC   C380            LDD     #FCB0       Store FCB
0524  02BB FD   047B            STD     FCB
0525  02BE CC   0479            LDD     #F0FLAG     File 0 Flag
0526  02C1 FD   0483            STD     FFLAG
      02C4              SUBOUT  Subroutine End
0528                    *
0529                    * Open Relevant File
0530                    * ******************
      02C6              OPENF   SUBIN   Subroutine Beginning
0532  02C8 8E   0726            LDX     #FNAMM      File Name Message
0533  02CB BD   CD1E            JSR     PSTRNG      Print
0534  02CE BD   CD1B            JSR     INBUFF      Read Into Buffer
0535  02D1 BE   047B            LDX     FCB         Get FCB Address
0536  02D4 BD   CD2D            JSR     GETFIL      Get File Spec
0537  02D7 1025 014A            LBCS    ERROR       Any Errors
0538  02DB 86   01              LDA     #1          Set to TXT Default
0539  02DD BD   CD33            JSR     SETEXT      Set Extension
0540  02E0 86   02              LDA     #OPENWR     Open for Write
0541  02E2 A7   84              STA     FUNCTN,X    Store Function
0542  02E4 BD   D406            JSR     FMS         Open for Write
```

```
02E7 1026 013A          LBNE   ERROR     Any Errors
02EB 86   00            LDA    #RDWR     Read/Write
02ED A7   84            STA    FUNCTN,X  Save in FCB
02EF B6   0474          LDA    FIL+ 9    Space Compression
02F2 A7   88 3B         STA    COMPRS,X  Store
                *
02F5 86   01            LDA    #1        Open File Flag
02F7 A7   9F 0483       STA    [FFLAG]   Store in Flag Place
02FB                    SUBOUT Subroutine End
                *
                * Close Storage File
                * ==================
02FD                    CLOSEF SUBIN  Subroutine Beginning
02FF B6   0478          LDA    F2FLAG    Check If File Open
0302 4D                 TSTA              Set Flags
0303 27   1F            BEQ    CLGEXT    Exit If Not Open
                *
0305 BE   047D          LDX    BUFSP     Point To Beginning
0308 BC   0481          CLEFLP CMPX   FPTR       See If Reached End
030B 27   12            BEQ    CLCLOS    Close File If Case
030D A6   80            LDA    ,X+       Get Character
030F 34   10            PSHS   X         Store X
0311 8E   C240          LDX    #FCB2     Point to FCB
0314 BD   D406          JSR    FMS       Write Character
0317 1026 010A          LBNE   ERROR     Any Errors
031B 35   10            PULS   X         Store X
031D 20   E9            BRA    CLEFLP    Loop
                *
031F 8D   85            CLCLOS BSR    FILE2     Set File 2
0321 17   00B5          LBSR   CLOUT     Close File
                *
0324                    CLGEXT SUBOUT Subroutine End
                *
                * Write To Storage File
                * =====================
0326                    PIFILE SUBIN  Subroutine Beginnin.
0328 F6   0478          LDB    F2FLAG    Check If File Open
032B 5D                 TSTB              Set Flags
032C 27   1A            BEQ    PIEXIT    Exit If not Open
032E BE   0481          LDX    FPTR      Get Pointer Value
0331 A7   84            STA    0,X       Store Character
0333 30   01            LEAX   1,X       Increment Pointer
0335 BF   0481          STX    FPTR      Store New Value
0338 BC   047F          CMPX   BUFEP     See If Reached End
033B 26   0B            BNE    PIEXIT    Exit If Not Equal
033D BE   06E1          LDX    #OVSAVM   Overflow Save Mess
0340 BD   CD1E          JSR    PSTRNG    Print
0343 8D   B8            BSR    CLOSEF    Close File
0345 17   00D0          LBSR   OVERUR    Overflow Error
                *
0348                    PIEXIT SUBOUT Subroutine End
                *
                * Write To Log File
                * =================
```

```
      034A                    LOGIN   SUBIN   Subroutine Beginning
598   034C F6   0477                  LDB     FIFLAG  See If File Open
599   034F 5D                  TSTB                    Set Flags
600   0350 27   0A            BEQ     LOGOUT  Exit If Closed
601   0352 8E   C100          LDX     #FCB1   Point To FCB
602   0355 BD   D406          JSR     FMS     Write Char
603   0358 1026 00C9          LBNE    ERROR   Any Errors ?
604                     *
      035C                    LOGOUT  SUBOUT Subroutine End
606                     *
607                     * Check For File Output
608                     * ====================
      035E                    FILCHK  SUBIN   Subroutine Beginning
610   0360 B6   0479          LDA     FOFLAG  Check If Open
611   0363 4D                  TSTA            Set Flags
612   0364 27   39            BEQ     FILEXT  Exit If File Not Open
613                     *
614   0366 B6   047A          LDA     WAIT    Check Wait Flag
615   0369 4D                  TSTA            Set Flags
616   036A 26   33            BNE     FILEXT  Exit in Wait Mode
617                     *
618   036C 8E   0485          LDX     DELCTR  Get Counter
619   036F BC   0472          CMPX    SPEED   Check If Reached Delay
620   0372 27   07            BEQ     FGETCH  Get Char From File
621   0374 30   01            LEAX    1,X     Increment X
622   0376 BF   0485          STX     DELCTR  Store Counter
623   0379 20   24            BRA     FILEXT  Exit If Not Equal
624                     *
625   037B 8E   C380  FGETCH  LDX     #FCB0   Output From File
626   037E BD   D406          JSR     FMS     Get Character
627   0381 26   0B            BNE     ENDFIL  Check If File End
628   0383 17   FCF1          LBSR    CHOUT   Output Character
629   0386 CC   0000          LDD     #0      Set Counter to Zero
630   0389 FD   0485          STD     DELCTR  Store in DELCTR
631   038C 20   11            BRA     FILEXT  Exit After Getting Char
632                     *
633   038E A6   01    ENDFIL  LDA     ERSTAT,X Get Error Status
634   0390 81   08            CMPA    #8      Is it EOF Error
635   0392 27   06            BEQ     NORCLS  Normal Close
636   0394 7F   0479          CLR     FOFLAG  Set File Closed Flag
637   0397 16   008B          LBRA    ERROR   Error Exit
638                     *
639   039A 17   FF19  NORCLS  LBSR    FILEC   Set File
640   039D 5D   3A            BSR     CLOUT   Close Output File
641                     *
      039F                    FILEXT  SUBOUT Subroutine End
643                     *
644                     * Open Output File
645                     * ================
      03A1                    OPNOUT  SUBIN   Subroutine Beginning
647   03A4 8E   0738          LDX     #FNAMM  File Name Message
648   03A6 BD   CD1E          JSR     PSTRNG  Print
649   03A9 BD   CD1B          JSR     INBUFF  Read Into Buffer
650   03AC 8E   C380          LDX     #FCB0   Get FCB Address
```

```
 651  03AF BD  CD2D          JSR   GETFIL    Get File Spec
 652  03B2 25  71            BCS   ERROR     Any Errors
 653  03B4 86  01            LDA   #1        Set to TXT Default
 654  03B6 BD  CD33          JSR   SETEXT    Set Extension
 655  03B9 86  01            LDA   #OPENRD   Open for Read
 656  03BB A7  84            STA   FUNCTN,X  Store Function
 657  03BD BD  D406          JSR   FMS       Open for Read
 658  03C0 26  63            BNE   ERROR     Any Errors
 659  03C2 86  00            LDA   #RDWR     Read/Write
 660  03C4 A7  84            STA   FUNCTN,X  Save in FCB
 661  03C6 B6  0474          LDA   FILTYP    Space Compression
 662  03C9 A7  88 3B         STA   COMPRS,X  Store
 663  03CC 86  01            LDA   #1        Open File
 664  03CE B7  0479          STA   FOPLAG    Store Flag
 665  03D1 CC  0000          LDD   #0        Set Counter to Zero
 666  03D4 FD  0485          STD   DELCTR    Store in DELCTR
 667                  *
      03D7                   SUBOUT Subroutine End
 669                  *
 670                  * Close Relevant File
 671                  * ===================
      03D9            CLOUT  SUBIN  Subroutine Beginning
 673  03DB A6  9F 0483       LDA   [FFLAG]   Is File Open
 674  03DF 4D                TSTA            Set Flags
 675  03E0 27  10            BEQ   CLOTEX    Exit If Closed
 676  03E2 6F  9F 0483       CLR   [FFLAG]   Set Flag
 677  03E6 BE  047B          LDX   FCB       Point To FCB
 678  03E9 86  04            LDA   #CLOSE    Close File Function
 679  03EB A7  84            STA   FUNCTN,X  Store Function
 680  03ED BD  D406          JSR   FMS       Do File Closure
 681  03F0 26  33            BNE   ERROR     Any Errors
 682                  *
      03F2            CLOTEX SUBOUT Subroutine End
 684                  *
 685                  * Purge Relevant File
 686                  * ===================
      03F4            PURGE  SUBIN  Subroutine Beginning
 688  03F6 A6  9F 0483       LDA   [FFLAG]   Is File Open
 689  03FA 4D                TSTA            Set Flags
 690  03FB 27  19            BEQ   PUREXT    Exit If Closed
 691  03FD 6F  9F 0483       CLR   [FFLAG]   Set Flag
 692  0401 BE  047B          LDX   FCB       Point to FCB
 693  0404 86  04            LDA   #CLOSE    Close File Function
 694  0406 A7  84            STA   FUNCTN,X  Store Function
 695  0408 BD  D406          JSR   FMS       Do File Closure
 696  040B 26  18            BNE   ERROR     Any Errors
 697  040D 86  0C            LDA   #DELETE   Delete File Function
 698  040F A7  84            STA   FUNCTN,X  Store Function
 699  0411 BD  D406          JSR   FMS       Do File Deletion
 700  0414 26  0F            BNE   ERROR     Any Errors
 701                  *
      0416            PUREXT SUBOUT Subroutine End
 703                  *
 704                  * Overflow Error
```

```
                        * ============
      0418              OVERROR  SUBIN   Subroutine Beginning
 041A 8E   06A1                  LDX     #OVMESS   Overflow Error
 041D BD   CD1E                  JSR     PSTRNG    Print
 0420 BD   CD15                  JSR     GETCHR    Wait for Char
      0423                       SUBOUT  Subroutine End
                        *
                        * Error Exit From Program
                        * ======================
 0425 BD   CD3F         ERROR    JSR     RPTERR    Report Error
 0428 8E   0667                  LDX     #ERMESS   Error Message
 042B BD   CD1E                  JSR     PSTRNG    Print
 042E 86   04                    LDA     #CLOSE    Close File Function
 0430 A7   84                    STA     FUNCTN,X  Store Function
 0432 BD   D406                  JSR     FMS       Do Closure
 0435 BD   CD15                  JSR     GETCHR    Wait for Char
      0438                       SUBOUT  Subroutine End
                        *
                        * Exit Subroutine
                        * ===============
 043A 17   FE59         EXIT     LBSR    FILE1     Close File 1
 043D 8D   9A                    BSR     CLOUT
 043F 17   FE64                  LBSR    FILE2     Close File 2
 0442 17   FEB8                  LBSR    CLOSEF
 0445 17   FE6E                  LBSR    FILEO     Output File
 0448 8D   8F                    BSR     CLOUT     Close Output File
 044A BD   D403                  JSR     FMSCLS    Close All Files
 044D 8E   05D7                  LDX     #FMESS    Finish Message
 0450 BD   CD1E                  JSR     PSTRNG    Print
 0453 7E   CD03                  JMP     WARMS     Warm Start
                        *
                        * Abort Exit
                        * ==========
 0456 17   FE3D         ABORT    LBSR    FILE1     Purge File 1
 0459 8D   99                    BSR     PURGE
 045B 17   FE48                  LBSR    FILE2     Purge File 2
 045E 8D   94                    BSR     PURGE
 0460 17   FE53                  LBSR    FILEO     Close Output File
 0463 17   FF73                  LBSR    CLOUT
 0466 BD   D403                  JSR     FMSCLS    Close All Files
 0469 8E   0607                  LDX     #ABMESS   Abort Message
 046C BD   CD1E                  JSR     PSTRNG    Print
 046F 7E   CD03                  JMP     WARMS     Warm Start
                        *
                        * Storage Locations
                        * ----------------
 0472 0000              SPEED    FDB     00        Speed
 0474 00                FILTYP   FCB     00        File Type
 0475 00                HALFDUP  FCB     00        Duplex Flag
 0476 00                NBSTRL   FCB     00        Back Space Translation Flag
 0477 00                F1FLAG   FCB     00        File 1 Flag
 0478 00                F2FLAG   FCB     00        File 2 Flag
 0479 00                FOFLAG   FCB     00        File Out Flag
 047A 00                WAIT     FCB     00        Wait Flag
```

```
 759  047B 0000          FCB   FDB  00      FCB Pointer Store
 760  047D 0000          BUFSP FDB  00      Buffer Start Pointer
 761  047F 0000          BUFEP FDB  00      Buffer End Pointer
 762  0481 0000          FPTR  FDB  00      File Pointer
 763  0483 0000          FFLAG FDB  00      Flag Pointer
 764  0485 0000          DELCTR FDB 00      Delay Counter
 765  0487 0000          NUMSTR FDB 00      Storage for Printed No.
 766                     *
 767                     *
 768  0489 0C 20 20 20   STATM1 FCC  12,/                    /
 769  049D 28 20 4D 2E          FCC  /( M. K. Hawes ) Serial Utility/
 770  04BB 0D 20 20 20          FCC  13,/                    /
 771  04CF 3D 3D 3D 3D          FCC  /==========================/
 772  04ED 0D 04                FCC  13,4
 773  04EF 45 73 63 61   FREEM  FCC  /Escape = Control Menu          /
 774  050E 42 75 66 66          FCC  /Buffer Free  $/,4
 775  051D 43 74 72 6C   USEDM  FCC  /Ctrl-S = Pause or  Go          /
 776  053C 42 75 66 66          FCC  /Buffer Used  $/,4
 777  054B 20 20 20 20   ALLNM  FCC  /                            /
 778  056A 42 75 66 66          FCC  /Buffer Total $/,4
 779  0579 0D 0D         STATM2 FCC  13,13
 780  057B 20 20 20 20          FCC  /      Function                    /
 781  059E 20 20 20 20          FCC  /   Status/,13
 782  05A9 20 20 20 20          FCC  /      ~~~~~~                      /
 783  05CC 20 20 20 20          FCC  /    ~~~~~~/,4
 784  05D7 0D 20 20 20   FMESS  FCC  13,/      Finished Utility/
 785  05EE 0D 20 20 20          FCC  13,/      ================/,13,4
 786  0607 0D 20 20 20   ABNESS FCC  13,/      Utility Aborted/
 787  061D 0D 20 20 20          FCC  13,/      ================/,13,4
 788  0635 0D 20 20 20   RETM   FCC  13,/      In Emulation Mode/
 789  064D 0D 20 20 20          FCC  13,/      ================/,13,4
 790  0667 0D 20 20 20   ERMESS FCC  13,/      - Error - Hit Any Key/
 791  0683 0D 20 20 20          FCC  13,/      ~~~~~~~~~~~~~~~~/
 792  069F 0D 04                FCC  13,4
 793  06A1 0D 20 20 20   OVMESS FCC  13,/      - Overflow - Hit Any Key/
 794  06C0 0D 20 20 20          FCC  13,/      ========================/
 795  06DF 0D 04                FCC  13,4
 796  06E1 0D 20 20 20   OVSAVM FCC  13,/      - Overflow - Saving File/
 797  0700 0D 20 20 20          FCC  13,/      ========================/
 798  071F 0D 04                FCC  13,4
 799  0721 20 20 20 20   SPACER FCC  /    /
 800  0726 0D 4E 65 77   FNAMM  FCC  13,/New File Name ? /,4
 801  0738 0D 45 78 69   FNAMM  FCC  13,/Existing File Name ? /,4
 802  074F 0D 49 6E 70   ISPEDM FCC  13,/Input Delay Number /
 803  0763 28 30 2D 46          FCC  /(0-FFFF or RET) ? /,4
 804  0776 31 20 20 20   MFDUP  FCC  /1   = Set   Half  Duplex      /
 805  0795 28 20 49 6E          FCC  /( In Full Duplex Mode )/,4
 806  07AD 31 20 20 20   MHDUP  FCC  /1   = Set   Full  Duplex      /
 807  07CC 28 20 49 6E          FCC  /( In Half Duplex Mode )/,4
 808  07E4 32 20 20 20   MBST   FCC  /2   = Don't Translate  BS     /
 809  0803 28 20 54 72          FCC  /( Translate BS to DEL )/,4
 810  081B 32 20 20 20   MBSNT  FCC  /2   = Translate BS to DEL     /
 811  083A 28 20 44 6F          FCC  /( Don't Translate  BS )/,4
 812  0852 33 20 20 20   NSPCM  FCC  /3   = Space   Compression     /
```

```
0871 28 20 4E 6F          FCC   /( No Space Compress )/,4
0889 33 20 20 20 HSPCM    FCC   /3 = No Space Compress /
08A8 28 20 53 70          FCC   /( Space Compression )/,4
08C0 34 20 20 20 SPEEDM   FCC   /4 = Change Delay Speed /
08DF 28 20 44 65          FCC   /( Delay Number /,4
08F1 35 20 20 20 F1CM     FCC   /5 = Open Log File /
0910 28 20 4C 6F          FCC   /( Log File Closed )/,4
0928 35 20 20 20 F1OM     FCC   /5 = Close Log File /
0947 28 20 4C 6F          FCC   /( Log File Opened )/,4
095F 36 20 20 20 F2CM     FCC   /6 = Open Storage File /
097E 28 20 53 74          FCC   /( Storage File Closed )/,4
0996 36 20 20 20 F2OM     FCC   /6 = Close Storage File /
09B5 28 20 53 74          FCC   /( Storage File Opened )/,4
09CD 37 20 20 20 FOCM     FCC   /7 = Open Output File /
09EC 28 20 4F 75          FCC   /( Output File Closed )/,4
0A04 37 20 20 20 FOOM     FCC   /7 = Close Output File /
0A23 28 20 4F 75          FCC   /( Output File Opened )/,4
0A3B 38 20 20 20 MENU     FCC   /8 = Exit to Flex System/,13
0A55 39 20 20 20          FCC   /9 = Abort this Utility/,13
0A6F 52 45 54 20          FCC   /RET = Back to Emulation/,13
0A89 0D 45 6E 74          FCC   13,/Enter Command ? /,04
                          *
                          * Now Insert Uart Routines (LIB UART)
                          * ====================================
```


```
                        ****************************************************
                        *
                        * Uart Specific Routines for File Transfer, Terminal
                        * ---------------------------------------------------
                        *
                        * Written For a MC6850 Uart
                        * -------------------------
                        *
                        * By M. K. Hawes
                        * --------------
                        *
                        * Last Updated 3 January 1984
                        * ---------------------------
                        *
                        *
                        * SERIAL CARD ROUTINES
                        * --------------------
001B                    ESC     EQU   $1B       Keyboard Break Char
0005                    SLOT    EQU   $5        I/O Slot for Apple
A0D5                    PARREG  EQU   $A085+(SLOT*$10) Parameter Switch Add
A0D6                    CTLREG  EQU   $A086+(SLOT*$10) Control/Status Reg
A0D7                    DATREG  EQU   $A087+(SLOT*$10) Data Register Add
A000                    KBDDAT  EQU   $A000     Keyboard Date Add
A010                    KBDCLR  EQU   $A010     Clear Keyboard
                        *
                        * Initialise Uart
                        * ----------------
                        *
                        * ---------------------------------------------------
                        * This Routine Must Initialise The Uart For       *
                        * Two Stop Bits And An 8 Bit Character Without    *
                        * Any Parity Bits. If Necessary It Must Set Up    *
                        * The Relevant Transfer Speed.                    *
                        * ---------------------------------------------------
0A9B 34   02            UINIT   PSHS  A
0A9D 86   03                    LDA   #$03      Reset UART
0A9F B7   A0D6                  STA   CTLREG
0AA2 86   11                    LDA   #$11      Set for /16 and 2 Stop Bits
0AA4 B7   A0D6                  STA   CTLREG
0AA7 35   02                    PULS  A
0AA9 39                         RTS
                        *
                        * State of Uart (Zero Flag set = No Char)
                        * ---------------------------------------
                        *
                        * ---------------------------------------------------
                        * This Routine Must Check If There Is a Char      *
                        * waiting to be input by the Uart. If there is    *
                        * a char waiting the Zero Flag Must Be Reset,      *
                        * and if there is no Char the Zero Flag must      *
                        * be set.                                         *
                        *                                                 *
                        * Char Waiting      = Zero Flag Reset             *
```

```
                        * Char Not Waiting = Zero Flag Set                    *
                        * ---------------------------------------------------
0AAA 34  02     USTAT   PSHS    A
0AAC B6  A0D6           LDA     CTLREG      Check Control Register
0AAF 84  21             ANDA    #$21        Check Overflow and Char
0AB1 35  02             PULS    A
0AB3 39                 RTS
                        *
                        * Input Character from Uart Into Acc
                        * ---------------------------------
                        *
                        * ---------------------------------------------------
                        * This Routine Must Input The Char From The Uart *
                        * into the A Register.                           *
                        * ---------------------------------------------------
0AB4 B6  A0D7   UIN     LDA     DATREG      Get Character
0AB7 39                 RTS
                        *
                        * Output Char from Acc to Uart
                        * ---------------------------
                        *
                        * ---------------------------------------------------
                        * This Routine Must Output The Char in the A Reg *
                        * from the Uart. If the Previous Char has not    *
                        * been sent yet then this routine must first     *
                        * wait for it to be sent then send the Char      *
                        * in the A Register.                             *
                        * ---------------------------------------------------
0AB8 34  02     UOUT    PSHS    A
0ABA B6  A0D6   UOUT1   LDA     CTLREG      Check If Last Char Sent
0ABD 84  02             ANDA    #$02
0ABF 27  F9             BEQ     UOUT1       If Not Then Wait
0AC1 35  02             PULS    A
0AC3 B7  A0D7           STA     DATREG      Output Character
0AC6 39                 RTS
                        *
                        * The Next Routines are Used by the Terminal Program
                        * **************************************************
                        *
                        * Get Parameter Register Value
                        * ---------------------------
                        *
                        * ---------------------------------------------------
                        * This Routine Inputs The Parameter Switches     *
                        * used to set up Terminal Characteristics. The   *
                        * format of the Switches is 0 1 0 1 0 S6 S5 S4   *
                        * and this is put in the A Reg.                  *
                        * ---------------------------------------------------
0AC7 B6  A0D5   PARIN   LDA     PARREG      Input from Parameter Reg
0ACA 39                 RTS                 Return from Subroutine
                        *
                        * Non Echo Keyboard Input
                        * ----------------------
                        *
```

```
946                     * ------------------------------------------------
947                     * This Routine Inputs a Key From the Keyboard    *
948                     * without echoing it on the Screen. The Key      *
949                     * value is placed in the A Reg.                  *
950                     * ------------------------------------------------
951  0ACB B6  A000      KEYIN   LDA     KBDDAT  Input From Keyboard
952  0ACE 84  7F                ANDA    #$7F    Strib MSB
953  0AD0 B7  A010              STA     KBDCLR  Clear Keyboard Strobe
954  0AD3 39                    RTS             Return From Subroutine
955                     *
956                     * End Of Uart Routines
957                     * --------------------
958                     *
959                     ************************************************
```

```
  961                     *
  962                     * End of Program
  963                     * ---------------
  964  0AD4 00            ENDPGM  FCB     00          Dummy Byte
  965                             END     TERM

0 ERROR(S) DETECTED
```

SYMBOL TABLE:

ABMESS	0607	ABORT	0456	ACTVST	0002	ADDBX	CD36	ALLMM	054B
BACREC	0016	BEGSEC	0012	BEGTRK	0011	BSTCHK	016D	*BUFEP*	*047F*
BUFFER	0040	BUFSP	047D	CERRR	*0240*	*CHKCHR*	00B2	CHOUT	0077
CLASS	CD21	*CLCLOS*	*031F*	CLEFLF	0308	CLGEXT	0324	CLOSE	0004
CLOSEF	02FD	CLOTEX	03F2	CLOUT	03D9	COLDS	CD00	COMPRS	00A2
CTLREG	A0D6	CTRLS	0013	DATNDX	0022	DATREG	*A0D7*	*DELAY*	00D0
DELCTR	0485	DELETE	000C	*DIRADD*	*002F*	DOCMND	CD4B	DRVNUM	0003
ENDFIL	*038E*	*ENDPGM*	DAD4	ENDSEC	0014	ENDTRK	0013	ERMESS	0667
ERROR	0425	ERSTAT	0001	ESC	001B	ESETRG	CC16	EXIT	043A
FICM	08F1	FIFLAG	0477	F10	01A2	FIOM	*0928*	*F2CM*	*095F*
F2FLAG	0478	F20	01BC	F20M	0996	F2PS	01AE	FATTRB	000F
FCB	*047B*	FCB1	C100	FCB2	C240	FCBLFT	001C	FCBO	C380
FCDATE	0019	FDELDP	0032	FEXTEN	000C	FFLAG	0483	FGETCH	037B
FILCHK	035E	FILE1	0296	FILE2	02A6	FILEO	*02B6*	FILEXT	039F
FILSIZ	0015	FILTYP	0474	FNAMM	*0738*	FMESS	05D7	FMS	D406
FMSCLS	D403	FMSINT	D400	FNAME	0004	FNAMM	0726	FOCM	09CD
FOFLAG	0479	FOO	01D6	FOOM	0A04	FOPS	01C8	FPTR	0481
FREEM	04EF	FSECNT	0017	FUN5	0256	*FUN5B*	0264	FUN5E	0267
FUN6	0269	FUN6B	0277	FUN6E	027A	FUN7	0282	FUN7B	0291
FUN7E	0294	FUNCTN	0000	GETCHR	CD15	GETFIL	CD2D	GETHEX	CD42
GETINF	0007	GETRND	0011	GETSEC	0009	HALFDU	0475	HSPCM	0889
INBUFF	CD1B	INCH	CD09	INCH2	CD0C	*INDEC*	*CD4B*	INIT	000E
ISANO	0149	ISPEIM	*074F*	KBDCHK	0048	KBDCLR	A010	KBDDAT	A000
KBDEXT	0075	KEYIN	0ACB	LOAD	CD30	LOGIN	034A	LOGOUT	035C
LOGPS	017E	LOOP	0005	MBSNT	081B	MBST	07E4	*MEMEND*	*CC2B*
MENU	0A3B	MFDUP	0776	MHDUP	*07AD*	*NAMEBF*	0024	NBSTRL	0476
NCTQ	0067	*NCTS*	0070	NESC	005B	NORCLS	039A	NOWSS	01E2
NSCOM	0180	NSPCM	0852	NSTR1P	0098	NUMSTR	0487	NXTCH	CD27
NXTDRV	0014	NXTSEQ	000F	OPENF	02C6	OPENRD	0001	*OPENUP*	*0003*
OPENWR	0002	OPNDTB	0006	OPNOUT	*03A1*	*OSIREC*	0010	OUTADR	CD45
OUTCH	CD0F	*OUTCH2*	*CD12*	OUTDEC	CD39	OUTHEX	CD3C	OVERDR	0418
OVSAVE	06A1	OVSAVM	06E1	PARIN	0AC7	PARREG	A0D5	PCRLF	CD24
PFNAME	0DE1	PIEXIT	0348	PIFILE	0326	PNAME	0106	*PNLOOP*	*010F*
PNNDT	0100	PRFREE	00A6	*PRINTD*	*0004*	PSPEED	0119	PSTRNG	CD1E
PUREXT	0416	PURGE	03F4	PUTCHR	CD18	PUTINF	0008	PUTRND	0012
PUTSEC	000A	RDWR	0000	RECNUM	0015	RECORD	0020	RENAME	000D
RENTER	CD06	RES1	000B	RES2	000E	RES3	0013	RES4	0010
RES5	0018	RETM	0635	REWIND	*0005*	RNDNDX	0023	RTERR	CD3F
RSTRIO	CD2A	SCWTCH	0035	SECTOR	001F	SETEXT	CD33	SETSTT	014E
SEXIT	024E	SHFDUP	016A	SLOT	0005	SNBST	017B	SPACER	0721
SPDCHK	018F	SPEED	0472	SPEEIM	08C0	SSAGN	0150	SSPEED	0133
SSSAGN	0135	STAT	CD4E	*STATM1*	*0489*	*STATM2*	0579	TERM	0000
TERM1	*0003*	TR1	01EF	TR2	01F5	TR3	0203	TR4	020D
TR5	0217	TR6	0220	TR7	0229	TR8	0232	TR9	0239
TRACK	001E	UIN	0AB4	UINIT	0A9B	UOUT	0AB8	UOUT1	0ABA
URTCHK	0087	URTEXT	00A4	USBIN	051D	USTAT	0AAA	VN	0002
WAIT	047A	WARMS	CD03						

APPENDIX I

LOGIC-SIMULATOR PRIMITIVES

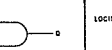
INTRODUCTION

These are the primitives that can be used in generating the circuit to be simulated. A drawing of the primitive is followed by the format of the input needed for the particular primitive.

LOGIC TYPE: AND GATE

LOGIC EQUATION: $O = A \cdot (B) \cdot (C) \cdot (D) \cdot (E)$

LOGIC SYMBOL:



TRUTH TABLE:

A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

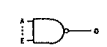
ELEMENT CODING: AND A (B) (C) (D) (E) O ($\overline{O}$)

NOTES: ( ) DENOTES OPTIONAL FIELDS, "0" IF NOT USED.

LOGIC TYPE: NAND GATE

LOGIC EQUATION: $O = \overline{A \cdot (B) \cdot (C) \cdot (D) \cdot (E)}$

LOGIC SYMBOL:



TRUTH TABLE:

A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

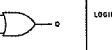
ELEMENT CODING: NAND A (B) (C) (D) (E) O ($\overline{O}$)

NOTES: ( ) DENOTES OPTIONAL FIELDS, "0" IF NOT USED.

LOGIC TYPE: OR GATE

LOGIC EQUATION: $O = A + (B) + (C) + (D) + (E)$

LOGIC SYMBOL:



TRUTH TABLE:

A	B	O
0	0	0
0	1	1
1	0	1
1	1	1

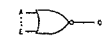
ELEMENT CODING: OR A (B) (C) (D) (E) O ($\overline{O}$)

NOTES: ( ) DENOTES OPTIONAL FIELDS, "0" IF NOT USED.

LOGIC TYPE: NOR GATE

LOGIC EQUATION: $O = \overline{A + (B) + (C) + (D) + (E)}$

LOGIC SYMBOL:



TRUTH TABLE:

A	B	O
0	0	1
0	1	0
1	0	0
1	1	0

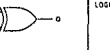
ELEMENT CODING: NOR A (B) (C) (D) (E) O ($\overline{O}$)

NOTES: ( ) DENOTES OPTIONAL FIELDS, "0" IF NOT USED.

LOGIC TYPE: EXCLUSIVE OR GATE

LOGIC EQUATION: $O = A \oplus B$

LOGIC SYMBOL:



TRUTH TABLE:

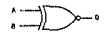
A	B	O
0	0	0
0	1	1
1	0	1
1	1	0

ELEMENT CODING: XOR A B O O O O ($\overline{O}$)

LOGIC TYPE: EXCLUSIVE NOR GATE

LOGIC EQUATION: $O = \overline{A \oplus B}$

LOGIC SYMBOL:



TRUTH TABLE:

A	B	O
0	0	1
0	1	0
1	0	0
1	1	1

ELEMENT CODING: XNOR A B O O O O ($\overline{O}$)

LOGIC TYPE: D-TYPE FLIP-FLOP

LOGIC SYMBOL:

TRUTH TABLE:

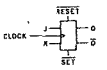
DATA	CLOCK	RESET	SET	Q	Q̄
X	X	0	0	0	1
X	X	1	0	1	0
X	X	0	1	0	1
0	↓	1	1	0	1
1	↓	1	1	1	0
X	↑	1	1	Q	Q̄

ELEMENT CODING: DFF DATA Q CLOCK ($\overline{RESET}$) ($\overline{SET}$) Q ($\overline{Q}$)

NOTES: X = DON'T CARE.
SET AND RESET ARE ACTIVE LOW; IF UNUSED "0", THEY DEFAULT TO LOGIC 1.

LOGIC TYPE: JK FLIP-FLOP

LOGIC SYMBOL



TRUTH TABLE:

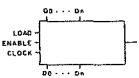
J	K	SET	RESET	Q	Q̄	CLOCK	NEXT Q	NEXT Q̄
1	X	1	1	0	1	↓	1	0
X	0	1	1	1	0	↓	1	0
0	X	1	1	0	1	↓	0	1
X	1	1	1	1	0	↓	0	1
X	X	1	1	X	X	↑	NO CHANGE	
X	X	0	1	X	X	X	1	0
X	X	1	0	X	X	X	0	1
X	X	0	0	X	X	X	0	1

ELEMENT CODING: JKFF J K CLOCK ($\overline{RESET}$) ($\overline{SET}$) Q ($\overline{Q}$)

NOTES: X = DON'T CARE.
SET AND RESET ARE ACTIVE LOW, IF UNUSED "0", THEY DEFAULT TO LOGIC 1.

LOGIC TYPE: N-BIT COUNTER

LOGIC SYMBOL:



TRUTH TABLE:

DATA	LOAD	ENABLE	RESET	CLOCK	COUNT IQ0 . . . QN1
0	1	X	1	↓	0
1	1	X	1	↓	1
X	0	X	0	X	0
X	0	1	1	↓	COUNT + 1
X	0	1	1	↓	NO CHANGE
X	0	0	1	↓	NO CHANGE

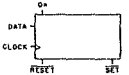
ELEMENT CODING:

CNTR	IDQI	ENABLE	CLOCK	(LOAD)	($\overline{RESET}$)	QO	($\overline{QO}$)
*	(DI)	0	S	0	0	0	Q1 ($\overline{Q1}$)
*	.	0	0	0	0	.	.
*	Dn					On	($\overline{On}$)

NOTES: X = DON'T CARE. LOAD AND ENABLE ARE ACTIVE HIGH AND LOAD OVERRIDES ENABLE.

LOGIC TYPE: N-BIT SHIFT REGISTER

LOGIC SYMBOL:



TRUTH TABLE:

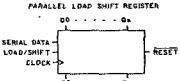
DATA	$\overline{RESET}$	$\overline{SET}$	CLOCK	Q1	On
0	1	1	↓	0	On-1
1	1	1	↓	1	On-1
X	1	1	↑	Q1	On NO CHANGE
X	0	1	X	0	0
X	1	0	X	1	1

ELEMENT CODING: SREG DATA #BITS CLOCK ($\overline{RESET}$) ($\overline{SET}$) On ($\overline{On}$)

NOTES, #BITS IS THE TOTAL NUMBER OF SHIFT BITS REQUIRED, IF "0", THEN DEFAULTS TO 1.
X = DON'T CARE.

LOGIC TYPE: PARALLEL LOAD SHIFT REGISTER

LOGIC SYMBOL:



TRUTH TABLE:

D0	DSER	LOAD	$\overline{RESET}$	CLOCK	Q0	Qn
X	X	X	0	X	0	0
1	X	1	1	X	1	1
0	X	1	1	X	0	0
X	0	0	1	↓	Q	Qn-1
X	1	0	1	↓	1	Qn-1

ELEMENT CODING:

PLSR	D0	DSER	CLOCK	LOAD	($\overline{RESET}$)	D0	($\overline{D0}$)
#	D1	0	0	0	0	D1	($\overline{D1}$)
#	Qn	0	0	0	0	Qn	($\overline{Qn}$)

NOTES; X = DON'T CARE
LOAD/SHIFT= LOAD ON HIGH, SHIFT ON LOW.

APPENDIX J

LOGIC-SIMULATOR LISTING

INTRODUCTION

This listing contains the program written in BASIC followed by the circuit to be simulated contained at the end in DATA statements. The circuit diagram with the signal names is contained in the next Appendix and the actual output of the simulator in the following Appendi .

INTRODUCTION

This listing contains the program written in BASIC followed by the circuit to be simulated contained at the end in DATA statements. The circuit diagram with the signal names is contained in the next Appendix and the actual output of the simulator in the following Appendix.

```
10 REM  AN ENHANCED LOGIC SIMULATION PROGRAM
20 REM  ====================================
30 REM         Robert M. Mcdermott
40 REM            AUGUST 1981
50 REM
60 REM
70 REM Adapted for the Nascom 1 Computer by
80 REM Michael Kerrigan Hawes. (June 1983)
90 REM
100 REM ---------------------------------------
110 REM FORMAT
120 REM ======
130 REM MACRO name1
140 REM ...
150 REM END
160 REM MACRO name2
170 REM ...
180 REM END
190 REM ELEMENTS
200 REM ...
210 REM END
220 REM EXTERNALS
230 REM ...
240 REM END
250 REM OUTPUTS
260 REM ...
270 REM END
280 REM %%%
290 REM ---------------------------------------
300 REM ****************************************
310 REM The following Program simulates a      *
320 REM Digital Logic design. The Logic        *
330 REM description may consist of Logic       *
340 REM 'Primitives' (AND, OR, XOR, ETC Gates *
350 REM D, JK Flip Flops; N-Bit Counter) or    *
360 REM 'Macros' ( Blocks of commonly used     *
370 REM primitives ). Each Logic Element       *
380 REM contains 5 inputs and 2 outputs,       *
390 REM unused 'Pins' are coded as '0'.        *
400 REM                                        *
410 REM The Logic is driven by 'External       *
420 REM Stimuli'. The Stimuli are described    *
430 REM as Periodic or Aperiodic, the signal   *
440 REM name, starting logic value, start      *
450 REM time, and 5 change times.              *
460 REM                                        *
470 REM ****************************************
480 REM
490 REM
500 CLEAR1000:WIDTH255:PRINTCHR$(12)
510 REM ***************************
520 REM *   Automatic Continue    *
530 REM ***************************
540 PRINT"Do You Want Automatic Continuing ? ";
```

```
550 AC=-1:SI$="Y":GOSUB1200
560 IFQ$<>"Y"ANDQ$<>"N"THEN540
570 IFQ$="N"THENAC=0
580 REM **********************
590 REM *   Printer Check    *
600 REM **********************
610 PRINT"Do You Want The Output Printed   ? ";
620 SI$="N":GOSUB1200
630 IFQ$<>"Y"ANDQ$<>"N"THEN610
640 PR=1:TN=8:IFQ$="N"THENPR=0:TN=6:GOTO760
650 START+:HEADING"Digital Simulator Output"
660 PRT+
670 REM *************************
680 REM *   START OF PROGRAM    *
690 REM *************************
700 REM ******************************************
710 REM The following parameters define the    *
720 REM maximum array limits. They can be      *
730 REM adjusted to fit a design to available *
740 REM Ram.                                   *
750 REM ******************************************
760 MNAMS=200:MLGIC=200:MEXTERN=30:MSAMPLES=14
770 MTYPES=50
780 PRINT:PRINTTAB(15*(1+PR));"LOGIC SIMULATOR"
790 PRINTTAB(15*(1+PR));"----------------":PRINT
800 PRINTMN;"NAMES":PRINTML;"LOGIC DEVICES"
810 PRINTMT;"TYPES":PRINTME;"EXTERNAL STIMULI"
820 PRINT:PRINT
830 DIMNAMS$(MNAMS),SRT(MNAMS)
840 DIMLO(MLGIC,8)
850 DIMEXTERNALS(MEXTERNALS,10)
860 DIMTYPE$(MTYPES)
870 DIMTYPEP(MTYPES)
880 DIMOL(MN)
890 DIMSA(MS)
900 DIMT(10),LI$(10),ST(672)
910 REM ***************************
920 REM *    Start of Program    *
930 REM ***************************
940 PRINT:PRINT"*** Initialising Arrays    ***"
950 PRINT"*** --------------------    ***"
960 GOSUB1330
970 PRINT:PRINT"*** Reading Any Macros     ***"
980 PRINT"*** --------------------     ***"
990 GOSUB1530
1000 PRINT:PRINT"*** Read In Logic Network ***"
1010 PRINT"*** ---------------------- ***"
1020 PRINT:GOSUB1990
1030 PRINT:PRINT"*** Expanding Any Macros  ***"
1040 PRINT"*** ---------------------  ***"
1050 GOSUB2200
1060 GOSUB2330
1070 PRINT:PRINT"*** Read External Stimili ***"
1080 PRINT"*** ---------------------- ***"
```

```
1090 PRINT:GOSUB2640
1100 PRINT:PRINT"*** Checking for Errors   ***"
1110 GOSUB2930
1120 PRINT:PRINT"*** Read Nodes To Sample  ***"
1130 PRINT"*** ---------------------  ***"
1140 PRINT:GOSUB3040
1150 PRINT:PRINT"*** Setting Up Simulation ***"
1160 PRINT"*** ---------------------- ***"
1170 GOSUB3920
1180 IFPRORHPTHENPRT+:PRINTCHR$(12):START-
1190 END
1200 REM ******************************
1210 REM *   Special Input Routine    *
1220 REM ******************************
1230 SC=0:PRINT" ";
1240 GETQ:IFQ<>0THEN1310
1250 SC=SC+1:IFSC>600ANDACTHEN1300
1260 SN=SN+1
1270 IFSN>5ANDSF=0THENPRINTCHR$(8);" ";:SF=1:SN=0
1280 IFSN>5ANDSF=1THENPRINTCHR$(8);"_";:SF=0:SN=0
1290 GOTO1240
1300 Q=ASC(SI$)
1310 Q$=CHR$(Q):PRINTCHR$(8);Q$
1320 RETURN
1330 REM ****************************
1340 REM *   Initialise Pointers    *
1350 REM *         [ 500 ]          *
1360 REM ****************************
1370 NL=0:NTYPE=0
1380 READQ$
1390 IFQ$<>"LAST"THENTY$(NT)=Q$:NT=NT+1:GOTO1380
1400 DATA "END","AND","NAND","OR","NOR","XOR"
1410 DATA "XNOR","DFF","JKFF","CNTR","FLSR"
1420 DATA "SREG","*","LAST"
1430 NPRIM=NTYPE:NTYPE=NTYPE+1
1440 REM Logic Print Values and Inversions
1450 L0=0:L1=3:LX=1
1460 DIMLV$(3),LP$(3),LN(3)
1470 LV$(L0)="0":LP$(L0)="-":LN(L0)=L1:LV(L0)=L0
1480 LV$(L1)="1":LP$(L1)="1":LN(L1)=L0:LV(L1)=L1
1490 LV$(LX)="X":LP$(LX)="?":LN(LX)=LX:LV(LX)=LX
1500 DIMTEX$(1):FORI=0TO1:READTEX$(I):NEXTI
1510 DATA "A","P"
1520 RETURN
1530 REM ***********************
1540 REM *   Read In Macros    *
1550 REM *     [ 1000 ]        *
1560 REM ***********************
1570 GOSUB6190
1580 IFLEFT$(L$,3)<>"MAC"THENRETURN
1590 X$=LI$(1)
1600 PRINT
1610 PRINT"*** Reading in Macro : ";X$;" ***"
1620 PRINT:TFTR=-1
```

```
1630 GOSUB6380
1640 IFFOUND=-1THEN1710
1650 IFITFTR=-1THEN1710
1660 PRINT"This Macro Name Already Exists"
1670 PRINT"Enter R(eplace) or N(oreplace)";
1680 SI$="R":GOSUB1200
1690 IFQ$="N"THEN1760
1700 IFQ$<>"R"THEN1670
1710 TPTR=NLGIC
1720 GOSUB6470:MT=FO
1730 GOSUB2070
1740 GOSUB1820
1750 GOTO1530
1760 REM ************************
1770 REM *   Skip This Macro    *
1780 REM *       [ 1200 ]       *
1790 REM ************************
1800 GOSUB6190:IFLI$(0)<>"END"THEN1800
1810 GOTO1530
1820 REM *******************************
1830 REM *   Check for Macro Errors    *
1840 REM *        [ 1500 ]             *
1850 REM *******************************
1860 PRINT:PRINT"*** Checking Macro ";TY$(MT);
1870 PRINT" for Errors ***"
1880 X1=TY(MT):GOSUB6810
1890 FORI=1TO5
1900 X=LO(X1,I):GOSUB6370
1910 NEXTI
1920 X1=X1+1
1930 X=LO(X1,0):IFTY$(X)="*"THEN1890
1940 ST=X1
1950 GOSUB7010
1960 GOSUB7100
1970 ER=0
1980 RETURN
1990 REM ***********************************
2000 REM *   Read In Logic Description     *
2010 REM *           [ 2000 ]              *
2020 REM ***********************************
2030 IFLEFT$(L$,8)="ELEMENTS"THEN2070
2040 PRINT:PRINT"Error, ELEMENTS Expected."
2050 IFPRORHPTHENPRT+:PRINTCHR$(12):START
2060 END
2070 SL=NL:NA$(0)="0":NNAM=1
2080 FORI=1TOMN:NA$(I)="":NEXTI
2090 GOSUB6190
2100 X$=LI$(0)
2110 GOSUB6380:IFFO=-1THENGOSUB6470
2120 T(0)=FO
2130 FORI=1TO7
2140 X$=LI$(I):GOSUB6560
2150 IFFO=-1THENGOSUB6730
2160 T(I)=FO:NEXTI
```

```
2170 GOSUB7300
2180 IFT(0)<>0THEN2090
2190 RETURN
2200 REM **********************
2210 REM *   Expand Macros    *
2220 REM *      [ 3000 ]      *
2230 REM **********************
2240 ST=SL
2250 FORI=STTONL-1
2260 IFLO(I,0)>NPRIMTHEN2290
2270 NEXTI
2280 RETURN
2290 GOSUB7590
2300 GOSUB8000
2310 REM Repeat Proc to Find Next Macro
2320 GOTO2200
2330 REM ***************************************
2340 REM *   List Expanded Logic Description   *
2350 REM *            [ 3500 ]                 *
2360 REM ***************************************
2370 PRINT:PRINT
2380 PRINT"*** Expanded Logic Descr..ion ***"
2390 PRINT"*** ------------------------ ***"
2400 PRINT:PRINT"No.";TAB(TN);
2410 PRINT"Type";TAB(4*TN);"Inputs";
2420 PRINTTAB(7*TN);"Outputs"
2430 FORI=1TO5:PRINTTAB(TN*I+TN-1);I;:NEXTI
2440 PRINTTAB(7*TN);"Q";TAB(8*TN);"Q~"
2450 FORI=0TO8:PRINT"+";
2460 FORII=1TOTN-1:PRINT"-";
2470 NEXTII:NEXTI:PRINT"+"
2480 NA$(0)="."
2490 FORI=SLTONL-1
2500 IFI=NL-1THEN2540
2510 PRINT"(";
2520 PRINTRIGHT$(STR$(I-SL+1),LEN(STR$(I-SL+1))-1);
2530 PRINT")";
2540 PRINTTAB(TN);
2550 PRINTTY$(LO(I,0));
2560 FORII=1TO7
2570 PRINTTAB(TN*II+TN);
2580 XX=LO(I,II)
2590 IFXX<0THENPRINT"/";-XX;:GOTO 2600
2600 PRINTNA$(XX);" ";
2610 NEXTII:PRINT:NEXTI
2620 NA$(0)="0"
2630 RETURN
2640 REM ************************************
2650 REM *   Read In External Description   *
2660 REM *            [ 4000 ]              *
2670 REM ************************************
2680 PRINT"No.";TAB(TN);"Type";TAB(2*TN);"Name";
2690 PRINTTAB(3*TN);"Val";TAB(7/2*TN+1);"Egn";
2700 FORI=1TO6:PRINTTAB((I+7)*TN/2+1);"Os";
```

```
2710 PRINTRIGHT$(STR$(I),1);:NEXTI:PRINT
2720 FORI=1TO3:PRINT"+";:FORII=1TOTN-1
2730 PRINT"-";:NEXTII:NEXTI:PRINT"+----";
2740 FORI=1TO7:PRINT"+";:FORII=1TOTN/2-1
2750 PRINT"-";:NEXTII:NEXTI:PRINT"+"
2760 NEX=0:GOSUB6190
2770 IFLEFT$(L$,9)<>"EXTERNALS"THEN2810
2780 PRINT:PRINT"Error, EXTERNALS Expected."
2790 IFPRORHTHENPRT+:PRINTCHR$(12):START-
2800 END
2810 GOSUB6190
2820 IFLI$(0)="END"THENPRINTTAB(TN);"END":RETURN
2830 FORI=0TO1:IFLI$(0)=EX$(I)THENI(0)=I
2840 NEXTI
2850 X$=LI$(1):GOSUB6560
2860 IFF0=-1THEN2810
2870 T(1)=F0
2880 FORI=0TO3:IFLI$(2)=LV$(I)THENI(2)=LV(I)
2890 NEXTI
2900 FORI=3TO9:T(I)=VAL(LI$(1)):NEXTI
2910 GOSUB7430
2920 GOTO2810
2930 REM ************************
2940 REM *   Check for Errors   *
2950 REM *      [ 5000 ]        *
2960 REM ************************
2970 GOSUB6810:MM=0
2980 FORI=0TONEX
2990 OL(EX(I,1))=1
3000 NEXTI
3010 ST=SL:GOSUB7010
3020 GOSUB7100
3030 RETURN
3040 REM ******************************
3050 REM *   Read in Sample Points    *
3060 REM *         [ 6000 ]           *
3070 REM ******************************
3080 NS=0
3090 GOSUB6190
3100 IFLEFT$(L$,7)="OUTPUTS"THEN3140
3110 PRINT:PRINT"Error, OUTPUTS Expected."
3120 IFPRORHTHENPRT+:PRINTCHR$(12):START-
3130 END
3140 GOSUB6190
3150 IFLI$(0)="END"THEN3220
3160 FORI=0TO10
3170 X$=LI$(I):IFX$=" "THEN3200
3180 GOSUB6560:IFF0=-1THEN3200
3190 IFNS<14THENNS=NS+1:SA(NS)=F0
3200 NEXTI
3210 GOTO3140
3220 IFNS>7THEN3260
3230 FORI=1TONS
3240 PRINTTAB((I-1)*TN);NA$(SA(I));
```

```
3250 NEXT I:PRINT:PRINT "END":RETURN
3260 FOR I=1TO7
3270 PRINTTAB((I-1)*TN);NA$(SA(I));
3280 NEXT I:PRINT:FOR I=8TO NS
3290 PRINTTAB((I-8)*TN);NA$(SA(I));
3300 NEXT I:PRINT:PRINT "END":RETURN
3310 REM ************************************
3320 REM *   Selective Trace Table Build    *
3330 REM *              [ 6700 ]            *
3340 REM ************************************
3350 STF=0
3360 A1=(NL+NN+(TL+NN))*2
3370 A2=FRE(0)-200
3380 IF A2<A1 THEN RETURN
3390 PRINT
3400 PRINT "*** Build Selective Trace Table ***"
3410 PRINT "*** ------------------------- ***"
3420 PRINT
3430 STF=1:DIM S0(NL),S1(NN),S2(TL+NN)
3440 TL=0
3450 FOR I=1TO NN:S1(I)=TL
3460 FOR J=S1TO NL:FOR K=1TO 5
3470 IF LO(J,K)=I THEN S2(TL)=J:TL=TL+1
3480 NEXT K:NEXT J
3490 S2(TL)=-1:TL=TL+1
3500 NEXT I
3510 GOSUB 3530
3520 RETURN
3530 REM *********************************
3540 REM *   Print Out Of Load Table     *
3550 REM *           [ 6700 ]            *
3560 REM *********************************
3570 FOR I=1TO NN
3580 IF LEFT$(NA$(I),1)<>"&" THEN SE=I
3590 SK(I)=I
3600 NEXT I
3610 FL=0
3620 FOR I=1TO SE-1
3630 IF NA$(SK(I))<NA$(SK(I+1)) THEN 3680
3640 II=SK(I)
3650 SK(I)=SK(I+1)
3660 SK(I+1)=II
3670 FL=-1
3680 NEXT I
3690 IF FL THEN 3610
3700 PRINT:PRINT:PRINT "*** Load Table ***"
3710 PRINT "*** ---------- ***"
3720 PRINT:PRINT "NODE";TAB(TN);"DEFINED";
3730 PRINTTAB(3*TN);"DRIVEN ELEMENTS"
3740 PRINT "----";TAB(TN);"-------";
3750 PRINTTAB(3*TN);"----------------"
3760 FOR JJ=1TO NN:PRINT NA$(SK(JJ));TAB(TN);
3770 X$=NA$(SK(JJ)):GOSUB 6560
3780 IF F0=-1 THEN 3880
```

```
3790 FORI=SLTONL-1
3800 IFLO(I,6)=FOTHENPRINT"[ Q  ";:Q=I-SL+1:GOTO3860
3810 IFLO(I,7)=FOTHENPRINT"[ Q- ";:Q=I-SL+1:GOTO3860
3820 NEXTI
3830 FORI=OTONEX
3840 IFEX(I,1)=FOTHENPRINT"[ EXT";:Q=I+1:GOTO3860
3850 NEXTI
3860 QS=RIGHT$("  "+STR$(Q)+" ]",6)
3870 PRINTQ$;TAB(3*TN);
3880 T1=S1(SRT(JJ))
3890 IFS2(T1)=-1THEN3910
3900 PRINTS2(T1)-SL+1;:T1=T1+1:GOTO3890
3910 PRINT:NEXTJJ:RETURN
3920 REM **********************
3930 REM *   Do Simulation    *
3940 REM *      [ 7000 ]      *
3950 REM **********************
3960 DIMNE(NN),ME(MM,1):IFER<>OTHENRETURN
3970 NN=NN-1:NL=NL-1:NE=NE-1:QA=0
3980 GOSUB3310
3990 GOSUB4080
4000 GOSUB4460
4010 GOSUB4830
4020 GOSUB5030
4030 IFQS="Q"THENRETURN
4040 IFQS="R"THEN3990
4050 GOSUB8230
4060 GOSUB4390
4070 GOTO4000
4080 REM *********************************
4090 REM *   Initialize for Simulation   *
4100 REM *          [ 7300 ]             *
4110 REM *********************************
4120 FORI=OTONN:NE(I)=LX:NEXTI
4130 FORI=SLTONL:LO(I,8)=LX:NEXTI
4140 PRINTCHR$(12):R1=0
4150 PRINT"Default for Start & Increment ? ";
4160 S1S="Y":GOSUB1200
4170 IFQS<>"Y"ANDQS<>"N"THEN4150
4180 S9=0:S2=1
4190 IFQS="Y"THEN4220
4200 INPUT"Enter Sample Start Time ";S9
4210 INPUT"Enter Increment Time    ";S2
4220 S9=INT(S9):S2=INT(S2)
4230 IFPRTHENPRT-
4240 IFS2=0THENS2=1
4250 E9=32767:FORI=OTONEX:E3=EX(I,3)
4260 GOSUB4760:NEXTI
4270 IFNS=0THENGOSUB5890
4280 IFMM<>0THENGOSUB4300
4290 QS="":RETURN
4300 REM *********************************
4310 REM *   Set Aside Mem for SREGs     *
4320 REM *           [ 7400 ]            *
```

```
4330 REM *********************************
4340 M1=0
4350 FORI=0TOMM:ME(I,0)=LX:ME(I,1)=LX:NEXTI
4360 FORI=0TONL
4370 IFTY$(LO(I,0))="SREG"THENLO(I,8)=M1:M1=M1-LO(I,2)
4380 NEXTI:RETURN
4390 REM *********************************
4400 REM *   Recalculate Ripple Time   *
4410 REM *          [ 7500 ]           *
4420 REM *********************************
4430 IFC9=ITHENRI=RI+1:RETURN
4440 RI=E9:IFRI>S9THENRI=S9
4450 RETURN
4460 REM ************************************
4470 REM *   Get Next Scheduled External   *
4480 REM *          [ 8000 ]               *
4490 REM ************************************
4500 IFRI<>E9THENRETURN
4510 E9=32767
4520 FORI=0TONEX
4530 E3=EX(I,10)
4540 IFE3=RITHENGOSUB4580
4550 GOSUB4760
4560 NEXTI
4570 RETURN
4580 REM *********************************
4590 REM *   Find Next Scheduled Change   *
4600 REM *          [ 8100 ]              *
4610 REM *********************************
4620 E0=EX(I,0):E1=EX(I,1)
4630 E2=EX(I,2):E3=EX(I,3)
4640 Y=NE(E1)
4650 IFRI=E3THENY=LN(E2)
4660 NE(E1)=LN(Y)
4670 REM *****************************
4680 REM *   Schedule Next Change    *
4690 REM *         [ 8150 ]          *
4700 REM *****************************
4710 IFTEX$(E0)="P"THENE3=EX(I,10)+EX(I,4):RETURN
4720 J=4:E3=32767
4730 IFJ>=9THENRETURN
4740 IFEX(I,J)<=RITHENJ=J+1:GOTO4730
4750 E3=EX(I,J):RETURN
4760 REM ***************************************
4770 REM *   Store Next Sched, Keep Track    *
4780 REM *            [ 8200 ]               *
4790 REM ***************************************
4800 EX(I,10)=E3
4810 IFE9>E3THENE9=E3
4820 RETURN
4830 REM ***************************************
4840 REM *   Move New to Old, Mark changes    *
4850 REM *          [ 8500 ]                  *
4860 REM ***************************************
```

```
4870 C9=0:IFST$<>0THENGOSUB4970
4880 FORI=1TONN
4890 IFNE(I)<>OL(I)THENGOSUB4920
4900 NEXTI
4910 RETURN
4920 C9=1:OL(I)=NE(I)
4930 IFST$=0THENRETURN
4940 II=S1(I)
4950 J=S2(II):IFJ=-1THENRETURN
4960 S0(J)=1:II=II+1:GOTO4950
4970 REM ****************
4980 REM *   Clear S0   *
4990 REM *   [ 8700 ]   *
5000 REM ****************
5010 FORII=SLTONL:S0(II)=0:NEXTII
5020 RETURN
5030 REM ****************************
5040 REM *   Display Logic Values   *
5050 REM *        [ 9000 ]          *
5060 REM ****************************
5070 SCREEN1,16:PRINT"Time";RI;:GET$QB:QA=QA+QB
5080 IFQB<>0THENSCREEN1,1:PRINT"Stopped"
5090 IFS9<>RITHENRETURN
5100 IFPC=0THENGOSUB5150
5110 GOSUB5380
5120 S9=S9+S2
5130 PC=PC+1:IF(PC>45)OR(QA<>0)THENGOSUB5640
5140 QA=0:RETURN
5150 REM ***************************************
5160 REM *   Set up Display (Names & Tics)   *
5170 REM *             [ 9100 ]              *
5180 REM ***************************************
5190 FORI=11TO47:SCREEN1,16
5200 PRINT" ";:NEXTI
5210 PC=11:FORI=1TO15:SCREEN1,I
5220 PRINTCHR$(27):NEXTJ:GOSUB5260
5230 FORI=1TONS:SCREEN1,I+1:PRINTNA$(SA(I));
5240 NEXTI
5250 RETURN
5260 REM ************************
5270 REM *   Print Tic Marks   *
5280 REM *       [ 9200 ]      *
5290 REM ************************
5300 SCREEN1,1:PRINTCHR$(27)
5310 FORX=12TO45:SCREENX,1
5320 PRINTCHR$(152);:NEXTX
5330 FORI=1TO6:SCREEN11+5*I,1:PRINTCHR$(153);
5340 NEXTI
5350 SCREEN11,1:PRINTCHR$(146);
5360 SCREEN46,1:PRINTCHR$(147);
5370 RETURN
5380 REM *********************************
5390 REM *   Print Values Down Column    *
5400 REM *        [ 9500 ]               *
```

```
5410 REM *********************************
5420 IFINT((PC-1)/5)*5<>(PC-1)THEN5440
5430 SCREENPC-1,16:PRINTRI;
5440 FORI=1TONS:Y=OL(SA(I))
5450 SCREENPC,1+I:PRINTLP$(Y);:NEXTI
5460 RETURN
5470 REM ********************
5480 REM *   Print Screen   *
5490 REM *     [ ???? ]     *
5500 REM ********************
5510 CT=1:HP=1
5520 FORI=2122TO2954STEP64
5530 FORJ=ITOI+46
5540 ST(CT)=PEEK(J):CT=CT+1
5550 NEXTJ
5560 ST(CT)=13:CT=CT+1
5570 NEXTI:SCREEN1,15:PRT+:PRINT:PRINT
5580 FORI=301ETO3064:PRINTCHR$(PEEK(I));
5590 NEXTI:PRINT:PRINT"                ";
5600 FORI=1TO7:PRINT"+---";:NEXTI:PRINT"+"
5610 FORI=1TO672:PRINTCHR$(ST(I));
5620 NEXTI:PRINT:PRT-
5630 RETURN
5640 REM *********************************
5650 REM *   Full Screen, Get Response   *
5660 REM *          [ 96C0 ]             *
5670 REM *********************************
5680 IFPR=1THENGOSUB5470
5690 SCREEN1,1
5700 PRINTCHR$(27);"C(hange),R(eset),Q(uit),";
5710 PRINT"M(ore),P(rint),N(ext` ";
5720 SI$="N":GOSUB1200
5730 SCREEN1,1:PRINTCHR$(27)
5740 IFQ$="C"ORQ$="R"ORQ$="Q"THEN5770
5750 IFQ$="M"ORQ$="P"ORQ$="N"THEN5770
5760 GOTO5690
5770 IFQ$="P"THENGOSUB5470:GOTO5690
5780 IFQ$="Q"THENGOSUB5260
5790 IFQ$="M"ANDPC>45THEN5690
5800 IFQ$="M"THENGOSUB5260:RETURN
5810 PC=0
5820 IFQ$<>"C"THENRETURN
5830 PRINTCHR$(12)
5840 PRINT"Change Signals D(isplayed) or ";
5850 PRINT"V(alues) ? ";
5860 INKEYQ:X$=CHR$(Q):PRINTX$
5870 IFX$<>"D"ANDX$<>"V"THEN5830
5880 IFX$="V"THEN5990
5890 PRINT"As Each Current Sample Node is"
5900 PRINT"Displayed, Hit Return for No"
5910 PRINT"Change, New Name for Change"
5920 FORI=1TO14
5930 X$="":PRINTNA$(SA(I));:INPUTX$
5940 IFX$=""THEN5970
```

```
5950 GOSUB6560:IFFO=-1THEN5930
5960 SA(I)=FO
5970 IFSA(I)<>OTHENNS=I
5980 NEXTI:RETURN
5990 PRINT
6000 PRINT"Type Signal Name or Return : ";
6010 X$="":INPUTX$
6020 IFX$=""THENRETURN
6030 GOSUB6560:IFFO=-1THEN6000
6040 OV=NE(FO)
6050 PRINTNA$(FO);"=";LP$(NE(FO));" --> ";
6060 X$="":INPUTX$
6070 IFX$="."THENNE(FO)=0:GOTO6120
6080 IFX$="1"THENNE(FO)=3:GOTO6120
6090 IFX$="?"THENNE(FO)=1:GOTO6120
6100 IFX$=""THEN6000
6110 GOTO6050
6120 I=FO:IFNE(I)<>OL(I)THENGOSUB4920
6130 IFOV=NE(FO)THEN6000
6140 IFPRTHENPRT+
6150 PRINTNA$(FO);TAB(TN);"(";LP$(OV);
6160 PRINT") --> (";LP$(NE(FO));")"
6170 IFPRTHENPRT-
6180 GOTO6000
6190 REM ****************************
6200 REM *   Read and Parse Line    *
6210 REM *        [ 10000 ]         *
6220 REM ****************************
6230 IFIE=1THENL$="END":GOTO6250
6240 READL$
6250 IFLEFT$(L$,1)="%"THENIE=1:L$="END"
6260 I=1:I3=1:LI=0:I2=LEN(L$)
6270 I=I+1:IFI>I2THEN6290
6280 IFMID$(L$,I,1)<>" "THEN6270
6290 LI$(LI)=MID$(L$,I3,I-I3)
6300 LI=LI+1
6310 IFLI>10THENRETURN
6320 I=I+1:IFI>I2THEN6350
6330 IFMID$(L$,I,1)=" "THEN6320
6340 I3=I:GOTO6270
6350 FORI=LITO10
6360 LI$(I)=" ":NEXTI
6370 RETURN
6380 REM ********************
6390 REM *   Look Up Type   *
6400 REM *     [ 11000 ]    *
6410 REM ********************
6420 FO=-1
6430 FORII=OTONTYPE-1
6440 IFX$=TYPE$(II)THENFO=II:TP=TYPE(II)
6450 NEXTII
6460 RETURN
6470 REM *****************
6480 REM *   ADD TYPE    *
```

```
6490 REM *   [ 11100 ]  *
6500 REM ****************
6510 IFFO=-1THENFO=NTYP:NTYP=NTYP+1
6520 IFFO>NTYPTHENNTYP=FO
6530 TYPE$(FO)=X$
6540 TYPE(FO)=TPTR
6550 RETURN
6560 REM ****************************
6570 REM *   LOOK UP SIGNAL NAME    *
6580 REM *         [ 11500 ]        *
6590 REM ****************************
6600 FO=-1:IFX$=" "THENX$="0"
6610 IFLEFT$(X$,1)="#"THEN6660
6620 FORII=0TONNAM
6630 IFX$=NAM$(II)THENFO=II
6640 NEXTII
6650 RETURN
6660 REM ***************************************
6670 REM *  "NAME" IS A NUMBER STORE AS NEG   *
6680 REM *              [ 11560 ]             *
6690 REM ***************************************
6700 XX$=RIGHT$(X$,LEN(X$)-1)
6710 FO=-VAL(XX$):IFFO=-1THENFO=0
6720 RETURN
6730 REM ************************
6740 REM *   Add Name to List   *
6750 REM *       [ 11600 ]      *
6760 REM ************************
6770 IFFO=-1THENFO=NNAM:NNAM=NNAM+1
6780 IFFO>NNAMTHENNNAM=FO+1
6790 NAM$(FO)=X$
6800 RETURN
6810 REM ************************
6820 REM *   Clear Old Array    *
6830 REM *      [ 12000 ]       *
6840 REM ************************
6850 FORI=0TOMN:OL(I)=0:NEXTI
6860 TL=0:RETURN
6870 REM ***************************************
6880 REM *   Mark and Check for Duplicates     *
6890 REM *              [ 12100 ]              *
6900 REM ***************************************
6910 IFX=0THENRETURN
6920 IFOL(X)<>0THENGOSUB6940
6930 OL(X)=1:RETURN
6940 PRINT"Error Multiple Defn Output ";NA$(X)
6950 GOSUB6960:RETURN
6960 PRINT"Enter I(gnore) To Ignore Error,"
6970 PRINT"Any Other to Prevent Simulation";
6980 S1$="I":GOSUB1200
6990 IFQ$<>"I"THENER=1
7000 RETURN
7010 REM ***********************************
7020 REM *   Check for Multiple Outputs     *
```

```
7030 REM *              [ 12200 ]            *
7040 REM ************************************
7050 X1=ST
7060 FORI=X1TONL-1
7070 FORJ=6TO7
7080 X=LO(I,J):GOSUB6870
7090 NEXTJ:NEXTI:RETURN
7100 REM ************************************
7110 REM *   Check for Defined Inputs    *
7120 REM *            [ 12300 ]           *
7130 REM *************************************
7140 X1=ST
7150 FORI=X1TONL-1
7160 FORJ=1TO5
7170 X=LO(I,J):GOSUB7200
7180 IFX<OTHENNM=NM-X
7190 NEXTJ:NEXTI:RETURN
7200 REM ***********************
7210 REM *    Check For Mark    *
7220 REM *       [ 12400 ]      *
7230 REM ***********************
7240 IFX<=OTHENRETURN
7250 IFOL(X)=OTHENGOSUB7270
7260 OL(X)=OL(X)+1:TL=TL+1:RETURN
7270 PRINT"Error Undefined Input ";NA$(X)
7280 GOSUB6960:RETURN
7290 ER=1:RETURN
7300 REM ************************************
7310 REM *   Add T(0)-T(7) To Logic Array    *
7320 REM *                 { 20000 }         *
7330 REM ************************************
7340 FORII=0TO7
7350 LO(NL,II)=T(II)
7360 NEXTII
7370 PRINTTY$(LO(NL,0));" ";
7380 IPTY$(LO(NL,0))="END"THEN7420
7390 FORII=1TO7:XX=LO(NL,II):PRINTTAB(TN*II)
7400 IFXX<OTHENPRINT"$";-XX;" ";:GOTO7420
7410 PRINTNA$(XX);" ";:NEXTII
7420 PRINT:NL=NL+1:RETURN
7430 REM ************************************
7440 REM *   Add T(0)-T(9) To Extern Array    *
7450 REM *                 [ 21000 ]         *
7460 REM ************************************
7470 FORII=0TO9
7480 EX(NE,II)=T(II)
7490 NEXTII
7500 Q$=STR$(NE+1)
7510 PRINT"(";RIGHT$(Q$,LEN(Q$)-1);")";
7520 PRINTTAB(TN);TEXS(EX(NE,0));" ";
7530 PRINTTAB(2*TN);NA$(EX(NE,1));" ";
7540 PRINTTAB(3*TN);LV$(EX(NE,2));" ";
7550 FORII=3TO9
7560 PRINTTAB((II+4)*(TN/2));STR$(EX(NE,II));
```

```
7570 NEXTII:PRINT
7580 NE=NE+1:RETURN
7590 REM ************************************
7600 REM *   Build Macro Correspondnce Array *
7610 REM *              [ 30000 ]            *
7620 REM ************************************
7630 LO=I:X=LO(I,0):J=TYPE(X):Y=LO(J,0)
7640 M1=J:OL(0)=0:PRINT
7650 PRINT"*** Expanding Macro: ";TY$(X);" ***"
7660 PRINT:FORK=1TOMNAM:OL(K)=-1:NEXTK
7670 IFX<>YTHEN7920
7680 FORK=1TO5
7690 I1=LO(M1,K):I2=LO(LO,K)
7700 IFI1=0ANDI2<>0THEN7720
7710 OL(I1)=I2
7720 NEXTK
7730 FORK=6TO7
7740 I1=LO(M1,K):I2=LO(LO,K)
7750 IFI1<>0ANDI2=0THEN7770
7760 OL(I1)=I2
7770 NEXTK
7780 GOSUB7820
7790 M1=M1+1:Y=LO(M1,0):X=LO(LO,0)
7800 IFTY$(Y)="A"THEN7670
7810 RETURN
7820 REM ****************************
7830 REM *   Move El Array Up One   *
7840 REM *          [ 30500 ]       *
7850 REM ****************************
7860 NL=NL-1
7870 FORK=LOTONL-1
7880 FORK1=0TO8
7890 LO(K,K1)=LO(K+1,K1)
7900 NEXTK1:NEXTK
7910 RETURN
7920 REM ********************************
7930 REM *   Mismatch on Macro Call     *
7940 REM *          [ 30900 ]           *
7950 REM ********************************
7960 REM Should Not Happen But Never Know
7970 PRINT" Mismatch on Macro Call, Cant cont"
7980 IFPROMPTHENPRT+:PRINTCHR$(12):START-
7990 END
8000 REM ************************************
8010 REM *   Add Expanded Macro to Elm Ary   *
8020 REM *              [ 31000 ]           *
8030 REM ************************************
8040 NL=NL-1
8050 T(0)=LO(M1,0)
8060 FORK=1TO7
8070 X=LO(M1,K)
8080 IFX<OTHENT(K)=X:GOTO8110
8090 IFOL(X)=-1THENGOSUB8150
8100 T(K)=OL(X)
```

```
7570 NEXT I1:PRINT
7580 NE=NE+1:RETURN
7590 REM ***************************************
7600 REM *   Build Macro Correspondnce Array *
7610 REM *              [ 30000 ]              *
7620 REM ***************************************
7630 LO=I:X=LO(I,0):J=TYPE(X):Y=LO(J,0)
7640 M1=1:OL(0)=0:PRINT
7650 PRINT"*** Expanding Macro: ";TY$(X),"
7660 PRINT:FORK=1TOMNAM:OL(K)=-1:NEXT K
7670 IFX<>YTHEN7920
7680 FORK=1TO5
7690 I1=LO(M1,K):I2=LO(LO,K)
7700 IFI1=0ANDI2<>0THEN7720
7710 OL(I1)=I2
7720 NEXTK
7730 FORK=6TO7
7740 I1=LO(M1,K):I2=LO(LO,K)
7750 IFI1<>0ANDI2=0THEN7770
7760 OL(I1)=I2
7770 NEXTK
7780 GOSUB7820
7790 M1=M1+1:Y=LO(M1,0):X=LO(LO,0)
7800 IFTY$(Y)="*"THEN7670
7810 RETURN
7820 REM *****************************
7830 REM *   Move El Array Up One    *
7840 REM *          [ 30500 ]        *
7850 REM *****************************
7860 NL=NL-1
7870 FORK=LOTONL-1
7880 FORK1=0TO8
7890 LO(K,K1)=LO(K+1,K1)
7900 NEXTK1:NEXTK
7910 RETURN
7920 REM *******************************
7930 REM *   Mismatch on Macro Call    *
7940 REM *          [ 30900 ]          *
7950 REM *******************************
7960 REM Should Not Happen But Never Know
7970 PRINT" Mismatch on Macro Call, Cant cont"
7980 IFPRORHPTHENPRT+:PRINTCHR$(12):START
7990 END
8000 REM ***************************************
8010 REM *   Add Expanded Macro to Elm Ary     *
8020 REM *              [ 31000 ]              *
8030 REM ***************************************
8040 NL=NL-1
8050 T(0)=LO(M1,0)
8060 FORK=1TO7
8070 X=LO(M1,K)
8080 IFX<0THENT(K)=X:GOTO8110
8090 IFOL(X)=-1THENGOSUB8150
8100 T(K)=OL(X)
```

```
8110 NEXT K
8120 GOSUB7300
8130 M1=M1+1:IFT(0)<>0THEN8050
8140 RETURN
8150 REM *********************************
8160 REM *   Add Unique Name, Number   *
8170 REM *          [ 31500 ]          *
8180 REM *********************************
8190 OL(X)=NNAM:X$=STR$(NNAM)
8200 NAM$(NNAM)="&"+RIGHT$(X$,LEN(X$)-1)
8210 NNAM=NNAM+1
8220 RETURN
8230 REM ***********************
8240 REM *   Simulate Gates    *
8250 REM *      [ 40000 ]      *
8260 REM ***********************
8270 IPC9=0THENRETURN
8280 L=SL
8290 IFSTF<>0THENIFS0(L)=0THEN8310
8300 GOSUB8330
8310 L=L+1:IFL<=NLTHEN8290
8320 RETURN
8330 REM *******************************
8340 REM *   Simulate this Gate (L)    *
8350 REM *          [ 40100 ]          *
8360 REM *******************************
8370 LT=LO(L,0):FORI=1TO5:IN(I)=LO(L,I):NEXTI
8380 Q0=LO(L,6):Q1=LO(L,7)
8390 ONLTGOSUB8550,8540,8590,8580,8630,8620,8660,8760,8860,9000,9130
8400 REM *** AND NAND OR NOR XOR XNOR ***
8410 REM *** DFF JKFF CNTR PLSR SREG  ***
8420 GOSUB8440
8430 RETURN
8440 REM ***********************
8450 REM *   Store Q and Q-    *
8460 REM *      [ 40200 ]      *
8470 REM ***********************
8480 NE(Q0)=Y:NE(Q1)=LN(Y):RETURN
8490 REM ***********************
8500 REM *   Get Q into XQ     *
8510 REM *      [ 40300 ]      *
8520 REM ***********************
8530 OL(0)=LN(OL(Q1)):XQ=OL(Q0):RETURN
8540 X=Q0:Q0=Q1:Q1=X:REM *** NAND GATE ***
8550 OL(0)=L1:Y=L1:REM *** AND  GATE ***
8560 FORI=1TO5:Y=YANDOL(IN(I)):NEXTI
8570 RETURN
8580 X=Q0:Q0=Q1:Q1=X:REM *** NOR  GATE ***
8590 OL(0)=L0:Y=L0:REM *** OR   GATE ***
8600 FORI=1TO5:Y=YOROL(IN(I)):NEXTI
8610 RETURN
8620 X=Q0:Q0=Q1:Q1=X:REM *** XNOR GATE ***
8630 OL(0)=LX:REM *** XOR  GATE ***
8640 Y0=OL(IN(1)):Y1=LN(Y0):YC=OL(IN(2))
```

```
8650 GOSUB9260:RETURN
8660 REM ********************
8670 REM *   D Flip Flop    *
8680 REM *    [ 41500 ]     *
8690 REM ********************
8700 GOSUB8490:OL(0)=LX:XD=OL(IN(1))
8710 XC=OL(IN(3))
8720 XL=LO(L,8):OL(0)=L1:XR=OL(IN(4))
8730 XS=OL(IN(5))
8740 GOSUB9320:LO(L,8)=YL
8750 RETURN
8760 REM ***********************
8770 REM *   J-K Flip Flop    *
8780 REM *     [ 41600 ]      *
8790 REM ***********************
8800 REM Set up as multiplexed DFF, Q ctls mux
8810 GOSUB8490:XM=XQ:OL(0)=LX:XC=OL(IN(3))
8820 DD=OL(IN(1)):D1=LN(OL(IN(2)))
8830 OL(0)=L1:XR=OL(IN(4)):XS=OL(IN(5))
8840 XL=LO(L,8)
8850 GOSUB9450:LO(L,8)=YL:RETURN
8860 REM ******************************
8870 REM *   Parallel Load Counter    *
8880 REM *        [ 41700 ]           *
8890 REM ******************************
8900 OL(0)=LX:YE=OL(IN(2)):XC=OL(IN(3))
8910 OL(0)=LO:XM=OL(IN(4))
8920 OL(0)=L1:XR=OL(IN(5)):XS=L1
8930 GOSUB8490:XL=LO(L,8)
8940 YC=YE:Y0=XQ:Y1=LN(Y0)
8950 GOSUB9260:D0=Y:D1=OL(IN(1)):GOSUB9450
8960 LO(L,8)=YL:IFTY$(LO(L+1,0))<>"*"THENRETURN
8970 GOSUB8440:L=L+1:IN(1)=LO(L,1)
8980 Q0=LO(L,6):Q1=LO(L,7)
8990 YE=YEANDY:GOTO8930
9000 REM ************************************
9010 REM *   Parallel Load Shift Register   *
9020 REM *            [ 41800 ]             *
9030 REM ************************************
9040 OL(0)=LX:D0=OL(IN(2)):XC=OL(IN(3))
9050 OL(0)=LO:XM=OL(IN(4))
9060 OL(0)=L1:XR=OL(IN(5)):XS=L1
9070 OL(0)=LX:D1=OL(IN(1)):GOSUB8490
9080 XL=LO(L,8):GOSUB9450
9090 LO(L,8)=YL:IFTY$(LO(L+1,0))<>"*"THENRETURN
9100 GOSUB8440:L=L+1:IN(1)=LO(L,1):Q0=LO(L,6)
9110 Q1=LO(L,7):D0=Y
9120 GOTO9070
9130 REM ******************************
9140 REM *   N Bit Shift Register   *
9150 REM *       [ 41900 ]          *
9160 REM ******************************
9170 IFIN(2)=0THEN8660
9180 OL(0)=LX:XD=OL(IN(1)):XC=OL(IN(3))
```

```
9190 OL(0)=L1:XR=OL(IN(4)):XS=OL(IN(5))
9200 M1=LO(1,8)
9210 FOR I=1TO-IN(2)
9220 XL=ME(M1,0):XQ=ME(M1,1)
9230 GOSUB9320:XD=XQ:ME(M1,0)=YL:ME(M1,1)=Y
9240 M1=M1+1:NEXT I
9250 RETURN
9260 REM ********************
9270 REM *   MUX Function   *
9280 REM *     [ 45000 ]    *
9290 REM ********************
9300 Y=(YOANDLN(YC))OR(Y1ANDYC)OR(YOANDY1)
9310 RETURN
9320 REM ********************************
9330 REM *   Shift Register Function    *
9340 REM *          [ 45100 ]           *
9350 REM ********************************
9360 YO=XD:YC=XC:Y1=XL
9370 GOSUB9260:GOSUB9400
9380 YL=Y:YO=XQ:GOSUB9260:GOSUB9400
9390 RETURN
9400 REM ********************************
9410 REM *   Set and Reset function     *
9420 REM *          [ 45200 ]           *
9430 REM ********************************
9440 Y=(YORIN(XS))ANDXR:RETURN
9450 REM ********************
9460 REM *   MUX and DFF    *
9470 REM *     [ 45300 ]    *
9480 REM ********************
9490 YO=DO:Y1=D1:YC=XM:GOSUB9260
9500 XD=Y:GOSUB9320
9510 RETURN
9520 REM
9530 REM ------------------------------------
9540 REM              INPUT
9550 REM           [ 50000 ]
9560 REM ------------------------------------
9570 REM
9580 DATA MACRO NJKFF
9590 DATA NJKFF J K CK- PRESET CLEAR Q Q-
9600 DATA JKFF J K CLOCK CLEAR PRESET Q Q-
9610 DATA NAND CK- 0 0 0 0 CLOCK
9620 DATA END
9630 DATA MACRO SHIFT
9640 DATA SHIFT CK J K- CLR- SH/LD QA
9650 DATA * 0 0 0 0 0 QB
9660 DATA * 0 0 0 0 0 QC
9670 DATA * 0 0 0 0 0 QD
9680 DATA FLSR 0 DSER CK LOAD CLR- QA
9690 DATA * 0 0 0 0 0 QB
9700 DATA * 0 0 0 0 0 QC
9710 DATA * 0 0 0 0 0 QD
9720 DATA AND J K- 0 0 0 DSER
```

```
9730 DATA NAND SH/LD 0 0 0 0 LOAD
9740 DATA END
9750 DATA MACRO MUX
9760 DATA MUX 1A 1B G SEL 0 1Y
9770 DATA * 2A 2B 0 0 0 2Y
9780 DATA * 3A 3B 0 0 0 3Y
9790 DATA * 4A 4B 0 0 0 4Y
9800 DATA AND 1A SG 0 0 0 T1
9810 DATA AND 1B SS 0 0 0 T2
9820 DATA AND 2A SG 0 0 0 T3
9830 DATA AND 2B SS 0 0 0 T4
9840 DATA AND 3A SG 0 0 0 T5
9850 DATA AND 3B SS 0 0 0 T6
9860 DATA AND 4A SG 0 0 0 T7
9870 DATA AND 4B SS 0 0 0 T8
9880 DATA OR T1 T2 0 0 0 1Y
9890 DATA OR T3 T4 0 0 0 2Y
9900 DATA OR T5 T6 0 0 0 3Y
9910 DATA OR T7 T8 0 0 0 4Y
9920 DATA NOR G SEL 0 0 0 SG
9930 DATA NAND SEL 0 0 0 0 SEL-
9940 DATA NOR G SEL- 0 0 0 SS
9950 DATA END
9960 DATA ELEMENTS
9970 DATA NAND EXINT 0 0 0 0 EXINT-
9980 DATA AND EXINT ENEG 0 0 0 T01
9990 DATA AND EXINT EPOS 0 0 0 T02
10000 DATA NAND CLK1 0 0 0 0 T03
10010 DATA NOR T02 T02 0 0 0 T04
10020 DATA NOR T03 T03 0 0 0 T05
10030 DATA OR T05 T05 0 0 0 T06
10040 DATA AND T01 T06 0 0 0 T07
10050 DATA OR T06 T04 0 0 0 T08
10060 DATA AND T03 T07 0 0 0 T09
10070 DATA NOR T08 T03 0 0 0 T10
10080 DATA NOR T09 T10 0 0 0 STB
10090 DATA NJKFF HI SPEED STB SPEED HI MXCTL MXCTL-
10100 DATA NJKFF MXCTL LO STB HI T16 0 T11
10110 DATA NJKFF HI LO QC1 HI T16 0 T12
10120 DATA NJKFF MXCTL- LO STB HI T18 0 T13
10130 DATA NJKFF HI LO QC2 HI T18 0 T14
10140 DATA NOR T11 QA1 0 0 0 T15
10150 DATA NOR EXINT- QA1 0 0 0 T16
10160 DATA NOR T13 QA2 0 0 0 T17
10170 DATA NOR EXINT- QA2 0 0 0 T18
10180 DATA SHIFT OSC T15 T15 EXINT HI QA1
10190 DATA * 0 0 0 0 0 QB1
10200 DATA * 0 0 0 0 0 QC1
10210 DATA * 0 0 0 0 0 QD1
10220 DATA SHIFT OSC T17 T17 EXINT HI QA2
10230 DATA * 0 0 0 0 0 QB2
10240 DATA * 0 0 0 0 0 QC2
10250 DATA * 0 0 0 0 0 QD2
10260 DATA OR QA1 QA1 0 0 0 FI1
```

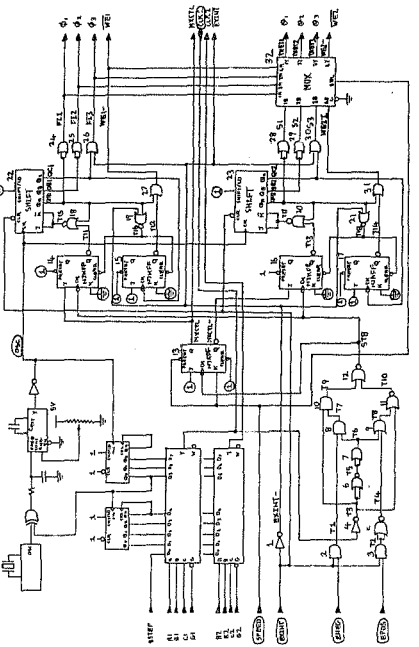
```
10270 DATA OR QB1 QB1 0 0 0 FI2
10280 DATA OR QC1 EXINT- 0 0 0 FI3
10290 DATA OR QA1 T12 0 0 0 WE1-
10300 DATA OR QA2 QA2 0 0 0 S1
10310 DATA OR QB2 QB2 0 0 0 S2
10320 DATA OR QC2 EXINT- 0 0 0 S3
10330 DATA OR QA2 T14 0 0 0 WE2I
10340 DATA MUX FI1 S1 LO SPEED 0 THET1
10350 DATA * FI2 S2 0 0 0 THET2
10360 DATA * FI3 S3 0 0 0 THET3
10370 DATA * WE1- WE2I 0 0 0 WE2-
10380 DATA END
10390 DATA EXTERNALS
10400 DATA A HI 1 0
10410 DATA A LO 0 0
10420 DATA P OSC 0 0 10
10430 DATA P CLK1 0 0 50
10440 DATA A SPEED 0 0 10
10450 DATA A EXINT 0 0 10
10460 DATA A ENEG 1 0
10470 DATA A EPOS 1 0
10480 DATA END
10490 DATA OUTPUTS
10500 DATA SPEED EXINT OSC CLK1 STB MXCTL FI1
10510 DATA FI2 FI3 WE1- THET1 THET2 THET3 WE2-
10520 DATA END
10530 DATA ZZZ
```

APPENDIX K

LOGIC-SIMULATOR TEST CIRCUIT

INTRODUCTION

This appendix contains the Clock Generation circuitry used in the Interface Control Board of the Test-Hardware. This circuitry is simulated using the Logic-Simulator. The signal names are marked on the circuit diagram. The next appendix actually contains the output from the simulator.



CLOCK GENERATION

LOGIC SIMULATOR TEST CIRCUIT

MULTIBUS INTERFACE AND CONTROL BOARD

APPENDIX L

LOGIC-SIMULATOR OUTPUT

INTRODUCTION

This is the actual output of the simulator. The circuit simulated is the Clock Generation Circuitry of the Interface Control Card and the output can actually be compared with the timing diagrams given in Chapter 7.

Digital Simulator Output

LOGIC SIMULATOR

```
200 NAMES
200 LOGIC DEVICES
50 TYPES
30 EXTERNAL STIMULI


*** Initialising Arrays   ***
*** -------------------   ***

*** Reading Any Macros    ***
*** -------------------   ***

*** Reading in Macro : NJKFF ***

NJKFF    J       K       CK-     PRESET  CLEAR   Q       Q-
JKFF     J       K       CLOCK   CLEAR   PRESET  Q       Q-
NAND     CK-     0       0       0       0       CLOCK   0
END

*** Checking Macro NJKFF for Errors ***

*** Reading in Macro : SHIFT ***

SHIFT    CK      J       K-      CLR-    SH/LD   QA      0
*        0       0       0       0       0       QB      0
*        0       0       0       0       0       QC      0
*        0       0       0       0       0       QD      0
PLSR     0       DSER    CK      LOAD    CLR-    QA      0
*        0       0       0       0       0       QB      0
*        0       0       0       0       0       QC      0
*        0       0       0       0       0       QD      0
AND      J       K-      0       0       0       DSER    0
NAND     SH/LD   0       0       0       0       LOAD    0
END

*** Checking Macro SHIFT for Errors ***

*** Reading in Macro : MUX ***

MUX      1A      1B      G       SEL     0       1Y      0
*        2A      2B      0       0       0       2Y      0
*        3A      3B      0       0       0       3Y      0
*        4A      4B      0       0       0       4Y      0
AND      1A      SG      0       0       0       T1      0
AND      1B      SS      0       0       0       T2      0
AND      2A      SG      0       0       0       T3      0
AND      2B      SS      0       0       0       T4      0
AND      3A      SG      0       0       0       T5      0
AND      3B      SS      0       0       0       T6      0
AND      4A      SG      0       0       0       T7      0
AND      4B      SS      0       0       0       T8      0
OR       T1      T2      0       0       0       1Y      0
OR       T3      T4      0       0       0       2Y      0
OR       T5      T6      0       0       0       3Y      0
OR       T7      T8      0       0       0       4Y      0
NOR      G       SEL     0       0       0       SG      0
NAND     SEL     0       0       0       0       SEL-    0
```

```
NOR     G       SEL-    0       0       0       SS      0
END

*** Checking Macro MUX for Errors ***

*** Read In Logic Network ***
*** ---------------------- ***

NAND    EXINT   0       0       0       0       EXINT-  0
AND     EXINT   ENEG    0       0       0       T01     0
AND     EXINT   EPDS    0       0       0       T02     0
NAND    CLK1    0       0       0       0       T03     0
NOR     T02     T02     0       0       0       T04     0
NOR     T03     T03     0       0       0       T05     0
OR      T05     T05     0       0       0       T06     0
AND     T01     T06     0       0       0       T07     0
OR      T06     T04     0       0       0       T08     0
AND     T03     T07     0       0       0       T09     0
NOR     T08     T02     0       0       0       T10     0
NOR     T09     T10     0       0       0       STB     0
NJKFF   HI      SPEED   STB     SPEED   HI      MXCTL   MXCTL-
NJKFF   MXCTL   LO      STB     HI      T16     0       T11
NJKFF   HI      LO      QC1     HI      T16     0       T12
NJKFF   MXCTL-  LO      STB     HI      T18     0       T13
NJKFF   HI      LO      QC2     HI      T18     0       T14
NOR     T11     QA1     0       0       0       T15     0
NOR     EXINT-  QA1     0       0       0       T16     0
NOR     T13     QA2     0       0       0       T17     0
NOR     EXINT-  QA2     0       0       0       T18     0
SHIFT   OSC     T15     T15     EXINT   HI      QA1     0
*       0       0       0       0       0       QB1     0
*       0       0       0       0       0       QC1     0
*       0       0       0       0       0       QD1     0
SHIFT   OSC     T17     T17     EXINT   HI      QA2     0
*       0       0       0       0       0       QB2     0
*       0       0       0       0       0       QC2     0
*       0       0       0       0       0       QD2     0
OR      QA1     QA1     0       0       0       FI1     0
OR      QB1     QB1     0       0       0       FI2     0
OR      QC1     EXINT-  0       0       0       FI3     0
OR      QA1     T12     0       0       0       WE1-    0
OR      QA2     QA2     0       0       0       S1      0
OR      QB2     QB2     0       0       0       S2      0
OR      QC2     EXINT-  0       0       0       S3      0
OR      QA2     T14     0       0       0       WE2I    0
MUX     FI1     S1      LO      SPEED   0       THET1   0
*       FI2     S2      0       0       0       THET2   0
*       FI3     S3      0       0       0       THET3   0
*       WE1-    WE2I    0       0       0       WE2-    0
END

*** Expanding Any Macros  ***
*** ---------------------  ***

*** Expanding Macro: NJKFF ***

JKFF    HI      SPEED   &51     HI      SPEED   MXCTL   MXCTL-
NAND    STB     0       0       0       0       &51     0
END

*** Expanding Macro: NJKFF ***
```

Digital Simulator Output

```
JKFF    MXCTL   LO      &52     T16     HI      &53     T11
NAND    STB     0       0       0       0       &52     0
END

*** Expanding Macro: NJKFF ***

JKFF    HI      LO      &54     T16     HI      &55     T12
NAND    QC1     0       0       0       0       &54     0
END

*** Expanding Macro: NJKFF ***

JKFF    MXCTL-  LO      &56     T18     HI      &57     T13
NAND    STB     0       0       0       0       &56     0
END

*** Expanding Macro: NJKFF ***

JKFF    HI      LO      &58     T18     HI      &59     T14
NAND    QC2     0       0       0       0       &58     0
END

*** Expanding Macro: SHIFT ***

PLSR    0       &60     OSC     &61     EXINT   QA1     0
*       0       0       0       0       0       QB1     0
*       0       0       0       0       0       QC1     0
*       0       0       0       0       0       QD1     0
AND     T15     T15     0       0       0       &60     0
NAND    HI      0       0       0       0       &61     0
END

*** Expanding Macro: SHIFT ***

PLSR    0       &62     OSC     &63     EXINT   QA2     0
*       0       0       0       0       0       QB2     0
*       0       0       0       0       0       QC2     0
*       0       0       0       0       0       QD2     0
AND     T17     T17     0       0       0       &62     0
NAND    HI      0       0       0       0       &63     0
END

*** Expanding Macro: MUX ***

AND     F11     &64     0       0       0       &65     0
AND     S1      &66     0       0       0       &67     0
AND     F12     &64     0       0       0       &68     0
AND     S2      &66     0       0       0       &69     0
AND     F13     &64     0       0       0       &70     0
AND     S3      &66     0       0       0       &71     0
AND     WE1-    &64     0       0       0       &72     0
AND     WE2I    &66     0       0       0       &73     0
OR      &65     &67     0       0       0       THET1   0
OR      &68     &69     0       0       0       THET2   0
OR      &70     &71     0       0       0       THET3   0
OR      &72     &73     0       0       0       WE2-    0
NOR     LO      SPEED   0       0       0       &64     0
NAND    SPEED   0       0       0       0       &74     0
NOR     LO      &74     0       0       0       &66     0
END
```

Digital Simulator Output

*** Expanded Logic Description ***

No.	Type	Inputs 1	2	3	4	5	Outputs Q	Q-
(1)	NAND	EXINT	.	.	.	.	EXINT-	.
(2)	AND	EXINT	ENEG	.	.	.	T01	.
(3)	AND	EXINT	EPOS	.	.	.	T02	.
(4)	NAND	CLK1	.	.	.	.	T03	.
(5)	NOR	T02	T02	.	.	.	T04	.
(6)	NOR	T03	T03	.	.	.	T05	.
(7)	OR	T05	T05	.	.	.	T06	.
(8)	AND	T01	T06	.	.	.	T07	.
(9)	OR	T06	T04	.	.	.	T08	.
(10)	AND	T03	T07	.	.	.	T09	.
(11)	NOR	T08	T02	.	.	.	T10	.
(12)	NOR	T09	T10	.	.	.	STB	.
(13)	NOR	T11	QA1	.	.	.	T15	.
(14)	NOR	EXINT-	QA1	.	.	.	T16	.
(15)	NOR	T13	QA2	.	.	.	T17	.
(16)	NOR	EXINT-	QA2	.	.	.	T18	.
(17)	OR	QA1	QA1	.	.	.	FI1	.
(18)	OR	QB1	QB1	.	.	.	FI2	.
(19)	OR	QC1	EXINT-	.	.	.	FI3	.
(20)	OR	QA1	T12	.	.	.	WE1-	.
(21)	OR	QA2	QA2	.	.	.	S1	.
(22)	OR	QB2	QB2	.	.	.	S2	.
(23)	OR	QC2	EXINT-	.	.	.	S3	.
(24)	OR	QA2	T14	.	.	.	WE2I	.
(25)	JKFF	HI	SPEED	&51	HI	SPEED	MXCTL	MXCTL-
(26)	NAND	STB	.	.	.	.	&51	.
(27)	JKFF	MXCTL	LO	&52	T16	HI	&53	T11
(28)	NAND	STB	.	.	.	.	&52	.
(29)	JKFF	HI	LO	&54	T16	HI	&55	T12
(30)	NAND	QC1	.	.	.	.	&54	.
(31)	JKFF	MXCTL-	LO	&56	T18	HI	&57	T13
(32)	NAND	STB	.	.	.	.	&56	.
(33)	JKFF	HI	LO	&58	T18	HI	&59	T14
(34)	NAND	QC2	.	.	.	.	&58	.
(35)	PLSR	.	&60	OSC	&61	EXINT	QA1	.
(36)	*	.	.	.	.	.	QB1	.
(37)	*	.	.	.	.	.	QC1	.
(38)	*	.	.	.	.	.	QD1	.
(39)	AND	T15	T15	.	.	.	&60	.
(40)	NAND	HI	.	.	.	.	&61	.
(41)	PLSR	.	&62	OSC	&63	EXINT	QA2	.
(42)	*	.	.	.	.	.	QB2	.
(43)	*	.	.	.	.	.	QC2	.
(44)	*	.	.	.	.	.	QD2	.
(45)	AND	T17	T17	.	.	.	&62	.
(46)	NAND	HI	.	.	.	.	&63	.
(47)	AND	FI1	&64	.	.	.	&65	.
(48)	AND	S1	&66	.	.	.	&67	.
(49)	AND	FI2	&64	.	.	.	&68	.
(50)	AND	S2	&66	.	.	.	&69	.
(51)	AND	FI3	&64	.	.	.	&70	.
(52)	AND	S3	&66	.	.	.	&71	.
(53)	AND	WE1-	&64	.	.	.	&72	.
(54)	AND	WE2I	&66	.	.	.	&73	.
(55)	OR	&65	&67	.	.	.	THET1	.
(56)	OR	&68	&69	.	.	.	THET2	.

```
(57)    OR      &70     &71     .       .       .       THET3   .
(58)    OR      &72     &73     .       .       .       WE2-    .
(59)    NOR     LO      SPEED   .       .       .       &64     .
(60)    NAND    SPEED   .       .       .       .       &74     .
(61)    NOR     LO      &74     .       .       .       &66     .
        END     .       .       .       .       .       .       .

*** Read External Stimuli ***
*** ---------------------- ***

No.     Type    Name    Val  Ben Os1 Os2 Os3 Os4 Os5 Os6
+-------+-------+-------+----+---+---+---+---+---+---+---+
(1)     A       HI      1    0   0   0   0   0   0   0
(2)     A       LO      0    0   0   0   0   0   0   0
(3)     P       OSC     0    0   10  0   0   0   0   0
(4)     P       CLK1    0    0   100 0   0   0   0   0
(5)     A       SPEED   0    0   10  0   0   0   0   0
(6)     A       EXINT   0    0   10  0   0   0   0   0
(7)     A       ENEG    1    0   0   0   0   0   0   0
(8)     A       EPOS    1    0   0   0   0   0   0   0
        END

*** Checking for Errors  ***

*** Read Nodes To Sample  ***
*** --------------------  ***

SPEED   EXINT   OSC     CLK1    STB     MXCTL   FI1
FI2     FI3     WE1-    THET1   THET2   THET3   WE2-
END

*** Setting Up Simulation ***
*** ---------------------- ***

*** Build Selective Trace Table ***
*** ---------------------------- ***


*** Load Table ***
*** ---------- ***

NODE    DEFINED         DRIVEN ELEMENTS
----    -------         ---------------
CLK1    [ EXT   4 ]     4
ENEG    [ EXT   7 ]     2
EPOS    [ EXT   8 ]     3
EXINT   [ EXT   6 ]     1  2  3  35  41
EXINT-  [ Q     1 ]     14  16  19  23
FI1     [ Q    17 ]     47
FI2     [ Q    18 ]     49
FI3     [ Q    19 ]     51
HI      [ EXT   1 ]     25  25  27  29  29  31  33  33  40  46
LO      [ EXT   2 ]     27  29  31  33  59  61
MXCTL   [ Q    25 ]     27
MXCTL-  [ Q-   25 ]     31
OSC     [ EXT   3 ]     35  41
QA1     [ Q    35 ]     13  14  17  17  20
QA2     [ Q    41 ]     15  16  21  21  24
QB1     [ Q    36 ]     18  18
QB2     [ Q    42 ]     22  22
QC1     [ Q    37 ]     19  30
```

Digital Simulator Output

QC2	[Q 43]	23	34
QD1	[Q 38]		
QD2	[Q 44]		
S1	[Q 21]	48	
S2	[Q 22]	50	
S3	[Q 23]	52	
SPEED	[EXT 5]	25	25 59 60
STB	[Q 12]	26	28 32
T01	[Q 2]	8	
T02	[Q 3]	5	5
T03	[Q 4]	6	6 10 11
T04	[Q 5]	9	
T05	[Q 6]	7	7
T06	[Q 7]	8	9
T07	[Q 8]	10	
T08	[Q 9]	11	
T09	[Q 10]	12	
T10	[Q 11]	12	
T11	[Q- 27]	13	
T12	[Q- 29]	20	
T13	[Q- 31]	15	
T14	[Q- 33]	24	
T15	[Q 13]	39	39
T16	[Q 14]	27	29
T17	[Q 15]	45	45
T18	[Q 16]	31	33
THET1	[Q 55]		
THET2	[Q 56]		
THET3	[Q 57]		
WE1-	[Q 20]	53	
WE2-	[Q 58]		
WE21	[Q 24]	54	
&51	[Q 26]	25	
&52	[Q 28]	27	
&53	[Q 27]		
&54	[Q 30]	29	
&55	[Q 29]		
&56	[Q 32]	31	
&57	[Q 31]		
&58	[Q 34]	33	
&59	[Q 33]		
&6\	[Q 39]	35	
&61	[Q 40]	35	
&62	[Q 45]	41	
&63	[Q 46]	41	
&64	[Q 59]	47	49 51 53
&65	[Q 47]	55	
&66	[Q 61]	48	50 52 54
&67	[Q 48]	55	
&68	[Q 49]	56	
&69	[Q 50]	56	
&70	[Q 51]	57	
&71	[Q 52]	57	
&72	[Q 53]	58	
&73	[Q 54]	58	
&74	[Q 60]	61	

Digital Simulator Output

```
Defau't for Start & Increment ? N
Enter Sample Start Time ? 0
Enter Increment Time    ? 5

Time 170  0    25   50   75   100  125  150
          +----+----+----+----+----+----+----+
SPEED     ..1111111111111111111111111111111111
EXINT     ..1111111111111111111111111111111111
OSC       ..11..11..11..11..11..11..11..11..11..1
CLK1      ,,,,,,,,,,,,,,,,,,,,11111111111111
STB       ?1111111111111111111.1111111111111
MXCTL     ?1111111111111111111.,,,,,,,,,,,,,,
FI1       ?.,,,,,,,,,,,,,,,,,,,,1111.,,,,,,,,
FI2       ?.,,,,,,,,,,,,,,,,,,,,,,,,1111.,,,,
FI3       ?11.,,,,,,,,,,,,,,,,,,,,,,,,,,,,1111
WE1-      ?1111111111111111111111111111111111
THET1     ?.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
THET2     ?.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
THET3     ?11.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
WE2-      ??1111111111111111111111111111111111

Time 345  175  200  225  250  275  300  325
          +----+----+----+----+----+----+----+
SPEED     1111111111111111111111111111111111
EXINT     1111111111111111111111111111111111
OSC       1..11..11..11..11..11..11..11..11..
CLK1      11111.,,,,,,,,,,,,,,,,,,,,1111111111
STB       111111.1111111111111111111.11111111
MXCTL     ......11111111111111111111.,,,,,,,,,
FI1       .,,,,,,,,,,,,,,,,,,,,,,,,,,,1111...
FI2       .,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,111
FI3       .,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
WE1-      .,,,,,,,,,,,,,,,,,,,,,,,,,,,,,1111111
THET1     ........1111.,,,,,,,,,,,,,,,,,,,,,,,
THET2     ............1111.,,,,,,,,,,,,,,,,,,,
THET3     ................1111.,,,,,,,,,,,,,,,
WE2-      1111111111111111111.,,,,,,,,,,,,,,,

Time 520  350  375  400  425  450  475  500
          +----+----+----+----+----+----+----+
SPEED     1111111111111111111111111111111111
EXINT     1111111111111111111111111111111111
OSC       11..11..11..11..11..11..11..11..11.
CLK1      1111111111.,,,,,,,,,,,,,,,,,,,,1111
STB       11111111111.1111111111111111111.111
MXCTL     ...........1111111111111111111....
FI1       .,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,11
FI2       1.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
FI3       .1111.,,,,,,,,,. ,,,,,,,,,,,,,,,,,,
WE1-      11111.,,,,,,,,,,,,,,,,,,,,,,,,,,,11
THET1     ............1111.,,,,,,,,,,,,,,,,,,
THET2     ................1111.,,,,,,,,,,,,,,
THET3     ....................1111.,,,,,,,,,,
WE2-      ............111111111111.,,,,,,,,,,

SPEED   (1) -> (.)
```

Digital Simulator Output

```
Time 695  525  550  575  600  625  650  675
          +----+----+----+----+----+----+----+
SPEED     ....................................
EXINT     iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
OSC       .ii..ii..ii..ii..ii..ii..ii..ii..ii
CLK1      iiiiiiiiiiiiii.....................
STB       iiiiiiiiiiiiiii.iiiiiiiiiiiiiiiiii
MXCTL     iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
FI1       ii..............iiii..............
FI2       ..iiii..............iiii..........
FI3       ......iiii..............iiii......
WE1-      iiiiiiiiii........iiiiiiiiii.....
THET1     ii..............iiii..............
THET2     ..iiii..............iiii..........
THET3     ......iiii..............iiii......
WE2-      iiiiiiiiii.......iiiiiiiiiii.....
```

```
Time 870  700  725  750  775  800  825  850
          +----+----+----+----+----+----+----+
SPEED     ....................................
EXINT     iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
OSC       ..ii..ii..ii..ii..ii..ii..ii..ii..i
CLK1      iiiiiiiiiiiiiiiiii..................
STB       i.iiiiiiiiiiiiiiiii.iiiiiiiiiiiii
MXCTL     iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
FI1       ...iiii..............iiii........
FI2       .......iiii..............iiii....
FI3       ...........iiii..............iiii
WE1-      ...iiiiiiiiiii........iiiiiiiiiii
THET1     ...iiii..............iiii........
THET2     .......iiii..............iiii....
THET3     ...........iiii..............iiii
WE2-      ...iiiiiiiiiii.......iiiiiiiiiii
```

```
Time 1045 875  900  925  950  975  1000 1025
          +----+----+----+----+----+----+----+
SPEED     ....................................
EXINT     iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
OSC       i..ii..ii..ii..ii..ii..ii..ii..ii..
CLK1      .....iiiiiiiiiiiiiiiiii.............
STB       iiiii iiiiiiiiiiiiiiiii.iiiiiiii
MXCTL     iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
FI1       ........iiii..............iiii...
FI2       ............iiii..............iii
FI3       ................iiii..............
WE1-      ..........iiiiiiiiii.....iiiiiii
THET1     ........iiii..............iiii...
THET2     ............iiii..............iii
THET3     ................iiii..............
WE2-      i.......iiiiiiiiiiii.....iiiiiii
```

APPENDIX M

TEST-HARDWARE CONTROL FILES

INTRODUCTION

These files are Control Files used to check the Test-Hardware. The files consist of keystrokes which would be sent to the monitor program residing on the B6/12A CPU board. The files are sent by the Serial Communication Program. Instead of typing the commands over the file is sent by the program.

```
* ==================
* Load Move Routine
* ==================


*************************************************
*                                               *
*              "Move" Program                   *
*              ==============                   *
*                                               *
* This Command File loads the "Move" Program.   *
* This program which starts at 0:1000H takes    *
* the Pattern memory from 0:2000H -> 0:2FFF and  *
* loads it into the Test Hardware's Pattern     *
* memory. At the same time it loads the Test    *
* Hardware's returned Pattern memory into       *
* locations 0:3000H -> 0:3FFFH. This is the     *
* pattern input from the test. The reason for   *
* this program is that Test Hardware's Memory   *
* uses the Pattern Output Bank when writing to  *
* it, and it uses the Pattern Returned Bank when *
* reading from it. If you write to the Test     *
* Hardware and read it back you get a different  *
* number which confuses the Move and Substitute *
* commands of the 8086 Monitor.                 *
*                                               *
* 0:2000 - 0:2FFF Pattern Output Buffer Area    *
* 0:3000 - 0:3FFF Pattern Returned Buffer Area  *
* 0:8000 - 0:8FFF Test Hardware Address Area    *
*                                               *
*************************************************

*       MOV   AX,0000H  ; Clear AX
s0:1000=b8/00/00
*       MOV   DS,AX     ; Clear Data Segment Register
s0:1003=8e/d8
*       MOV   ES,AX     ; Clear Extra Segment Register
s0:1005=8e/c0
*       CLD             ; Clear Direction Flag
s0:1007=fc
*       MOV   SI,2000H  ; Set Source of Pattern
s0:1008=be/00/20
*       MOV   DI,8000H  ; Set Destination Test Hardware
s0:100b=bf/00/80
*       MOV   CX,0800H  ; Set Length
s0:100e=b9/00/08
* MLOOP MOVSW           ; Memory to Memory Move
s0:1011=a5
*       LOOP  MLOOP     ; Loop Until End
s0:1012=e2/fd
*       MOV   SI,8000H  ; Set Source Test Hardware
s0:1014=be/00/80
```

```
*       MOV   DI,3000H    ; Set Destination Pattern Returned
s0:1017=bf/00/30
*       MOV   CX,0800H    ; Set Length
s0:101a=b9/00/08
* ELOOP MOVSW             ; Memory to Memory Move
s0:101d=a5
*       LOOP  ELOOP       ; Loop Until End
s0:101e=e2/fd
*       JMP   0FE00:06F1 ; Return to Monitor
s0:1020=ea/f1/06/00/fe

* ==========
* Finished Load
* ==========
```

```
* ============
* Load Ram Test
* ============


************************************************
*                                              *
* This Command File loads Test Patterns for    *
* the HM6148LP 1K by 4 Static Ram Chip. This    *
* test is by no means exhaustive and will only  *
* really pick up faults with stuck or shorted   *
* data lines. The Address Lines are Not Tested. *
*                                              *
************************************************


*                   RAM TIMING DIAGRAMS
*                   ===================
*
* Definition of Terms:
* 1 = High
* . = Low
* ? = Undefined
* X = Defined
* - = Tri-State
*
*                   READ
*
*          |     |     |     |     |
*  CS/      11111 ..... ..... 11111
*  WE/      11111 11111 11111 11111
*  ADDR     ????? XXXXX XXXXX ?????
*  DATA     ----- ----- ----- -----
*
*                   WRITE
*
*          |     |     |     |     |
*  CS/      11111 ..... ..... 11111
*  WE/      11111 ..... ..... 11111
*  ADDR     ????? XXXXX XXXXX ?????
*  DATA     ----- XXXXX XXXXX -----
*
*
*                   RAM LINE DEFINITIONS
*                   ====================
*
*  1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16
*  A0  A1  A2  A3   4  A5  A6  A7  A8  A9  WE/ CS/ IO1 IO2 IO3 IO4
*
*
*                   PATTERN OUTPUT CONTROL
*                   ======================
```

```
*
* T = Tri-State Control (0 = Tri-State) (1 = Driven)
* V = Value             (0 = Low      ) (1 = High  )
*
* 0:2000 = Pattern Output Buffer Area (Lower  8 Tri-State Bits)
* 0:2800 = Pattern Output Buffer Area (Higher 8 Tri-State Bits)
*
* 0:2000  T  V  T  V  T  V  T  V    T  V  T  V  T  V  T  V
*          4  4  3  3  2  2  1  1    8  8  7  7  6  6  5  5
*
* 0:2800  T  V  T  V  T  V  T  V    T  V  T  V  T  V  T  V
*         12 12 11 11 10 10  9  9   16 16 15 15 14 14 13 13
*
*
*                    PATTERN RETURNED DATA
*                    =====================
*
* a = Low  Input (Referenced to 0.8 Volts)
* b = High Input (Referenced to 2.4 Volts)
*
* 0:3000 = Pattern Returned Buffer Area (Lower  8 Tri-State Bits)
* 0:3800 = Pattern Returned Buffer Area (Higher 8 Tri-State Bits)
*
* 0:3000  b  a  b  a  b  a  b  a    b  a  b  a  b  a  b  a
*          4  4  3  3  2  2  1  1    8  8  7  7  6  6  5  5
*
* 0:3800  b  a  b  a  b  a  b  a    b  a  b  a  b  a  b  a
*         12 12 11 11 10 10  9  9   16 16 15 15 14 14 13 13
*
*
* Setup Default Pattern
* ---------------------
s0:2000=aa/aa
s0:2800=fa/00
m0:2000#7fe,0:2002
*****
m0:2800#7fe,0:2802
*****

* Now Load Test Pattern
* ---------------------

* <=== Write 000,0
s0:2004=aa/aa/aa/aa/aa/aa/aa/aa
s0:2804=fa/00/aa/aa/aa/aa/fa/00

* <=== Read 000
s0:200c=aa/aa/aa/aa/aa/aa/aa/aa
s0:280c=fa/00/ba/00/ba/00/fa/00

* <=== Write 000,F
s0:2014=aa/aa/aa/aa/aa/aa/aa/aa
s0:2814=fa/00/aa/ff/aa/ff/fa/00
```

```
* <=== Read 000
s0:201c=aa/aa/aa/aa/aa/ a/aa/aa
s0:281c=.a/00/ba/00/f. 00/fa/00

* <=== Write 000,1
s0:2024=aa/aa/aa/aa/aa/aa/aa/aa
s0:2824=fa/00/aa/ab/aa/ab/fa/00

* <=== Read 000
s0:202c=aa/aa/aa/aa/aa/aa/aa/aa
s0:282c=fa/00/ba/00/ba/00/fa/00

* <=== Write 000,2
s0:2034=aa/aa/aa/aa/aa/aa/aa/aa
s0:2834=fa/00/aa/ae/aa/ae/fa/00

* <=== Read 000
s0:203c=aa/aa/aa/aa/aa/aa/aa/aa
s0:283c=fa/00/ba/00/ba/00/fa/00

* <=== Write 000,4
s0:2044=aa/aa/aa/aa/aa/aa/aa/aa
s0:2844=fa/00/aa/ba/aa/ba/fa/00

* <=== Read 000
s0:204c=aa/aa/aa/aa/aa/aa/aa/aa
s0:284c=fa/00/ba/00/ba/00/fa/00

* <=== Write 000,8
s0:2054=aa/aa/aa/aa/aa/aa/aa/aa
s0:2854=fa/00/aa/ea/aa/ea/fa/00

* <=== Read 000
s0:205c=aa/aa/aa/aa/aa/aa/aa/aa
s0:285c=fa/00/ba/00/ba/00/fa/00

* <=== Write 3FF,0
s0:2064=aa/aa/ff/ff/ff/ff/aa/aa
s0:2864=fa/00/af/aa/af/aa/fa/00

* <=== Read 3FF
s0:206c=aa/aa/ff/ff/ff/ff/aa/aa
s0:286c=fa/00/bf/00/bf/00/fa/00

* <=== Write 3FF,F
s0:2074=aa/aa/ff/ff/ff/ff/aa/aa
s0:2874=fa/00/af/ff/af/ff/fa/00

* <=== Read 3FF
s0:207c=aa/aa/ff/ff/ff/ff/aa/aa
s0:287c=fa/00/bf/00/bf/00/fa/00

* <=== Write 3FF,1
s0:2084=aa/aa/ff/ff/ff/ff/aa/aa
```

```
s0:2884=fa/00/af/ab/af/ab/fa/00

* <=== Read 3FF
s0:208c=aa/aa/ff/ff/ff/ff/aa/aa
s0:288c=fa/00/bf/bf/bf/00/fa/00

* <=== Write 3FF,2
s0:2094=aa/aa/ff/ff/ff/ff/aa/aa
s0:2894=fa/00/af/ae/af/ae/fa/00

* <=== Read 3FF
s0:209c=aa/aa/ff/ff/ff/ff/aa/aa
s0:289c=fa/00/bf/00/bf/00/fa/00

* <=== Write 3FF,4
s0:20a4=aa/aa/ff/ff/ff/ff/aa/aa
s0:28a4=fa/00/af/be/af/be/fa/00

* <=== Read 3FF
s0:20ac=aa/aa/ff/ff/ff/ff/aa/aa
s0:28ac=fa/00/bf/00/bf/00/fa/00

* <=== Write 3FF,8
s0:20b4=aa/aa/ff/ff/ff/ff/aa/aa
s0:28b4=fa/00/af/ea/af/ea/fa/00

* <=== Read 3FF
s0:20bc=aa/aa/ff/ff/ff/ff/aa/aa
s0:28bc=fa/00/bf/00/bf/00/fa/00

* Get Access
* ----------
ow100,0110

* Load Test Hardware Ram
* ----------------------
g0:1000

*****
* Reset State
* -----------
ow100,1500
ow100,1501
ow100,0110

* Set System Looping
* ------------------
ow100,7422

* ==========
* End Of Load
* ==========
```

```
* ====================
* Test and Save Pattern
* ====================

*************************************************
*                                               *
* This Command File Tests the RAM Chip and saves *
* the pattern. This is normally done on a known *
* good chip and then Unknown Chips are Retested *
* using the "Re-Test" Commar ; File which will  *
* compare against the pattern saved in this     *
* Test. In this way faults can be isolated.     *
*                                               *
*************************************************

* Reset State
* -----------
ow100,1500
ow100,1501
ow100,0110

* Initiate Test Run
* -----------------
ow100,7422

* Select Address
* --------------
ow100,[illegible]

* Get Pattern Returned
* --------------------
r:100
**********

* Save Good Pattern
* -----------------

ar:100#1000,0:4000
**********

* Reset State
* -----------
ow100,1500
ow100,1501
ow100,0110

* Set System Looping
* ------------------
ow100,7422

* ==========
```

* End of Test
* ==========

```
* ========================================
* Re-Test and Compare with Saved Pattern
* ========================================


************************************************
*                                              *
* This Command File Re-Tests the RAM Chip being *
* tested and compares the Pattern Returned with *
* a previously saved pattern setup by the "Test" *
* Command File. The Differences are Displayed if *
* any. In this way Faults can be detected if the *
* initial test was on a good RAM Chip.          *
*                                              *
************************************************

* Reset State
* -----------
ow100,1500
ow100,1501
ow100,0110

* Do Test Run
* -----------
ow100,5422

* Get Access
* -----------
ow100,0110

* Get Pattern Returned
* --------------------
g0:1000
**********

* Compare with Good Pattern
* -------------------------

c0:3000#d0,0:4000
*****
c0:3800#d0,0:4800
*****

* Reset State
* -----------
ow100,1500
ow100,1501
ow100,0110

* Set System Looping
* ------------------
ow100,7422
```

```
* ==========
* End of Re-Test
* ==========
```

```
* ========================
* Display Pattern Obtained
* ========================


*************************************************
*                                               *
* This Command File shows the Pattern Obtained  *
* from the Test Hardware in doing the RAM Test. *
* The Pattern is displayed along with a comment *
* to Describe what Instruction the Pattern      *
* represents. The pattern displayed is taken    *
* from the Pattern Returned Buffer Area, as this *
* program assumes that the "Move" Program has   *
* already moved the Pattern Returned into the   *
* Pattern Returned Buffer Area.                 *
*                                               *
*************************************************

* ===> Write 000,0
d0:3008#8
d0:3808#8

* ===> Read 000
d0:3010#8
d0:3810#8

* ===> Write 000,F
d0:3018#8
d0:3818#8

* ===> Read 000
d0:3020#8
d0:3820#8

* ===> Write 000,1
d0:3028#8
d0:3828#8

* ===> Read 000
d0:3030#8
d0:3830#8

* ===> Write 000,2
d0:3038#8
d0:3838#8

* ===> Read 000
d0:3040#8
d0:3840#8

* ===> Write 000,4
```

```
d0:3048#8
d0:3848#8

* ==> Read 000
d0:3050#8
d0:3850#8

* ==> Write 000,8
d0:3058#8
d0:3858#8

* ==> Read 000
d0:3060#8
d0:3860#8

* ==> Write 3FF,0
d0:3068#8
d0:3868#8

* ==> Read 3FF
d0:3070#8
d0:3870#8

* ==> Write 3FF,F
d0:3078#8
d0:3878#8

* ==> Read 3FF
d0:3080#8
d0:3880#8

* ==> Write 3FF,1
d0:3088#8
d0:3888#8

* ==> Read 3FF
d0:3090#8
d0:3890#8

* ==> Write 3FF,2
d0:3098#8
d0:3898#8

* ==> Read 3FF
d0:30a0#8
d0:38a0#8

* ==> Write 3FF,4
d0:30a8#8
d0:38a8#8

* ==> Read 3FF
d0:30b0#8
d0:38b0#8
```

```
* ===> Write 3FF,8
d0:30b8f8
d0:38b8f8

* ===> Read 3FF
d0:30c0f8
d0:38c0f8

* ==========
* End of Display
* ==========
```

```
* ========================
* Run Logic Analyser Mode
* ========================

*****************************************************
*                                                   *
* This Command File Sets the Test Hardware Up       *
* in the Logic Analyser Mode. All Lines are         *
* placed in Tri-State, and the Clock is driven      *
* from the Test Hardware. Input Lines are           *
* sampled on the Negative edge of the clock.        *
* When the Logic Analyser has Triggered the         *
* number being displayed will stabilise. The        *
* actual trigger point in the Pattern returned      *
* buffer area after using the "Move" Program is     *
* 3900H + ( 2 * Lowest 10 bits of number from       *
* status register). This is for the Lower 8         *
* Tri-State Bits or Lower 16 Bi-State bits,         *
* a , b1 .... a8, b8. For the Higher 8 Tri-State    *
* bits or 16 Bi-State bits the Address is           *
* 3800H + ( 2 * Lowest 10 bits of number from)      *
*                                                   *
*****************************************************

* Load Tri-State Pattern
* ----------------------
ow0:2000=00
bw0:2000@1000,0,255
*****

* Set Address
* -----------
ow100,0110
iw100

* Load Test Hardware Base
* -----------------------
pw0:1000
*****

* Reset State
* -----------
ow100,1500
ow100,1501
ow100,0110

* Set Logic Analyser Going
* ------------------------
ow100,6c22
100000<iw100>
```

```
* Get Access
* ----------
ow100,0110

* Get Pattern Returned
* --------------------
g0:1000
*****

* Display Returned Pattern (Lower 16 Bits)
* ----------------------------------------
d0:3000#800

* Display Returned Pattern (Upper 16 Bits)
* ----------------------------------------
d0:3800#800

* ========================
* End of Logic Analyser
* ========================
```

```
* ====================
* LA Pos Edge Speed=0
* ====================


*****************************************************
*                                                   *
* This Command File Starts the Logic Analyser       *
* to read on Positive Edge and With Speed = 0.      *
* Speed = 0 means the Test Hardware is in the       *
* non-multiplexing mode.                            *
*                                                   *
*****************************************************

* Reset State
* -----------
ow100,1500
ow100,1501
ow100,0116

* Set LA Off
* -----------
ow100,7422
10<iw100>

* ==========
* End of LA
* ==========
```

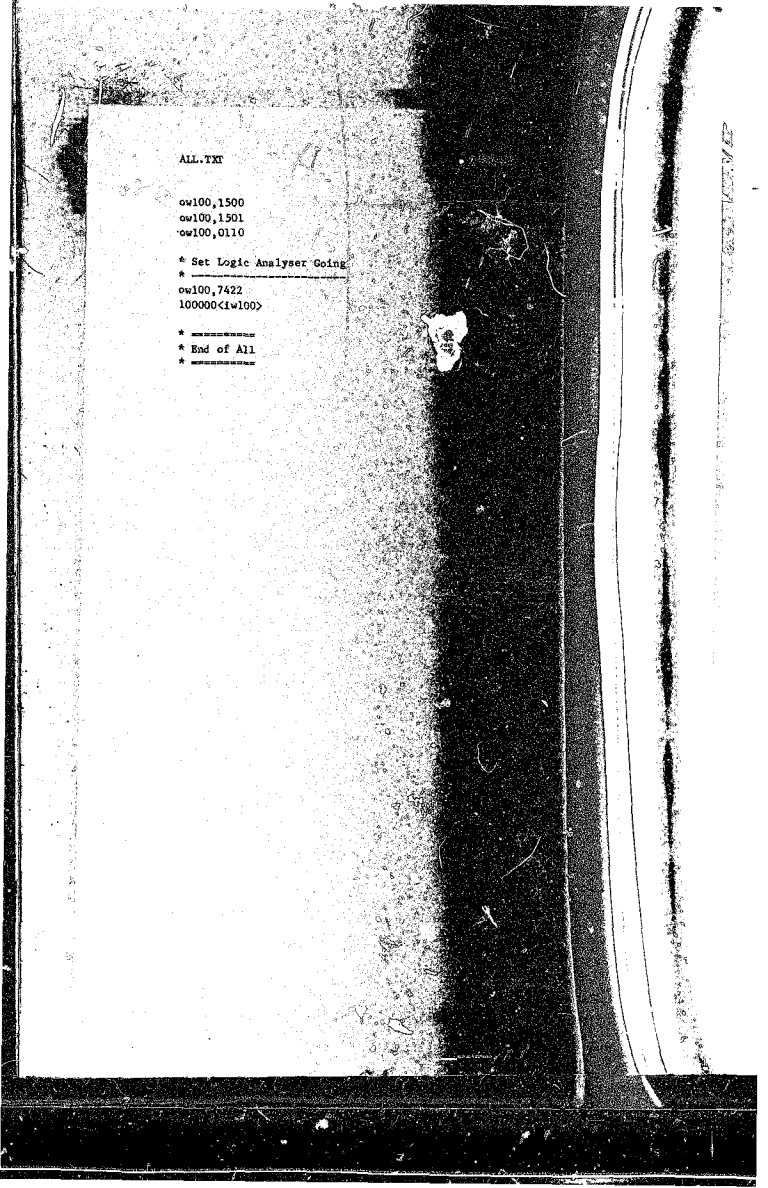
```
* ============
* Move Routine
* ============


*************************** ***********************
*                                                 *
* This Command File basically gets control of     *
* the test hardware and initiates a move by       *
* calling the "Move" Program.                     *
*                                                 *
***************************************************

* Get Access
* ----------
ow100,0110
iw100

* Do Move Routine
* ---------------
g0:1000

*****

* Display Pattern Input
* ---------------------
d0:2000#30
d0:2800#30

* Display Pattern Returned
* ------------------------
d0:3000#30
d0:3800#30

* ============
* End of Move
* ============
```

```
* ==================
* Reset into Tri-State
* ==================


**************************************************
*                                                *
* This Command File basically resets the Test    *
* hardware and places all outputs into           *
* Tri-State. The Pattern Output Buffer Area and  *
* the Pattern Output Memory in the Test Hardware *
* are both cleared.                              *
*                                                *
**************************************************

* Reset State
* -----------
ow100,1300
ow100,1301
ow100,0110

* Zero Pattern Memory
* -------------------
s0:2000=00
m0:2000f1000,0:2001
*****

* Move Pattern into Test Hardware
* -------------------------------
g0:1000
*****

* Output First location (Into Tri-State)
* --------------------------------------
ow100,5400
ow100,5401

* Reset State
* -----------
ow100,1300
ow100,1301
ow100,0110

* ==========
* End of Reset
* ==========
```

```
* ======
* Do All
* ======


*************************************************
*                                               *
* This Command File works on the Assumption that *
* the Test System Has just been powered up. It   *
* first loads the "Move" program and then sets   *
* up an oscillating pattern in the Pattern       *
* memory. The system is then placed in the Logic *
* Analyser mode, where it waits for the trigger  *
* and samples on the positive edge of the clock. *
*                                               *
*************************************************

* Load Move Program
* -----------------
s0:1000=b8/00/00
s0:1003=8e/d8
s0:1005=8e/c0
s0:1007=fc
s0:1008=be/00/20
s0:100b=bf/00/80
s0:100e=b9/00/08
s0:1011=a5
s0:1012=e2/fd
s0:1014=be/00/80
s0:1017=bf/00/30
s0:101a=b9/00/08
s0:101d=a5
s0:101e=e2/fd
s0:1020=ea/f1/06/00/fe

* Load Oscillating Pattern
* ------------------------
s0:2000=aa/aa/ff/ff
m0:2000#1000,0:2004
**************

* Get Access
* ----------
ow100,0110

* Load Test Hardware Ram
* ----------------------
g0:1000
**************

* Reset State
* -----------
```

ALL.TXT

```
ow100,1500
ow100,1501
ow100,0110

* Set Logic Analyser Going
* ------------------------
ow100,7422
100000<iw100>

* ==========
* End of All
* ==========
```



APPENDIX N

RAM TEST EXAMPLE

INTRODUCTION

This is the sequence of commands and the reply from the Test-Hardware used to test a HM6148-LP 1024-word X 4-bit High Speed CMOS RAM chip. The test is first done on a good chip and the pattern stored. Subsequent tests on the RAM with various faults introduced compare the Returned-Pattern with the Expected-Pattern which was stored.

```
.*********************
.*                   *
.* Test HM6148LP RAM *
.*                   *
.*********************
.
.*------------------------------------------------------------------------
.
.* ================
.* Load Move Routine
.* ================
.
.
.**************************************************
.*                                                *
.*               "Move" Program                   *
.*               ==============                   *
.*                                                *
.* This Command File loads the "Move" Program.    *
.* This program which starts at [illegible] takes *
.* the Pattern memory [illegible] and             *
.* loads it into the Test Hardware's Pattern      *
.* memory. At the same time it loads the Test     *
.* hardware's returned pattern memory into        *
.* [illegible] This is the                        *
.* [illegible] from the test. The reason for      *
.* [illegible] is that Test Hardware's Memory     *
.* uses [illegible] pattern output bank when writing to *
.* it, and it uses the Pattern Returned bank when *
.* reading from it. If you write to the Test      *
.* Hardware and read it back you get a different  *
.* number which confuses the Move and Substitute  *
.* commands of the [illegible].                   *
.*                                                *
.* 0:2000 - 0:2FFF Pattern Output Buffer Area     *
.* 0:3000 - 0:3FFF Pattern Returned Buffer Area   *
.* 0:8000 - 0:8FFF Test Hardware Address Area     *
.*                                                *
.**************************************************
.
.*      MOV   AX,0000H   ; Clear AX
.s0:1000=b8/00/00
.*      MOV   DS,AX      ; Clear Data Segment Register
.s0:1003=8e/d8
.*      MOV   ES,AX      ; Clear Extra Segment Register
.s0:1005=8e/c0
.*      CLD              ; Clear Direction Flag
.s0:1007=fc
.*      MOV   SI,2000H   ; Set Source of Pattern
.s0:1008=be/00/20
.*      MOV   DI,8000H   ; Set Destination Test Hardware
.s0:100b=bf/00/80
.*      MOV   CX,0800H   ; Set Length
```

```
.s0:100e=b9/00/08
.* MLOOP MOVSW              ; Memory to Memory Move
.s0:1011=a5
.*       LOOP  MLOOP        ; Loop Until End
.s0:1012=e2/fd
.*       MOV   SI,8000H     ; Set Source Test Hardware
.s0:1014=be/00/80
.*       MOV   DI,3000H     ; Set Destination Pattern Returned
.s0:1017=bf/00/30
.*       MOV   CX,0800H     ; Set Length
.s0:101a=b9/00/08
.* ELOOP MOVSW              ; Memory to Memory Move
.s0:101d=a5
 *       LOOP  ELOOP        ; Loop Until End
.s0:101e=e2/fd
.*       JMP   0FE00:06F1 ; Return to Monitor
.s0:1020=ea/f1/06/00/fe
.
.* ============
.* Finished Load
.* ============
.
.
.*-------------------------------------------------------------------------
.
.
.* ============
.* Load Ram Test
.* ============
.
.
.************************************************
.*                                              *
.* This Command File loads Test Patterns for    *
.* the RM6148LP 1K by 4 Static Ram Chip. This    *
.* test is by no means exhaustive and will only  *
.* really pick up faults with stuck or shorted   *
.* data lines. The Address Lines are Not Tested. *
.*                                              *
.************************************************
.
.
.*                    RAM TIMING DIAGRAMS
.*                    ===================
.*
.* Definition of Terms:
.* I = High
.* . = Low
.* ? = Undefined
.* X = Defined
.* ~ = Tri-State
.*
.*                    READ
.*
```

```
.*          !     !     !     !     !
.*  CS/      11111 ..... ..... 11111
.*  WE/      11111 11111 11111 11111
.*  ADDR     ????? XXXXX XXXXX ?????
.*  DATA     ----- ----- ----- -----
.*
.*                      WRITE
.*
.*          !     !     !     !     !
.*  CS/      11111 ..... ..... 11111
.*  WE/      11111 ..... ..... 11111
.*  ADDR     ????? XXXXX XXXXX ?????
.*  DATA     ----- XXXXX XXXXX -----
.*
.*
.*                      RAM LINE DEFINITIONS
.*                      ====================
.*
.*  1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16
.*  A0  A1  A2  A3  A4  A5  A6  A7  A8  A9  WE/ CS/ IO1 IO2 IO3 IO4
.*
.*
.*                      PATTERN OUTPUT CONTROL
.*                      ======================
.*
.* T = Tri-State Control (0 = Tri-State) (1 = Driven)
.* V = Value             (0 = Low      ) (1 = High   )
.*
.* 0:2000 = Pattern Output Buffer Area (Lower  8 Tri-State Bits)
.* 0:2800 = Pattern Output Buffer Area (Higher 8 Tri-State Bits)
.*
.* 0:2000  T  V  T  V  T  V  T  V    T  V  T  V  T  V  T  V
.*          4  4  3  3  2  2  1  1    8  8  7  7  6  6  5  5
.*
.* 0:2800  T  V  T  V  T  V  T  V    T  V  T  V  T  V  T  V
.*         12 12 11 11 10 10  9  9   16 16 15 15 14 14 13 13
.*
.*
.*                      PATTERN RETURNED DATA
.*                      =====================
.*
.* a = Low  Input (Referenced to 0.8 Volts)
.* b = High Input (Referenced to 2.4 Volts)
.*
.* 0:3000 = Pattern Returned Buffer Area (Lower  8 Tri-State Bits)
.* 0:3800 = Pattern Returned Buffer Area (Higher 8 Tri-State Bits)
.*
.* 0:3000  b  a  b  a  b  a  b  a    b  a  b  a  b  a  b  a
.*          4  4  3  3  2  2  1  1    8  8  7  7  6  6  5  5
.*
.* 0:3800  b  a  b  a  b  a  b  a    b  a  b  a  b  a  b  a
.*         12 12 11 11 10 10  9  9   16 16 15 15 14 14 13 13
.*
.*
```

```
.* Setup Default Pattern
.* ---------------------
.s0:2000=aa/aa
.s0:2800=fa/00
.m0:2000#7fe,0:2002
.*****
.m0:2800#7fe,0:2802
.*****
.
.* Now Load Test Pattern
.* ---------------------
.
.* <=== Write 000,0
.s0:2004=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2804=fa/00/aa/aa/aa/aa/fa/00
.
.* <=== Read 000
.s0:200c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:280c=fa/00/ba/00/ba/00/fa/00
.
.* <=== Write 000,F
.s0:2014=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2814=fa/00/aa/ff/aa/ff/fa/00
.
.* <=== Read 000
.s0:201c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:281c=fa/00/ba/00/ba/00/fa/00
.
.* <=== Write 000,1
.s0.2024=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2824=fa/00/aa/ab/aa/ab/fa/00
.
.* <=== Read 000
.s0:202c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:282c=fa/00/ba/00/ba/00/fa/00
.
.* <=== Write 000,2
.s0:2034=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2834=fa/00/aa/ae/aa/ae/fa/00
.
.* <=== Read 000
.s0:203c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:283c=fa/00/ba/00/ba/00/fa/00
.
.* <=== Write 000,4
.s0:2044=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2844=fa/00/aa/ba/aa/ba/fa/00
.
.* <=== Read 000
.s0:204c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:284c=fa/00/ba/00/ba/00/fa/00
.
.* <=== Write 000,8
.s0:2054=aa/aa/aa/aa/aa/aa/aa/aa
```

```
.s0:2854=fa/00/aa/ea/aa/ea/fa/00
.
.* <==- Read 000
.s0:205c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:285c=fa/00/ba/00/ba/00/fa/00
.
.* <==- Write 3FF,0
.s0:2064=aa/aa/ff/ff/ff/ff/aa/aa
.s0:2864=fa/00/af/aa/af/aa/fa/00
.
.* <==- Read 3FF
.s0:206c=aa/aa/ff/ff/ff/ff/aa/aa
.s0:286c=fa/00/bf/00/bf/00/fa/00
.
.* <==- Write 3FF,F
.s0:2074=aa/aa/ff/ff/ff/ff/aa/aa
.s0:2874=fa/00/af/ff/af/ff/fa/00
.
.* <==- Read 3FF
.s0:207c=aa/aa/ff/ff/ff/ff/aa/aa
.s0:287c=fa/00/bf/00/bf/00/fa/00
.
.* <==- Write 3FF,1
.s0:2084=aa/aa/ff/ff/ff/ff/aa/aa
.s0:2884=fa/00/af/ab/af/ab/fa/00
.
.* <==- Read 3FF
.s0:208c=aa/aa/ff/ff/ff/ff/aa/aa
.s0:288c=fa/00/bf/00/bf/00/fa/00
.
.* <==- Write 3FF,2
.s0:2094=aa/aa/ff/ff/ff/ff/aa/aa
.s0:2894=fa/00/af/ae/af/ae/fa/00
.
.* <==- Read 3FF
.s0:209c=aa/aa/ff/ff/ff/ff/aa/aa
.s0:289c=fa/00/bf/00/bf/00/fa/00
.
.* <==- Write 3FF,4
.s0:20a4=aa/aa/ff/ff/ff/ff/aa/aa
.s0:28a4=fa/00/af/ba/af/ba/fa/00
.
.* <==- Read 3FF
.s0:20ac=aa/aa/ff/ff/ff/ff/aa/aa
.s0:28ac=fa/00/bf/00/bf/00/fa/00
.
.* <==- Write 3FF,8
.s0:20b4=aa/aa/ff/ff/ff/ff/aa/aa
.s0:28b4=fa/00/af/ea/af/ea/fa/00
.
.* <==- Read 3FF
.s0:20bc=aa/aa/ff/ff/ff/ff/aa/aa
.s0:28bc=fa/00/bf/00/bf/00/fa/00
.
```

```
.* Get Access
.* ----------
.ow100,0110
.
.* Load Test Hardware Ram
.* ----------------------
.g0:1000

iSBC 86/12 MONITOR V2.0
.*****
.* Reset State
.* -----------
.ow100,1500
.ow100,1501
.ow100,0110
.
.* Set System Looping
.* ------------------
.ow100,7422
.
.* ==========
.* End Of Load
.* ==========
.
.
.*-------------------------------------------------------------------------
.
.
.* ==========================
.* Test and Save Pattern
.* ==========================
.
.
.************************************************
.*                                              *
.* This Command File Tests the RAM Chip and saves *
.* the pattern. This is normally done on a known  *
.* good chip and then Unknown Chips are Retested  *
.* using the "Re-Test" Command File which will    *
.* compare against the pattern saved in this      *
.* Test. In this way faults can be isolated.      *
.*                                              *
.************************************************
.
.* Reset State
.* -----------
.ow100,1500
.ow100,1501
.ow100,0110
.
.* Do Test Run
.* -----------
.ow100,5422
.
```

```
.* Get Access
.* ----------
.owl00,0110
.
.* Get Pattern Returned
.* --------------------
.g0:1000

iSBC 86/12 MONITOR V2.0
.*********
.
.* Save Good Pattern
.* ----------------
.
.m0:3000f1000,0:4000
.********
.
.* Reset State
.* -----------
.owl00,1500
.owl00,1501
.owl00,0110
.
.* Set System Looping
.* ------------------
.owl00,7422
.
.* ==========
.* End of Test
.* ==========
.
.
.*-------------------------------------------------------------------------
.
.
.* ==========================================
.* Re-Test and Compare with Saved Pattern
.* ==========================================
.
.
.************************************************
.*                                              *
.* This Command File Re-Tests the RAM Chip being *
.* tested and compares the Pattern Returned with *
.* a previously saved pattern setup by the "Test" *
.* Command File. The Differences are Displayed if *
.* any. In this way Faults can be detected if the *
.* initial test was on a good RAM Chip.          *
.*                                              *
.************************************************
.
.* Reset State
.* -----------
.owl00,1500
```

```
.ow100,1501
.ow100,0110
.
.* Do Test Run
.* ----------
.ow100,5422
.
.* Get Access
.* ----------
.ow100,0110
.
.* Get Pattern Returned
.* ---------------------
.g0:1000

iSBC 86/12 MONITOR V2.0
.*********
.
.* Compare with Good Pattern
.* ------------------------
.
.c0:3000#d0,0:4000
.*****                              ( No Differences Lower 8 Bits )
.c0:3800#d0,0:4800
.*****                              ( No Lifferences Upper 8 Bits )
.
.* Reset State
.* ----------
.ow100,1500
.ow100,1501
.ow100,0110
.
.* Set System Looping
.* ------------------
.ow100,7422
.
.* ==========
.* End of Re-Test
.* ==========
.
.
.*-------------------------------------------------------------------------------
.
.
.* ====================
.* Display Pattern Obtained
.* ====================
.
.
.*************************************************
.*                                               *
.* This Command File shows the Pattern Obtained  *
.* from the Test Hardware in doing the RAM Test. *
.* The Pattern is displayed along with a comment *
```

```
.* to Describe what Instruction the Pattern       *
.* represents. The pattern displayed is taken     *
.* from the Pattern Returned Buffer Area, as this *
.* program assumes that the "Move" Program has     *
.* already moved the Pattern Returned into the     *
.* Pattern Returned Buffer Area.                   *
.*                                                 *
.*************************************************
.
.* ===> Write 000,0
.d0:3008#8
0000.3008 00 00 00 00 00 00 00 00                    *........*
.d0:3808#8
0000:3808 F0 55 00 00 00 00 F0 55                    *.U.....U*
.
.* ===> Read 000
.d0:3010#8
0000:3010 00 00 00 00 00 00 00 00                    *........*
.d0:3810#8
0000:3810 F0 55 30 00 30 00 F0 55                    *.U0.0..U*
.
.* ===> Write 000,F
.d0:3018#8
0000:3018 00 00 00 00 00 00 00 00                    *........*
.d0:3818#8
0000:3818 F0 55 00 FF 00 FF F0 55                    *.U.....U*
.
.* ===> Read 000
.d0:3020#8
0000:3020 00 00 00 00 00 00 00 00                    *........*
.d0:3820#8
0000:3820 F0 55 30 FF 30 FF F0 55                    *.U0.0..U*
.
.* ===> Write 000,1
.d0:3028#8
0000:3028 00 00 00 00 00 00 00 00                    *........*
.d0:3828#8
0000:3828 F0 55 00 03 00 03 F0 55                    *.U.....U*
.
.* ===> Read 000
.d0:3030#8
0000:3030 00 00 00 00 00 00 00 00                    *........*
.d0:3830#8
0000:3830 F0 55 30 03 30 03 F0 55                    *.U0.0..U*
.
.* ===> Write 000,2
.d0:3038#8
0000:3038 00 00 00 00 00 00 00 00                    *........*
.d0:3838#8
0000:3838 F0 55 00 0C 00 0C F0 55                    *.U.....U*
.
.* ===> Read 000
.d0:3040#8
0000:3040 00 00 00 00 00 00 00 00                    *........*
```

```
.d0:3840@8
0000:3840 F0 55 30 0C 30 0C F0 55                    *.U0.0..U*
.
.* ==> Write 000,4
.d0:3848@8
0000:3848 00 00 00 00 00 00 00 00                    *........*
.d0:3848@8
0000:3848 F0 55 00 30 00 30 F0 55                    *.U.0.0.U*
.
.* ==> Read 000
.d0:3850@8
0000:3850 00 00 00 00 00 00 00 00                    *........*
.d0:3850@8
0000:3850 F0 55 30 30 30 30 F0 55                    *.U0000.U*
.
.* ==> Write 000,8
.d0:3858@8
0000:3858 00 00 00 00 00 00 00 00                    *........*
.d0:3858@8
0000:3858 F0 55 00 C0 00 C0 F0 55                    *.U.....U*
.
.* ==> Read 000
.d0:3860@8
0000:3860 00 00 00 00 00 00 00 00                    *........*
.d0:3860@8
0000:3860 F0 55 30 C0 30 C0 F0 55                    *.U0.0..U*
.
.* ==> Write 3FF,0
.d0:3868@8
0000:3868 00 00 FF FF FF FF 00 00                    *........*
.d0:3868@8
0000:3868 F0 55 0F 00 0F 00 F0 55                    *.U.....U*
.
.* ==> Read 3FF
.d0:3870@8
0000:3870 00 00 FF FF FF FF 00 00                    *........*
.d0:3870@8
0000:3870 F0 55 3F 00 3F 00 F0 55                    *.U?.?..U*
.
.* ==> Write 3FF,F
.d0:3878@8
0000:3878 00 00 FF FF FF FF 00 00                    *........*
.d0:3878@8
0000:3878 F0 55 0F FF 0F FF F0 55                    *.U.....U*
.
.* ==> Read 3FF
.d0:3880@8
0000:3880 00 00 FF FF FF FF 00 00                    *........*
.d0:3880@8
0000:3880 F0 55 3F FF 3F FF F0 55                    *.U?.?..U*
.
.* ==> Write 3FF,1
.d0:3888@8
0000:3888 00 00 FF FF FF FF 00 00                    *........*
```

```
.d0:3888#8
0000:3888 F0 55 0F 03 0F 03 F0 55                    *.U.....U*
.
.* ===> Read 3FF
.d0:3090#8
0000:3090 00 00 FF FF FF FF 00 00                    *........*
.d0:3890#8
0000:3890 F0 55 3F 03 3F 03 F0 55                    *.U?.?..U*
.
.* ===> Write 3FF,2
.d0:3098#8
0000:3098 00 00 FF FF FF FF 00 00                    *........*
.d0:3898#8
0000:3898 F0 55 0F 0C 0F 0C F0 55                    *.U.....U*
.
.* ===> Read 3FF
.d0:30a0#8
0000:30A0 00 00 FF FF FF FF 00 00                    *........*
.d0:38a0#8
0000:38A0 F0 55 3F 0C 3F 0C F0 55                    *.U?.?..U*
.
.* ===> Write 3FF,4
.d0:30a8#8
0000:30A8 00 00 FF FF FF FF 00 00                    *........*
.d0:38a8#8
0000:38A8 F0 55 0F 30 0F 30 F0 55                    *.U.0.0.U*
.
.* ===> Read 3FF
.d0:30b0#8
0000:30B0 00 00 FF FF FF FF 00 00                    *........*
.d0:38b0#8
0000:38B0 F0 55 3F 30 3F 30 F0 55                    *.U?0?0.U*
.
.* ===> Write 3FF,8
.d0:30b8#8
0000:30B8 00 00 FF FF FF FF 00 00                    *........*
.d0:38b8#8
0000:38B8 F0 55 0F C0 0F C0 F0 55                    *.U.....U*
.
.* ===> Read 3FF
.d0:30c0#8
0000:30C0 00 00 FF FF FF FF 00 00                    *........*
.d0:38c0#8
0000:38C0 F0 55 3F C0 3F C0 F0 55                    *.U?.?..U*
.
.* ==========
.* End of Display
.* ==========
.
.
.*-------------------------------------------------------------------------
.
.
.****************************************
```

```
.*                                      *
.* Now Introduce a Fault and do a Retest *
.*                                      *
.* Short Data Lines 3 and 4.            *
.*                                      *
.****************************************
.
.
.*---------------------------------------------------------------------
.
.
.* ==================================
.* Re-Test and Compare with Saved Pattern
.* ==================================
.
.
.************************************************
.*                                              *
.* This Command File Re-Tests the RAM Chip being *
.* tested and compares the Pattern Returned with *
.* a previously saved pattern setup by the "Test" *
.* Command File. The Differences are Displayed if *
.* any. In this way Faults can be detected if the *
.* initial test was on a good RAM Chip.         *
.*                                              *
.************************************************
.
.* Reset State
.* -----------
.ow100,1500
.ow100,1501
.ow100,0110
.
.* Do Test Run
.* -----------
.ow100,5422
.
.* Get Access
.* ----------
.ow100,0110
.
.* Get Pattern Returned
.* --------------------
.g0:1000

iSBC 86/12 MONITOR V2.0
.*********
.
.* Compare with Good Pattern
.* ------------------------
.
.c0:3000#d0,0:4000
.*****                              ( No Differences Lower 8 Bits )
.c0:3800#d0,0:4800
```

```
0000:384B 00 30 0000:4B4B
0000:384D 00 30 0000:4B4D
0000:3853 00 30 0000:4853
0000:3855 00 30 0000:4855
0000:385B 00 C0 0000:485B
0000:385D 00 C0 0000:485D
0000:3863 00 C0 0000:4863
0000:3865 00 C0 0000:4865
0000:38AB 00 30 0000:48AB
0000:38AD 00 30 0000:48AD
0000:38B3 00 30 0000:48B3
0000:38B5 00 30 0000:48B5
0000:38BB 00 C0 0000:48BB
0000:38BD 00 C0 0000:48BD
0000:38C3 00 C0 0000:48C3
0000:38C5 00 C0 0000:48C5
.*****                                ( 16 Differences Upper 8 Bits )
.
.* Reset State
.* -----------
.ow100,1500
.ow100,1501
.ow100,0110
.
.* Set System Looping
.* -------------------
.ow100,7422
.
.* ============
.* End of Re-Test
.* ==============
.
.
.*-------------------------------------------------------------------
.
.
.* ==========================
.* Display Pattern Obtained
.* ==========================
.
.
.*************************************************
.*                                               *
.* This Command File shows the Pattern Obtained  *
.* from the Test Hardware in doing the RAM Test. *
.* The Pattern is displayed along with a comment *
.* to Describe what Instruction the Pattern      *
.* represents. The pattern displayed is taken    *
.* from the Pattern Returned Buffer Area, as this *
.* program assumes that the "Move" Program has   *
.* already moved the Pattern Returned into the   *
.* Pattern Returned Buffer Area.                 *
.*                                               *
.*************************************************
```

```
.
.* ==> Write 000,0
.d0:3008#8
0000:3008 00 00 00 00 00 00 00 00                    *........*
.d0:3808#8
0000:3808 F0 55 00 00 00 00 F0 55                    *.U.....U*
.
.* ==> Read 000
.d0:3010#8
0000:3010 00 00 00 00 00 00 00 00                    *........*
.d0:3810#8
0000:3810 F0 55 30 00 30 00 F0 55                    *.U0.0..U*
.
.* ==> Write 000,F
.d0:3018#8
0000:3018 00 00 00 00 00 00 00 00                    *........*
.d0:3818#8
0000:3818 F0 55 00 FF 00 FF F0 55                    *.U.....U*
.
.* ==> Read 000
.d0:3020#8
0000:3020 00 00 00 00 00 00 00 00                    *........*
.d0:3820#8
0000:3820 F0 55 30 FF 30 FF F0 55                    *.U0.0..U*
.
.* ==> Write 000,1
.d0:3028#8
0000:3028 00 00 00 00 00 00 00 00                    *........*
.d0:3828#8
0000:3828 F0 55 00 03 00 03 F0 55                    *.U.....U*
.
.* ==> Read 000
.d0:3030#8
0000:3030 00 00 00 00 00 00 00 00                    *........*
.d0:3830#8
0000:3830 F0 55 30 03 30 03 F0 55                    *.U0.0..U*
.
.* ==> Write 000,2
.d0:3038#8
0000:3038 00 00 00 00 00 00 00 00                    *........*
.d0:3838#8
0000:3838 F0 55 00 0C 00 0C F0 55                    *.U.....U*
.
.* ==> Read 000
.d0:3040#8
0000:3040 00 00 00 00 00 00 00 00                    *........*
.d0:3840#8
0000:3840 F0 55 30 0C 30 0C F0 55                    *.U0.0..U*
.
.* ==> Write 000,4
.d0:3048#8
0000:3048 00 00 00 00 00 00 00 00                    *........*
.d0:3848#8
0000:3848 F0 55 00 00 00 00 F0 55                    *.U.....U*
```

```
.*                    ^    ^
.* Should Be        30    30
.*
.
.* ===> Read 000
.d0:3050/8
0000:3050 00 00 00 00 00 00 00 00                    *........*
.d0:3850/8
0000:3850 F0 55 30 00 30 00 F0 55                    *.U0.0..U*
.*                    ^    ^
.* Should Be        30    30
.*
.
.* ===> Write 000,8
.d0:3058/8
0000:3058 00 00 00 00 00 00 00 00                    *........*
.d0:3858/8
0000:3858 F0 55 00 00 00 00 F0 55                    *.U.....U*
.*                    ^    ^
.* Should Be        C0    C0
.*
.
.* ===> Read 000
.d0:3060/8
0000:3060 00 00 00 00 00 00 00 00                    *........*
.d0:3860/8
0000:3860 F0 55 30 00 30 00 F0 55                    *.U0.0..U*
.*                    ^    ^
.* Should Be        C0    C0
.*
.
.* ===> Write 3FF,0
.d0:3068/8
0000:3068 00 00 FF FF FF FF 00 00                    *........*
.d0:3868/8
0000:3868 F0 55 0F 00 0F 00 F0 55                    *.U.....U*
.
.* ===> Read 3FF
.d0:3070/8
0000:3070 00 00 FF FF FF FF 00 00                    *........*
.d0:3870/8
0000:3870 F0 55 3F 00 3F 00 F0 55                    *.U?.?..U*
.
.* ===> Write 3FF,F
.d0:3078/8
0000:3078 00 00 FF FF FF FF 00 00                    *........*
.d0:3878/8
0000:3878 F0 55 0F FF 0F FF F0 55                    *.U.....U*
.
.* ===> Read 3FF
.d0:3080/8
0000:3080 00 00 FF FF FF FF 00 00                    *........*
.d0:3880/8
0000:3880 F0 55 3F FF 3F FF F0 55                    *.U?.?..U*
```

```
.
.* ===> Write 3FF,1
.d0:3088@8
0000:3088 00 00 FF FF FF FF 00 00                    *........*
.d0:3888@8
0000:3888 F0 55 0F 03 0F 03 F0 55                    *.U.....U*
.
.* ===> Read 3FF
.d0:3090@8
0000:3090 00 00 FF FF FF FF 00 00                    *........*
.d0:3890@8
0000:3890 F0 55 3F 03 3F 03 F0 55                    *.U?.?..U*
.
.* ===> Write 3FF,2
.d0:3098@8
0000:3098 00 00 FF FF FF FF 00 00                    *........*
.d0:3898@8
0000:3898 F0 55 0F 0C 0F 0C F0 55                    *.U.....U*
.
.* ===> Read 3FF
.d0:30a0@8
0000:30A0 00 00 FF FF FF FF 00 00                    *........*
.d0:38a0@8
0000:38A0 F0 55 3F 0C 3F 0C F0 55                    *.U?.?..U*
.
.* ===> Write 3FF,4
.d0:30a8@8
0000:30A8 00 00 FF FF FF FF 00 00                    *........*
.d0:38a8@8
0000:38A8 F0 55 0F 00 0F 00 F0 55                    *.U.....U*
.*                  ^     ^
.* Should Be        30    30
.*
.
.* ===> Read 3FF
.d0:30b0@8
0000:30B0 00 00 FF FF FF FF 00 00                    *........*
.d0:38b0@8
0000:38B0 F0 55 3F 00 3F 00 F0 55                    *.U?.?..U*
.*                  ^     ^
.* Should Be        30    30
.*
.
.* ===> Write 3FF,8
.d0:30b8@8
0000:30B8 00 00 FF FF FF FF 00 00                    *........*
.d0:38b8@8
0000:38B8 F0 55 0F 00 0F 00 F0 55                    *.U.....U*
.*                  ^     ^
.* Should Be        C0    C0
.*
.
.* ===> Read 3FF
.d0:30c0@8
```

```
0000:30C0 00 00 FF FF FF 00 00                    *........*
.d0:38c0f8
0000:38C0 F0 55 3F 00 3F 00 F0 55                 *.U?.?..U*
.*                  ^     ^
.* Should Be        C0    C0
.*
.
.* ==========
.* End of Display
.* ==========
.
.
.********************
.*                  *
.* End of RAM Test  *
.*                  *
.********************
```

APPENDIX O

LOGIC-ANALYSER EXAMPLE

INTRODUCTION

This is an example of the Test-Hardware being used in the Logic-Analyser mode where all outputs are defined as being in Tri-State and the Test-Hardware is placed in the Logic-Analyser Mode. The circuit used to test this mode is a simple counter.

```
.**************************
.*                        *
.* Logic Analyser Example *
.*                        *
.**************************
.
.
.*-------------------------------------------------------------------------
.
.
.* ==================
.* Load Move Routine
.* ==================
.
.
.*************************************************
.*                                               *
.*              "Move" Program                   *
.*              ==============                   *
.*                                               *
.* This Command File loads the "Move" Program.   *
.* This program which starts at 0:1000H takes    *
.* the Pattern memory from 0:2000H -> 0:2FFF and  *
.* loads it into the Test Hardware's Pattern     *
.* memory. At the same time it loads the Test    *
.* Hardware's returned Pattern memory into       *
.* locations 0:3000H -> 0:3FFFH. This is the      *
.* pattern input from the test. The reason for   *
.* this program is that Test Hardware's Memory   *
.* uses the Pattern Output Bank when writing to  *
.* it, and it uses the Pattern Returned Bank when*
.* reading from it. If you write to the Test     *
.* Hardware and read it back you get a different  *
.* number which confuses the Move and Substitute *
.* commands of the 8086 Monitor.                 *
.*                                               *
.* 0:2000 - 0:2FFF Pattern Output Buffer Area    *
.* 0:3000 - 0:3FFF Pattern Returned Buffer Area  *
.* 0:8000 - 0:8FFF Test Hardware Address Area    *
.*                                               *
.*************************************************
.
.*      MOV   AX,#0000H  ; Clear AX
.s0:1000=b8/00/00
.*      MOV   DS,AX      ; Clear Data Segment Register
.s0:1003=8e/d8
.*      MOV   ES,AX      ; Clear Extra Segment Register
.s0:1005=8e/c0
.*      CLD              ; Clear Direction Flag
.s0:1007=fc
.*      MOV   SI,2000H   ; Set Source of Pattern
.s0:1008=be/00/20
.*      MOV   DI,8000H   ; Set Destination Test Hardware
```

```
.s0:100b=bf/00/80
.*       MOV   CX,0800H    ; Set Length
.s0:100e=b9/00/08
.* MLOOP MOVSW              ; Memory to Memory Move
.s0:1011=a5
.*       LOOP  MLOOP       ; Loop Until End
.s0:1012=e2/fd
.*       MOV   SI,8000H    ; Set Source Test Hardware
.s0:1014=be/00/80
.*       MOV   DI,3000H    ; Set Destination Pattern Returned
.s0:1017=bf/00/30
.*       MOV   CX,0800H    ; Set Length
.s0:101a=b9/00/08
.* ELOOP MOVSW              ; Memory to Memory Move
.s0:101d=a5
.*       LOOP  ELOOP       ; Loop Until End
.s0:101e=e2/fd
.*       JMP   0FEC0:06F1 ; Return to Monitor
.s0:1020=ea/f1/06/00/fe
.
.* ============
.* Finished Load
.* ============
.
.
.*-------------------------------------------------------------------------------
.
.
.* ====================
.* Run Logic Analyser Mode
.* ====================
.
.
.*************************************************
.*                                               *
.* This Command File Sets the Test Hardware Up   *
.* in the Logic Analyser Mode. All Lines are     *
.* placed in Tri-State, and the Clock is driven  *
.* from the Test Hardware. Input Lines are       *
.* sampled on the Negative edge of the clock.    *
.* When the Logic Analyser has Triggered the     *
.* number being displayed will stabilise. The    *
.* actual trigger point in the Pattern returned  *
.* buffer area after using the "Move" Program is *
.* 3000H + ( 2 * Lowest 10 bits of number from   *
.* status register). This is for the Lower 8     *
.* Tri-State Bits or Lower 16 Bi-State bits,     *
.* a1, b1 .... a8, b8. For the Higher 8 Tri-State *
.* bits or 16 Bi-State bits the Address is       *
.* 3800H + ( 2 * Lowest 10 bits of status reg.)  *
.*                                               *
.*************************************************
.
.* Load Tri-State Pattern
```

```
.* ----------------------
.s0:2000=00
.m0:2000#1000,0:2001
.***
.
.* Get Access
.* ----------
.ow100,0110
.iw100
AF8A
.
.* Load Test Hardware Ram
.* ----------------------
.g0:1000

iSBC 86/12 MONITOR V2.0
.****
.
.* Reset State
.* ----------
.ow100,1500
.ow100,1501
.ow100,0110
.
.* Set Logic Analyser Going
.* ----------------------
.ow100,6c22
.100000<iw100>
ADA7
AFFD
AF51
AEA7
AD06
AF54
AEC4
AD29
AF91
AEE7
AE44
AD9E
ACEE
AC3F
AF96
AEDL
AE3F
AD9B
ACF1
AC3D
A5A4
ADF5
AD4A
AC9D
AFF5
AF5A
```

```
AEAA
ADFE
AD56
ACAD
AC0F
AE68
ADC3
AD20
AC81
CDAA                                    ( <- Trigger Point )
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
CDAA
*Control C*
.
.* Get Access
.* ----------
.ow100,0110
.
.* Get Pattern Returned
.* --------------------
.g0:1000

iSBC 86/12 MONITOR V2.0
.****
.
.* Display Returned Pattern (Lower 16 Bits)
.* ----------------------------------------
.d0:3000#800
0000:3000 C3 3C CC 3C CF 3C F0 3C F3 3C FC 3C FF 3C 00 3C  *.<.<.<.<.<.<.<.<*
0000:3010 03 3C 0C 3C 0F 3C 30 3C 33 3C 3C 3C 3F 3C C0 3F  *.<.<.<0<3<<<?<.?*
0000:3020 C3 3F CC 3F CF 3F F0 3F F3 3F FC 3F FF 3F 00 3F  *.?.?.?.?.?.?.?.?*
0000:3030 03 3F 0C 3F 0F 3F 30 3F 33 3F 3C 3F 3F 3F C0 C0  *.?.?.?0?3?<???..*
0000:3040 C3 C0 CC C0 CF C0 F0 C0 F3 C0 FC C0 FF C0 00 C0  *................*
0000:3050 03 C0 0C C0 0F C0 30 C0 33 C0 3C C0 3F C0 C0 C3  *......0.3.<.?...*
0000:3060 C3 C3 CC C3 CF C3 F0 C3 F3 C3 FC C3 FF C3 00 C3  *................*
0000:3070 03 C3 0C C3 0F C3 30 C3 33 C3 3C C3 3F C3 C0 CC  *......0.3.<.?...*
0000:3080 C3 CC CC CC CF CC F0 CC F3 CC FC CC FF CC 00 CC  *................*
0000:3090 03 CC 0C CC 0F CC 30 CC 33 CC 3C CC 3F CC C0 0F  *......0.3.<.?...*
```

```
0000:30A0 C3 CF CC CF CF CF F0 CF F3 CF FC CF FF CF 00 CF  *...............*
0000:30B0 03 CF 0C CF 0F CF 30 CF 33 CF 3C CF 3F CF C0 F0  *......0.3.<.?...*
0000:30C0 C3 F0 CC F0 CF F0 F0 F0 F3 F0 FC F0 FF F0 00 F0  *...............*
0000:30D0 03 F0 0C F0 0F F0 30 F0 33 F0 3C F0 3F F0 C0 F3  *......0.3.<.?...*
0000:30E0 C3 F3 CC F3 CF F3 F0 F3 F3 F3 FC F3 FF F3 00 F3  *...............*
0000:30F0 03 F3 0C F3 0F F3 30 F3 33 F3 3C F3 3F F3 C0 FC  *......0.3.<.?...*
0000:3100 C3 FC CC FC CF FC F0 FC F3 FC FC FC FF FC 00 FC  *...............*
0000:3110 03 FC 0C FC 0F FC 30 FC 33 FC 3C FC 3F FC C0 FF  *......0.3.<.?...*
0000:3120 C3 FF CC FF CF FF F0 FF F3 FF FC FF FF FF 00 FF  *...............*
0000:3130 03 FF 0C FF 0F FF 30 FF 33 FF 3C FF 3F FF C0 00  *......0.3.<.?...*
0000:3140 C3 00 CC 00 CF 00 F0 00 F3 00 FC 00 FF 00 00 00  *...............*
0000:3150 03 00 0C 00 0F 00 30 00 33 00 3C 00 3F 00 C0 03  *......0.3.<.?...*
0000:3160 C3 03 CC 03 CF 03 F0 03 F3 03 FC 03 FF 03 00 03  *...............*
0000:3170 03 03 0C 03 0F 03 30 03 33 03 3C 03 3F 03 C0 0C  *......0.3.<.?...*
0000:3180 C3 0C CC 0C CF 0C F0 0C F3 0C FC 0C FF 0C 00 0C  *...............*
0000:3190 03 0C 0C 0C 0F 0C 30 0C 33 0C 3C 0C 3F 0C C0 0F  *......0.3.<.?...*
00
*Control C*
.
.* Display Returned Pattern (Upper 16 Bits)
.* ------------------------------------
.d0:3800@800
0000:3800 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3810 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3820 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3830 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3840 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3850 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3860 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3870 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3880 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3890 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:38A0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:38B0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:38C0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:38D0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:38E0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:38F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3900 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3910 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3920 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3930 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3940 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3950 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3960 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  *UUUUUUUUUUUUUUUU*
0000:3970 55 55 55 55 55 55 55 55
*Control C*
.
.* ====================
.* End of Logic Analyser
.* ====================
.
.
.*-------------------------------------------------------------------------
```

```
.
.
.**************************
.*                        *
.* Display Trigger Point *
.*                        *
.**************************
.*
.* Status Register was CDAA (H)
.* Lower 10 bits = 1AA
.* Address = 0:3000 + 1AA * 2
.*         = 0:3000 + 354
.*         = 0:3354
.*
.d0:3300@200
0000:3300 3F F3 C0 FC FC FC C3 FC CC FC CF FC F0 FC F3 FC  *?..............*
0000:3310 FC FC FF FC 00 FC 03 FC 0C FC 0F FC 30 FC 33 FC  *............0.3.*
0000:3320 3C FC 3F FC C0 FF C3 FF CC FF CF FF F0 FF F3 FF  *<.?.............*
0000:3330 FC FF FF FF 00 FF 03 FF 0C FF 0F FF 30 FF 33 FF  *............0.3.*
0000:3340 F0 00 3C FF 3F FF C0 00 C3 00 CC 00 CF 00 F0 00  *..<.?...........*
0000:3350 F3 00 FC 00 FF 00 00 00 03 00 0C 00 0F 00 30 00  *..............0.*
.*                              ^^^^
0000:3360 33 00 3C 00 3F 00 C0 03 C3 03 CC 03 CF 03 F0 03  *3.<.?...........*
0000:3370 F3 03 FC 03 FF 03 00 03 03 03 0C 03 0F 03 30 03  *..............0.*
0000:3380 33 03 3C 03 3F 03 C0 0C C3 0C CC 0C CF 0C F0 0C  *3.<.?...........*
0000:3390 F3 0C FC 0C FF 0C 00 0C 03 0C 0C 0C 0F 0C 30 0C  *..............0.*
0000:33A0 33 0C 3C 0C 3F 0C C0 0F C3 0F CC 0F CF 0F F0 0F  *3.<.?...........*
0000:33B0 F3 0F FC 0F FF 0F 00 0F 03 0F 0C 0F 0F 0F 30 0F  *..............0.*
0000:33C0 33 0F 3C 0F 3F 0F C0 30 C3 30 CC 30 CF 30 F0 30  *3.<.?..0.0.0.0.0*
0000:33D0 F3 30 FC 30 FF 30 00 30 03 30 0C 30 0F 30 30 30  *.0.0.0.0.0.0.000*
0000:33E0 33 30 3C 30 3F 30 C0 33 C3 33 CC 33 CF 33 F0 33  *30<0?0.3.3.3.3.3*
0000:33F0 F3 33 FC 33 FF 33 00 33 03 33 0C 33 0F 33 30 33  *.3.3.3.3.3.3.303*
0000:3400 33 33 3C 33 3F 33 C0 3C C3 3C CC 3C CF 3C F0 3C  *33<3?3.<.<.<.<.<*
0000:3410 F3 3C FC 3C FF 3C 00 3C 03 3C 0C 3C 0F 3C 30 3C  *.<.<.<.<.<.<.<0<*
0000:3420 33 3C 3C 3C 3F 3C C0 3F C3 3F CC 3F CF 3F F0 3F  *3<<<?<.?.?.?.?.?*
0000:3430 F3 3F FC 3F FF 3F 00 3F 03 3F 0C 3F 0F 3F 30 3F  *.?.?.?.?.?.?.?0?*
0000:3440 33 3F 3C 3F 3F 3F C0 C0 C3 C0 CC C0 CF C0 F0 C0  *3?<???..........*
0000:3450 F3 C0 FC C0 FF C0 00 C0 03 C0 0C C0 0F C0 30 C0  *..............0.*
0000:3460 33 C0 3C C0 3F C0 C0 C0 C3 C0 CC C0 CF C0 F0 C0  *3.<.?...........*
0000:3470 F3 C0 FC C0 FF C0 00 C0 03 C0 0C C0 0F C0 30 C0  *..............0.*
0000:3480 33 C0 3C C0 3F C0 C0 CC C3 CC CC CC CF CC F0 CC  *3.<.?...........*
0000:3490 F3 CC FC CC FF CC 00 CC 03 CC 0C CC 0F CC 30 CC  *..............0.*
0000:34A0 33 CC 3C CC 3F CC C0 CF C3 CF CC CF CF CF F0 CF  *3.<.?...........*
0000:34B0 F3 CF FC CF FF CF 00 CF 03 CF 0C CF 0F CF 30 CF  *..............0.*
0000:34C0 33 CF 3C CF 3F CF FF F0 C3 F0 CC F0 CF F0 F0 F0  *3.<.?...........*
0000:34D0 F3 F0 FC F0 FF F0 00 F0 03 F0 0C F0 0F F0 30 F0  *..............0.*
0000:34E0 33 F0 3C F0 3F F0 C0 F3 C3 F3 CC F3 CF F3 F0 F3  *3.<.?...........*
0000:34F0 F3 F3 FC F3 FF F3 00 F3 03 F3 0C F3 0F F3 30 F3  *..............0.*
.*
.*
.* Note Trigger was on Value 00 00
.*
.*
.**********************************
```

```
.*                                  *
.* End of Logic Analyser Example *
.*                                  *
.**********************************
```