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
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.

Name of candidate

17th day of JUNE 1985
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

<table>
<thead>
<tr>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECLARATION</td>
<td>(a)</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>(b)</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>(c)</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>(d)</td>
</tr>
<tr>
<td>LIST OF APPENDICES</td>
<td>(h)</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>(i)</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>(j)</td>
</tr>
<tr>
<td>GLOSSARY</td>
<td>(k)</td>
</tr>
</tbody>
</table>

1. INTRODUCTION ......................................................... 81
   1.1 INTRODUCTION ................................................. 81
   1.2 WHY USE AUTOMATIC-TEST-EQUIPMENT ................. 81
   1.3 WHAT IS AUTOMATIC-TEST-EQUIPMENT ................. 83
      1.3.1 Analog Circuits / Digital Circuits .......... 83
      1.3.2 Functional-Testers / In-Circuit-Testers .... 84
   1.4 WHY DEVELOP A LOCAL-TEST-SYSTEM ................. 84
   1.5 AIMS OF THE LOCAL-TEST-SYSTEM DEVELOPED ....... 85
   1.6 OVERVIEW OF THE LOCAL-TEST-SYSTEM DEVELOPED .... 87
   1.7 CONCLUSION ............................................... 11
2. FUNCTIONAL OVERVIEW OF TEST SYSTEMS ---------------------------- 12
   2.1 INTRODUCTION ................................................. 12
   2.2 GENERAL TEST PRINCIPLES .................................... 12
   2.3 BARE BOARD TESTING ........................................... 14
       2.3.1 Bed-of-Nails Test Fixtures 14
   2.4 FUNCTIONAL-TESTING ............................................ 15
       2.4.1 Signature-Analysis 16
       2.4.2 High Frequency Pattern Generation 17
       2.4.3 Slow-Speed-Functional-Tester 18
       2.4.4 Dynamic-Functional-Testing (High Speed) 19
   2.5 IN-CIRCUIT-TESTING .......................................... 20
       2.5.1 In-Circuit-Emulation 20
       2.5.2 Back-Driving 21
   2.6 ESTABLISHING EXPECTED-RESPONSE ............................. 22
       2.6.1 Known-Good-Board 22
       2.6.2 Logic-Simulation 23
   2.7 CONCLUSION ...................................................... 24

3. FAULT MODELLING AND TEST-PATTERN GENERATION ----------------------- 25
   3.1 INTRODUCTION .................................................... 25
   3.2 TYPES OF FAULTS ................................................. 26
       3.2.1 Stuck-At-Faults 26
       3.2.2 Shorts 27
       3.2.3 Parametric Faults 28
   3.3 TYPES OF CIRCUITS .............................................. 28
       3.3.1 Combinational Circuits 29
       3.3.2 Synchronous Sequential Circuits 30
       3.3.3 Asynchronous Sequential Circuits 30
   3.4 FAULT FOLDING .................................................. 31
   3.5 TEST-PATTERN GENERATION ..................................... 32
       3.5.1 Manual Method 33
       3.5.2 Random Method 33
       3.5.3 Automatic Test-Pattern Generation 34
   3.6 CONCLUSION ...................................................... 35
8. TECHNICAL DESCRIPTION OF THE TEST-HARDWARE

6.1 INTRODUCTION ........................................ 65
6.2 FUNCTIONS OF THE TEST-HARDWARE ..................... 66
6.3 MULTIBUS INTERFACE AND CONTROL BOARD ............... 69
   6.3.1 MULTIBUS Interface ............................. 69
   6.3.2 Clock and Timing Control ......................... 73
   6.3.3 Addressing and Mode Control ...................... 77
6.4 MEMORY, LATCH, MULTIPLEXER AND COMPARATOR BOARD ... 78
6.5 ANALOG INTERFACE BOARD ................................ 79
   6.5.1 Bi-State Mode .................................... 79
   6.5.2 Tri-State Mode .................................... 88
   6.5.3 Custom Analog Interfaces ......................... 88
6.6 TIMING AND CONTROL CONNECTOR ........................ 81
6.7 DIGITAL I/O CONNECTOR ................................ 82
6.8 ANALOG I/O CONNECTOR ................................ 83
6.9 TRIGGER-PATTERN SWITCHES ............................. 84
6.10 CONCLUSION ........................................... 84

7. POTENTIAL USES OF THE TEST-SYSTEM .................... 86

7.1 INTRODUCTION ........................................ 86
7.2 STAND ALONE LOGIC-ANALYSER ........................ 87
7.3 STAND ALONE DYNAMIC-FUNCTIONAL-TESTER ............... 88
   7.3.1 Test-Pattern Generation ......................... 88
   7.3.2 Expected-Responses ................................ 89
7.4 SLAVE DYNAMIC-FUNCTIONAL-TESTER ..................... 90
   7.4.1 Test-Pattern Generation ......................... 90
   7.4.2 Expected-Responses .............................. 91
7.5 CONCLUSION ........................................... 91

8. CONCLUSION ............................................. 92

8.1 INTRODUCTION ........................................ 92
8.2 IMPROVEMENTS ......................................... 93
   8.2.1 Signature-Analysis Addition ..................... 93
   8.2.2 Cascading Hardware ................................ 93
8.3 HARDWARE BUGS ........................................ 94
   8.3.1 p Resistor Bug .................................. 94
   8.3.2 Addressing ...................................... 94
8.4 CONCLUSION ........................................... 94

(g)
LIST OF APPENDICES

A. Test-Hardware Printed Circuit Diagrams
B. Test-Hardware Board Layouts
C. Connector Pinouts
D. MULTIBUS Interface Switch Selection / Memory Map
E. Control / Status-Registers and I/O Structure
F. Serial Communication Board Circuit Diagram
G. Serial Communication Board Control
H. Serial Communication Program Listing
I. Logic-Simulator Primitives
J. Logic-Simulator Listing
K. Logic-Simulator Test Circuit
L. Logic-Simulator Output
M. Test-Hardware Control Files
N.Ram Test Example
O. Logic-Analyser Example
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Basic Block Diagram</td>
<td>89</td>
</tr>
<tr>
<td>2. Basic Physical Layout</td>
<td>10</td>
</tr>
<tr>
<td>3. Signature-Analysis</td>
<td>17</td>
</tr>
<tr>
<td>4. Stuck-At-Fault</td>
<td>26</td>
</tr>
<tr>
<td>5. Short Fault</td>
<td>27</td>
</tr>
<tr>
<td>6. Synchronous One-Shot</td>
<td>29</td>
</tr>
<tr>
<td>7. Combinational Circuits</td>
<td>29</td>
</tr>
<tr>
<td>8. Synchronous Sequential Circuits</td>
<td>30</td>
</tr>
<tr>
<td>9. Asynchronous Sequential Circuits</td>
<td>31</td>
</tr>
<tr>
<td>10. Synchronous Cycle Format</td>
<td>38</td>
</tr>
<tr>
<td>11. Table-Driven-Simulator</td>
<td>39</td>
</tr>
<tr>
<td>12. Basic Block Diagram</td>
<td>44</td>
</tr>
<tr>
<td>13. Test-Hardware Functional Diagram</td>
<td>51</td>
</tr>
<tr>
<td>14. Basic Physical Layout</td>
<td>65</td>
</tr>
<tr>
<td>15. Test-Hardware Functional Diagram</td>
<td>67</td>
</tr>
<tr>
<td>16. Speed 8 Timing Diagram (Non-Multiplexed)</td>
<td>75</td>
</tr>
<tr>
<td>17. Speed 1 Timing Diagram (Multiplexed)</td>
<td>76</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Different Configuration Mode</td>
<td>88</td>
</tr>
<tr>
<td>2. Different Configuration Mode</td>
<td>58</td>
</tr>
<tr>
<td>3. Control-Register</td>
<td>58</td>
</tr>
<tr>
<td>4. Status-Register</td>
<td>58</td>
</tr>
<tr>
<td>5. Output Memory Bit Definitions</td>
<td>58</td>
</tr>
<tr>
<td>6. Pattern-Output-Memory Structure</td>
<td>58</td>
</tr>
<tr>
<td>7. Input Memory Bit Definitions</td>
<td>59</td>
</tr>
<tr>
<td>8. Pattern-Returned-Memory Structure</td>
<td>60</td>
</tr>
<tr>
<td>9. Trigger-Pattern Bit Definitions</td>
<td>61</td>
</tr>
<tr>
<td>10. Trigger-Pattern Structure</td>
<td>62</td>
</tr>
<tr>
<td>11. Switch Numbering</td>
<td>70</td>
</tr>
<tr>
<td>12. Default Switch Settings</td>
<td>70</td>
</tr>
<tr>
<td>13. Dedicated I/O Memory Base Address</td>
<td>71</td>
</tr>
<tr>
<td>14. Control/Status-Register Address</td>
<td>72</td>
</tr>
<tr>
<td>15. Wait State Switch Selection</td>
<td>73</td>
</tr>
<tr>
<td>16. Timing and Control Connector</td>
<td>81</td>
</tr>
<tr>
<td>17. Digital I/O Connector</td>
<td>82</td>
</tr>
<tr>
<td>18. Analog I/O Connector</td>
<td>83</td>
</tr>
</tbody>
</table>
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 ULSI 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.
1.3 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.

1.3.1 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.
1.9.2 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-Analyzer (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 elements. This is achieved by forcing the inputs of a component to a known 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-Analyzer is often used as a guided probe in this method.

1.9 WAY 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 implement, 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.
1.4 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 1:

<table>
<thead>
<tr>
<th>Memory/Pin</th>
<th>No. Pins</th>
<th>Max Speed</th>
<th>Detect State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) 1K</td>
<td>32</td>
<td>8 MHz</td>
<td>Bi-State</td>
</tr>
<tr>
<td>2) 1K</td>
<td>16</td>
<td>8 MHz</td>
<td>Tri-State</td>
</tr>
<tr>
<td>3) 2K</td>
<td>16</td>
<td>16 MHz</td>
<td>Bi-State</td>
</tr>
<tr>
<td>4) 2K</td>
<td>8</td>
<td>16 MHz</td>
<td>Tri-State</td>
</tr>
</tbody>
</table>

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 1:

![Test Hardware Block Diagram](image)

**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](image-url)
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 B885 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 Functional 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:

4 Digit Hexadecimal Display

15 Bit Shift Register

Input bit stream: 1100101010...

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, while 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.
2.4.1 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 18 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 successfully 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/no-go 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 board 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 independent 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 propagated out. In this way initialization 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.
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.

![Stuck-at-Fault Diagram](image)

<table>
<thead>
<tr>
<th>Pattern Applied</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1 i2 i3 i4</td>
<td>r1</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

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 demonstrated in the figure below where the shorting of inputs 2 and 3 (12 and 13) cause the function of the circuit to change.

![Diagram of good and faulty models, showing shorts between inputs and outputs, leading to a logical AND of the two shorted lines.]

<table>
<thead>
<tr>
<th>Pattern Applied</th>
<th>Response</th>
<th>Good Machine Model</th>
<th>Faulted Machine Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1 i2 i3 i4</td>
<td>r1 r2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
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.
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 aLatch 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 B.

![Figure B: Synchronous Sequential Circuits](image)

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.

![Asynchronous Sequential Circuit Diagram]

**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 Huehldorf (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 usefulness 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 off in a known state certain initialization 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 initialization 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 usefulness 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 off 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.

3.3.3 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 the Sensitisation Method and D-Algorithm method. These methods work on applying a pattern which will propagate a fault to one of the outputs. The various methods are described in greater detail in Muschdorf (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, Macro circuit 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 Table-Driven-Simulators.

![Synchronous Standard Format](image)

**Figure 10 : Synchronous 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 other 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.
4.2.4 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 frequency 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 AFF DELAY

Most circuits do not have an exact delay. Delay is usually specified as a maximum and minimum value and varies from chip to chip, 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 INITIALIZATION PROBLEM

With Simulators it is possible to detect initialization 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 \((X,) = X)\). 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 dependent 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 variety of 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 and 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 deactivated 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 B88d 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:

![Basic Block Diagram](image)

**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 separate digital inputs and outputs to the tester's driver/sensor pins which drive and sense tri-state signals.

The main functions of the various constituents will now be discussed 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 158 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 6869 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 8866 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 8866 Single-Board-Computer.

A listing of this program is contained in Appendix N, 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 8866 SINGLE-BOARD-COMPUTER

The 8866 Single-Board-Computer was actually an 86/12A SBC manufactured by Intel. This Single-Board-Computer is a MULTIBUS compatible computer with an 8866 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 8250 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 Inteltec - ISBC 86/12A Interface and Execution Package User's Guide by Intel (1988). 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 8B66 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 8B66 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 programmer's 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 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/D 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 TIL 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:

<table>
<thead>
<tr>
<th></th>
<th>Memory/Pin</th>
<th>No.Pins</th>
<th>Max Speed</th>
<th>Detect State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>1K</td>
<td>32</td>
<td>8 MHz</td>
<td>BI - State</td>
</tr>
<tr>
<td>2)</td>
<td>1K</td>
<td>16</td>
<td>8 MHz</td>
<td>TRI - State</td>
</tr>
<tr>
<td>3)</td>
<td>2K</td>
<td>16</td>
<td>16 MHz</td>
<td>BI - State</td>
</tr>
<tr>
<td>4)</td>
<td>2K</td>
<td>8</td>
<td>16 MHz</td>
<td>TRI - State</td>
</tr>
</tbody>
</table>

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 described in the Table below:

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>STONTRIB</th>
<th>0 = Normal Start</th>
<th>Start without Trigger-Pattern matching input.</th>
<th>Undefined for CLEAR=0.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 = Start on Trigger</td>
<td>Start when Trigger-Pattern matches input.</td>
<td></td>
</tr>
<tr>
<td>Bit 14</td>
<td>CLEAR</td>
<td>0 = Clear</td>
<td>Clear Test-Hardware.</td>
<td>Release Clear.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Go</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit 13</td>
<td>NORMLA</td>
<td>0 = Normal Single Scan</td>
<td>Normal single scan mode, and reset Logic-Analyser Mode.</td>
<td>Logic-Analyser mode for looping.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Logic-Analyser On</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

54
<table>
<thead>
<tr>
<th>Bit 12</th>
<th>EPDS</th>
<th>0 = Disable Pos Edge</th>
<th>No positive edge clocking.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 = Enable Pos Edge</td>
<td>Clock on positive edges.</td>
</tr>
<tr>
<td>Bit 11</td>
<td>ENEG</td>
<td>0 = Disable Neg Edge</td>
<td>No negative edge clocking.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Enable Neg Edge</td>
<td>Clock on negative edges.</td>
</tr>
<tr>
<td>Bit 10</td>
<td>EXINT</td>
<td>0 = External Access</td>
<td>External access of memory from ROM (SEC). Disables clocking as well.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Internal Access</td>
<td>Internal Access for running Test Hardware.</td>
</tr>
<tr>
<td>Bit 9</td>
<td>SPEED</td>
<td>0 = Normal Speed</td>
<td>Non-multiplexed mem access.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Full Speed</td>
<td>Multiplexed memory access.</td>
</tr>
<tr>
<td>Bit 8</td>
<td>C2</td>
<td>0 = Enable Clock2</td>
<td>Drive Auxiliary Clock 2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Disable Clock 2</td>
<td>Tri-State Auxiliary Clock 2.</td>
</tr>
<tr>
<td>Bit 7-5</td>
<td>C2,B2,A2</td>
<td>0,0,0 = Single Step</td>
<td>Speed of Clock 2. (Auxiliary Clock).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,0,1 = 60.25 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1,0 = 60.59 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1,1 = 61.88 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0,0 = 82.00 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0,1 = 84.00 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1,0 = 86.08 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1,1 = id.88 MHz</td>
<td></td>
</tr>
<tr>
<td>Bit 4</td>
<td>C1</td>
<td>0 = Enable Clock1</td>
<td>Drive Main Clock 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Disable Clock 1</td>
<td>Tri-State Main Clock 1.</td>
</tr>
<tr>
<td>Bit 3-1</td>
<td>B1,A1</td>
<td>0,0,0 = Single Step</td>
<td>Speed of Clock 1. (Main Clock).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,0,1 = 60.25 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1,0 = 60.59 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1,1 = 61.88 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0,0 = 82.00 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0,1 = 84.00 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1,0 = 86.08 MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1,1 = id.88 MHz</td>
<td></td>
</tr>
<tr>
<td>Bit 8</td>
<td>SSTEP</td>
<td>0 = Step Low</td>
<td>Clock Low Under Single Step.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Step High</td>
<td>Clock High Under Single Step</td>
</tr>
</tbody>
</table>
### 5.8.3 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:

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>TRIG STARTED</th>
<th>0 = Start FF Off</th>
<th>Under triggered start the Test-Hardware has not yet triggered.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 = Start FF On</td>
<td>The Hardware has triggered or is permanently enabled.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 14</th>
<th>LA TRIGGERED</th>
<th>0 = No Trigger</th>
<th>Logic-Analyser not triggered</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 = Triggered</td>
<td>Logic-Analyser has been triggered.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 13</th>
<th>LA FINISHED</th>
<th>0 = LA Finished</th>
<th>Logic-Analyser has finished counting after being triggered.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 = LA Busy</td>
<td>Logic-Analyser is waiting for trigger or is counting after being triggered.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 12</th>
<th>LA CLOCKING</th>
<th>0 = LA Stopped</th>
<th>The Logic-Analyser clock is stopped either because finished or not in LA Mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 = LA Clocking</td>
<td>The Logic-Analyser is operating.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit 11</th>
<th>CLOCKING</th>
<th>0 = Not Clocking</th>
<th>The main counters which clock through memory are stopped after single pass or because disabled.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 = Clocking</td>
<td>The main counters are busy clocking in single pass or are counting during LA Mode before trigger.</td>
</tr>
</tbody>
</table>

---

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

56
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 9-6  | ADDRESS  | Trigger Address  | Address of trigger under LA node. 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 output memory: one at address offset 800H above the memory base, and the second at address offset 800H. 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 800H. 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 800H 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 or 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 drivers are controlled by writing words to the Pattern-Output-Memory using the format detailed in the Table below:

<table>
<thead>
<tr>
<th>Tn = Tri-State Control Bit n</th>
<th>0 = Put Pin in Tri-State</th>
<th>1 = Drive pin according to Vn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vn = Drive Control Bit n</td>
<td>0 = Drive Pin Low</td>
<td>1 = Drive Pin High</td>
</tr>
</tbody>
</table>

**Table 5: Output Memory Bit Definitions**

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

**Speed = 8 (Non-Multiplexed Mode)**

<table>
<thead>
<tr>
<th>At offset 888H (default 0:0800)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Address</td>
</tr>
<tr>
<td>T4   V4   T3   V3   T2   V2   T1   V1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>At offset 888H (default 0:0800)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Address</td>
</tr>
<tr>
<td>T12  V12  T11  V11  T10  V10  T9  V9</td>
</tr>
</tbody>
</table>

**Speed = 1 (Multiplexed Mode)**

<table>
<thead>
<tr>
<th>At offset 888H (default 0:0800) and offset 888H (default 0:0800)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Address</td>
</tr>
<tr>
<td>T4   V4   T3   V3   T2   V2   T1   V1</td>
</tr>
</tbody>
</table>
5.3.3 PATTERN-RETURNED-MEMORY

There are two banks of input memory one at address offset 880H above the memory base address, and the second at address offset 888H. In the non-multiplexed memory access mode the low bits are read into the bank at address offset 880H and the high order bits to the bank at address offset 888H. In the multiplexed memory access mode only the high bits are read and they are put into the low bank at address offset 880H and the high bank at address offset 888H 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 4.8 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:

<table>
<thead>
<tr>
<th>Low Signal</th>
<th>Input High = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input Low = 8</td>
</tr>
<tr>
<td>Tri-State Signal</td>
<td>Input High = 8</td>
</tr>
<tr>
<td></td>
<td>Input Low = 1</td>
</tr>
<tr>
<td>High Signal</td>
<td>Input High = 1</td>
</tr>
<tr>
<td></td>
<td>Input Low = 1</td>
</tr>
</tbody>
</table>

| Hi = Input High Bit n    | 0 = Below High Threshold |
|                         | 1 = Above High Threshold |
| Ll = 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 = 8 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 node 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:

\[
\text{Speed} = 0 \text{ (Non-Multiplexed Mode)}
\]

<table>
<thead>
<tr>
<th>Low Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2 L2 H1 L1</td>
<td>H16 L16 H15 L14</td>
</tr>
<tr>
<td>H12 L12 H11 L10</td>
<td>H16 L16 H15 L14</td>
</tr>
</tbody>
</table>

\[
\text{Speed} = 1 \text{ (Multiplexed Mode)}
\]

<table>
<thead>
<tr>
<th>Low Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>H4 L4 H3 L3</td>
<td>H8 L8 H7 L7</td>
</tr>
<tr>
<td>H4 L4 H3 L3</td>
<td>H8 L8 H7 L7</td>
</tr>
</tbody>
</table>

Table 8: Pattern-Returned-Memory Structure
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 = 8) 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 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 Trigger-Pattern set up on the switches is only operational in the Logic-Analyzer mode and the Triggered Digital-Test Mode. In the normal Digital-Test Mode these switches are not used at all.

The Trigger-Pattern Bit Definitions are detailed in the Table below:

<table>
<thead>
<tr>
<th>Hn = Pattern High Bit</th>
<th>Ln = Pattern Low Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Above High Threshold</td>
<td>Below Low Threshold</td>
</tr>
<tr>
<td>Above Low Threshold</td>
<td>Below High Threshold</td>
</tr>
</tbody>
</table>

The bit definitions are detailed in the Table below:

<table>
<thead>
<tr>
<th>Trigger-Pattern Bit Definitions</th>
<th>Hn</th>
<th>Ln</th>
</tr>
</thead>
<tbody>
<tr>
<td>Above High Threshold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Above Low Threshold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Below Low Threshold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Below High Threshold</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24</th>
<th>23 22 21 20 19 18 17 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>H16 L16 H15 L15 H14 L14 H13 L13</td>
<td>H12 L12 H11 L11 H10 L10 H9 L9 L8</td>
</tr>
<tr>
<td>15 14 13 12 11 10 9 8</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>H8 L8 H7 L7 H6 L6 H5 L5</td>
<td>H4 L4 H3 L3 H2 L2 X1 L1</td>
</tr>
</tbody>
</table>

Table 18: 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 8118H
CLEAR SEQUENCE OF TEST-HARDWARE 1508H 1501H 811BH
SETTING LOGIC-ANALYSER OFF

<table>
<thead>
<tr>
<th>1MHz</th>
<th>1/2MHz</th>
<th>1MHz</th>
<th>2MHz</th>
<th>...</th>
<th>16MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos Edge Speed = 0</td>
<td>7422H</td>
<td>7444H</td>
<td>7466H</td>
<td>7488H</td>
<td>74EEH</td>
</tr>
<tr>
<td>Neg Edge Speed = 0</td>
<td>6C22H</td>
<td>6C44H</td>
<td>6C66H</td>
<td>6C88H</td>
<td>6CEEH</td>
</tr>
<tr>
<td>Both Edges Speed = 0</td>
<td>7C22H</td>
<td>7C44H</td>
<td>7C66H</td>
<td>7C88H</td>
<td>7CEEH</td>
</tr>
<tr>
<td>Pos Edge Speed = 1</td>
<td>7E22H</td>
<td>7E44H</td>
<td>7E66H</td>
<td>7E88H</td>
<td>7E66H</td>
</tr>
<tr>
<td>Neg Edge Speed = 1</td>
<td>6E22H</td>
<td>6E44H</td>
<td>6E66H</td>
<td>6E88H</td>
<td>6EEEH</td>
</tr>
<tr>
<td>Both Edges Speed = 1</td>
<td>7E22H</td>
<td>7E44H</td>
<td>7E66H</td>
<td>7E88H</td>
<td>7EEEH</td>
</tr>
</tbody>
</table>

62
Before the Logic-Analyzer has Triggered get A???H from Status-Register.

After the Logic-Analyzer has Triggered get C???H from Status-Register.

SETTING DIGITAL-TESTER MODE OFF

<table>
<thead>
<tr>
<th>Edge Speed</th>
<th>1/8MHz</th>
<th>1/4MHz</th>
<th>1MHz</th>
<th>2MHz</th>
<th>...</th>
<th>16MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos Edge</td>
<td>5422H</td>
<td>5444H</td>
<td>5466H</td>
<td>5488H</td>
<td>54EEH</td>
<td></td>
</tr>
<tr>
<td>Neg Edge</td>
<td>4C22H</td>
<td>4C44H</td>
<td>4C66H</td>
<td>4C88H</td>
<td>4CEEH</td>
<td></td>
</tr>
<tr>
<td>Both Edges</td>
<td>5C22H</td>
<td>5C44H</td>
<td>5C66H</td>
<td>5C88H</td>
<td>5CEEH</td>
<td></td>
</tr>
</tbody>
</table>

SETTING TRIGGERED DIGITAL-TESTER MODE OFF

<table>
<thead>
<tr>
<th>Edge Speed</th>
<th>1/8MHz</th>
<th>1/4MHz</th>
<th>1MHz</th>
<th>2MHz</th>
<th>...</th>
<th>16MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos Edge</td>
<td>D422H</td>
<td>D444H</td>
<td>D466H</td>
<td>D488H</td>
<td>D4EEH</td>
<td></td>
</tr>
<tr>
<td>Neg Edge</td>
<td>CC22H</td>
<td>CC44H</td>
<td>CC66H</td>
<td>CC88H</td>
<td>CCEEH</td>
<td></td>
</tr>
<tr>
<td>Both Edges</td>
<td>DC22H</td>
<td>DC44H</td>
<td>DC66H</td>
<td>DC88H</td>
<td>DCEEH</td>
<td></td>
</tr>
</tbody>
</table>

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 D for examples of the programming.

In the examples in the Appendix the Control & Status-Register were set up at 1/0 port number 180H. 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 0000:0000H to 0000:0FFF. The lower bank is from 0000:0000 to 0000:07FF and the upper bank is from 0000:0800 to 0000:0FFF. 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.
TECHNICAL DESCRIPTION OF THE TEST-HARDWARE

Chapter 6

4.1 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 8866 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 8866 Single-Board-Computer whose function 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:

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 = 0 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:

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.

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.

3

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

4

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

5

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.

6

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

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 separate boards and the functions of these three boards will now be described in the following sections.

6.2 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 8883 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 Trans. 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 acknowledgement wait.

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

<table>
<thead>
<tr>
<th>--Card Edge--</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (34)</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7 8</td>
</tr>
</tbody>
</table>

Table 11: Switch Numbering

The default switch positions for the memory at 8:8888H -> 8:8FFF and the Control/Status-Register at 180H is in the Table below:

<table>
<thead>
<tr>
<th>A (34)</th>
<th>B (23)</th>
<th>C (12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>1 1 0 0 0 0 0</td>
<td>0 0 0 0 1 0 0 0</td>
</tr>
</tbody>
</table>

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 84 must be open, and closed for a 20 address 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 address is set up according to the following Table:

<table>
<thead>
<tr>
<th>B4 = open for 16 address bits</th>
<th>= closed for 20 address bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ADR13 -&gt; C1</td>
<td>0 = Closed</td>
</tr>
<tr>
<td>/ADR12 -&gt; C2</td>
<td>1 = Open</td>
</tr>
<tr>
<td>/ADR11 -&gt; C3</td>
<td>for /ADR</td>
</tr>
<tr>
<td>/ADR10 -&gt; C4</td>
<td></td>
</tr>
<tr>
<td>/ADR F -&gt; C5</td>
<td>0 = Open</td>
</tr>
<tr>
<td>/ADR E -&gt; C6</td>
<td>1 = Closed</td>
</tr>
<tr>
<td>/ADR D -&gt; C7</td>
<td>for ADR</td>
</tr>
<tr>
<td>/ADR C -&gt; C8</td>
<td></td>
</tr>
</tbody>
</table>

/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:

<table>
<thead>
<tr>
<th>Al = open for 8 address bits</th>
<th>= closed for 12 address bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ADR B -&gt; B5</td>
<td>/ADR A -&gt; B6</td>
</tr>
<tr>
<td>/ADR 9 -&gt; B7</td>
<td>/ADR 8 -&gt; B8</td>
</tr>
<tr>
<td>/ADR 7 -&gt; A2</td>
<td>/ADR 6 -&gt; A3</td>
</tr>
<tr>
<td>/ADR 5 -&gt; A4</td>
<td>/ADR 4 -&gt; A5</td>
</tr>
<tr>
<td>/ADR 3 -&gt; A6</td>
<td>/ADR 2 -&gt; A7</td>
</tr>
<tr>
<td>/ADR 1 -&gt; A8</td>
<td>/ADR 0 = Closed</td>
</tr>
<tr>
<td></td>
<td>/ADR 1 = Open</td>
</tr>
</tbody>
</table>

/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 9 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:

<table>
<thead>
<tr>
<th>P1</th>
<th>B2</th>
<th>B3</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>Closed</td>
<td>Closed</td>
<td>1 Clock Cycle</td>
</tr>
<tr>
<td>Closed</td>
<td>Closed</td>
<td>Open</td>
<td>2 Clock Cycles</td>
</tr>
<tr>
<td>Closed</td>
<td>Open</td>
<td>Closed</td>
<td>3 Clock Cycles</td>
</tr>
<tr>
<td>Closed</td>
<td>Open</td>
<td>Open</td>
<td>4 Clock Cycles</td>
</tr>
<tr>
<td>Open</td>
<td>Closed</td>
<td>Closed</td>
<td>5 Clock Cycles</td>
</tr>
<tr>
<td>Open</td>
<td>Closed</td>
<td>Open</td>
<td>6 Clock Cycles</td>
</tr>
<tr>
<td>Open</td>
<td>Open</td>
<td>Closed</td>
<td>7 Clock Cycles</td>
</tr>
<tr>
<td>Open</td>
<td>Open</td>
<td>Open</td>
<td>8 Clock Cycles</td>
</tr>
</tbody>
</table>

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.

6.2.2 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 8.25 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 separately programmable 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 16 MHz, internal timing was controlled by a 67 MHz 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 = 8 (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:

+ve Edge Clock From D.U.T.  

MXCTL

Bank 1 Controls

\[ \phi_1 \]

Latch In Data & Inc Counter

\[ \phi_2 \]

Latch Out Data

\[ \phi_3 \]

Latch Count Through

\[ \text{WE1} \]

Write Data to Memory

Bank 2 Controls

\[ \phi_1 \]

Latch In Data

\[ \phi_2 \]

Latch Out Data

\[ \phi_3 \]

Latch Count Through

\[ \text{WE2} \]

Write Data to Memory

Figure 16: Speed 8 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 MXTCL 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:

![Timing Diagram](image)

---

Figure 17: Speed 1 Timing Diagram (Multiplexed)
4.9.2 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.
4.5 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.

4.5.1 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.
4.5.2 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 6.0 Volts and the other 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 6.8 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.

4.5.3 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.
### 4.6 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 control signals from one board to the other.

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

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. \( \text{OUT}_n \) are outputs and \( \text{IN}_n \) and \( \text{PAT}_n \) are inputs. The trigger matches when for all \( n \), \( \text{IN}_n \) equals \( \text{PAT}_n \).

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

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

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.

<table>
<thead>
<tr>
<th>Component Side (Top)</th>
<th>Solder Side (Bottom)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I/O 1</td>
</tr>
<tr>
<td>3</td>
<td>I/O 3</td>
</tr>
<tr>
<td>5</td>
<td>I/O 5</td>
</tr>
<tr>
<td>7</td>
<td>I/O 7</td>
</tr>
<tr>
<td>9</td>
<td>I/O 9</td>
</tr>
<tr>
<td>11</td>
<td>I/O 11</td>
</tr>
<tr>
<td>13</td>
<td>I/O 13</td>
</tr>
<tr>
<td>15</td>
<td>I/O 15</td>
</tr>
<tr>
<td>17</td>
<td>I/O 17</td>
</tr>
<tr>
<td>19</td>
<td>I/O 19</td>
</tr>
<tr>
<td>21</td>
<td>I/O 21</td>
</tr>
<tr>
<td>23</td>
<td>I/O 23</td>
</tr>
<tr>
<td>25</td>
<td>I/O 25</td>
</tr>
<tr>
<td>27</td>
<td>I/O 27</td>
</tr>
<tr>
<td>29</td>
<td>I/O 29</td>
</tr>
<tr>
<td>31</td>
<td>I/O 31</td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Voltage A</td>
</tr>
<tr>
<td>39</td>
<td>Voltage B</td>
</tr>
<tr>
<td>41</td>
<td>Voltage C</td>
</tr>
<tr>
<td>43</td>
<td>Voltage D</td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>CLOCK1 Main Clock</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>CLOCK2 Auxiliary Clock</td>
</tr>
</tbody>
</table>

**Table 18: Analog I/O Connector**
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-Analyzer Mode or the Triggered Tester Mode without the Trigger-Pattern matching the Returned-Pattern.

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 possible. 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 PDP 11 Mini Computer, or an Apple II+ Micro Computer. Most uses need a terminal attached to the serial link on the 8886 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 0 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.
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 8686 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 9

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 27 (74LS246) 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 87-90


Diagnosis & Reliable Design of Digital Systems
Pitman Publishing Limited 1976

[4] HENDKELS, L. P. and HASS, R. M. (BenRad, USA)

The Known-good-board approach versus software simulation
Pulse August 1982, Pages 92-94,97

[5] INTEL

ISBC 857A Interlec - ISBC86/12A Interface and Execution
Package User’s Guide

[6] INTEL

ISBC 86/12A Single Board Computer Hardware Reference Manual
Manual Order Number 9883874-01 (1979)
**Simulation of Simple Digital Logic through a Computer-Aided Design System**  
Byte Publications, Volume 8 No 1, January 1983, Pages 396-414

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

[19] MUEHLDORF, E. I. (Senior member IEEE) and SAHAR, A. D.  
**LSI Logic Testing - An Overview**  

[20] POPPY, A. (Millennium 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 118-115

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.

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.


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 87-90

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.
BOURICUS, 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.

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] BROOK, G. (Gould Inc, USA)
Logic analyser trace functions
Pulse November 1982, Pages 84-98

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

[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.

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.
FROHNERK, 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.

GROVES, A. G. (Hewlett Packard)

Rapid Digital Fault Isolation with FASTRACE
Hewlett Packard Journal, March 19??, 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 once a Test-Program has been generated.

HARVEY, K. P. (Fluke SA)

Fluke automated test systems in South Africa
Pulse August 1982, Pages 184-187

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.

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

This manual covers the use of the 88B6 based Single-Board-Computer used in the Test-Hardware.

This article covers the actual implementation of a simple Table Driven Logic-Simulator. It covers the implementation in detail and actually includes the listing in BASIC. Very good article to actually get to grips with what is involved in Logic-Simulators.
This article covers the design of an Advanced Logic-Simulator which has Named Nodes, Macroircuit Capability, Error Checking, Selective Trace and Sampled Output. Primitive elements include all the basic combinatorial 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 I with various improvements. Very useful article about Table Driven Simulators.

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

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.

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 put forward.

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.
ROTH, J. P. (IBM)

Diagnosis of Automata Failures: A Calculus and a Method
IBM Journal, Jul. 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.

ROTH, J.P.(IBM), BOURICUS, W.G.(IBM) and SCHNEIDER, P.R.(IBM)

Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits

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.

SCHURMANN, H. D.(Mem IEEE), LINDBLOM, E. and CARPENTER, R.G.

The Weighted Random Test-Pattern Generator
IEEE Transactions on Computers, Vol C-24, July 1975, Pages 695-708

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.
129) SCHOFIELD, M. (Wayne Kerr, UK)

Automatic digital diagnostics: extend bump-too ATE capability
Pulse August 1982, Pages 95-97

This article basically describes the Wayne Kerr A8888 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.

130) SELLERS, F. F. (Jr mem IEEE), HSIAO, M. Y., and BEARNSON, L. W.

Analyzing Errors with the Boolean Difference

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

131) SU, S. Y. H. (Member IEEE) and CHO, Y. (Student member IEEE)

A New Approach to the Fault Location of Combinational Circuits

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

132) SUBSKIND, A. K. (Lehigh University)

Diagnostics for logic networks
IEEE Spectrum, October 1973, Pages 40-47

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.
133) TAYLOR, E. (Spescom)  
Evaluating board testers  
Pulse August 1982, Page 108

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

134) THOMAS, J. J. (LTV Aerospace Corporation, Dallas, Texas)  
Automatic 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.

135) 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.

136) TO, K. (Western Electric, USA)  
Fault Folding for Irredundant and Redundant Combinational Circuits  

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

<table>
<thead>
<tr>
<th>Number</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>74LS00</td>
<td>Quadruple 2-Input Positive NAND Gate</td>
</tr>
<tr>
<td>10,22</td>
<td>74S02</td>
<td>Quadruple 2-Input Positive NOR Gate</td>
</tr>
<tr>
<td>42</td>
<td>74S84</td>
<td>Hex Inverters</td>
</tr>
<tr>
<td>25,36</td>
<td>74LS04</td>
<td>Hex Inverters</td>
</tr>
<tr>
<td>44</td>
<td>74S88</td>
<td>Quadruple 2-Input Positive AND Gate</td>
</tr>
<tr>
<td>16</td>
<td>74LS88</td>
<td>Quadruple 2-Input Positive AND Gate</td>
</tr>
<tr>
<td>1,14</td>
<td>74LS21</td>
<td>Dual 4-Input Positive AND Gate</td>
</tr>
<tr>
<td>8,19,33</td>
<td>74S32</td>
<td>Quad 2-Input Positive OR Gate</td>
</tr>
<tr>
<td>47</td>
<td>74LS32</td>
<td>Quad 2-Input Positive OR Gate</td>
</tr>
<tr>
<td>52</td>
<td>74S86</td>
<td>Quadruple 2-Input EXCLUSIVE-OR Gate</td>
</tr>
<tr>
<td>2,13</td>
<td>74LS86</td>
<td>Quadruple 2-Input EXCLUSIVE-OR Gate</td>
</tr>
<tr>
<td>10,11,15,21</td>
<td>74S112</td>
<td>Dual J-K Negative Edge Flip-Flops</td>
</tr>
<tr>
<td>49</td>
<td>74S124</td>
<td>Dual Voltage Controlled Oscillators</td>
</tr>
<tr>
<td>7,24,37,48</td>
<td>74S157</td>
<td>Quad 2- to 1-Line Data selectors / MUX's</td>
</tr>
<tr>
<td>4,5,6,27,38,49</td>
<td>74S163</td>
<td>Binary 4 Bit Counter with Synchronous Clear</td>
</tr>
<tr>
<td>46</td>
<td>74164</td>
<td>8 bit Parallel Output Serial Shift Register</td>
</tr>
<tr>
<td>9,20</td>
<td>74S190</td>
<td>4-bit Parallel Access Shift Registers</td>
</tr>
<tr>
<td>31,32</td>
<td>74S197</td>
<td>Binary Presetable Asynchronous Counter</td>
</tr>
<tr>
<td>26</td>
<td>74LS244</td>
<td>Octal Buffer Line Drivers / Receivers</td>
</tr>
<tr>
<td>45</td>
<td>74S251</td>
<td>8- to 1-Line Data Selector / MUX</td>
</tr>
<tr>
<td>29,30</td>
<td>74LS251</td>
<td>8- to 1-Line Data Selector / MUX</td>
</tr>
<tr>
<td>24</td>
<td>74LS266</td>
<td>Quadruple 2-Input EXCLUSIVE-NOR Gates (OC)</td>
</tr>
<tr>
<td>17,39</td>
<td>74LS373</td>
<td>Octal D-Type Latches with Tri-State Outputs</td>
</tr>
<tr>
<td>48</td>
<td>74S374</td>
<td>Octal D-Type Flip-Flops</td>
</tr>
<tr>
<td>41</td>
<td>74LS374</td>
<td>Octal D-Type Flip-Flops</td>
</tr>
<tr>
<td>54</td>
<td>74S224</td>
<td>Octal D-Type Flip-Flops</td>
</tr>
<tr>
<td>58,51,52</td>
<td>74S227</td>
<td>Inverting Octal Bus Transceiver</td>
</tr>
<tr>
<td>35</td>
<td>25LS2521</td>
<td>8 Bit Digital Comparator</td>
</tr>
<tr>
<td>12,23,34</td>
<td>SWITCH</td>
<td>8 Dual in line switches</td>
</tr>
</tbody>
</table>
MULTIBUS INTERFACE

CARD 1

MULTIBUS INTERFACE AND CONTROL BOARD
MULTIBUS INTERFACE AND CONTROL BOARD
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.

<table>
<thead>
<tr>
<th>Number</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>74LS138</td>
<td>3-To-8 Line Decoders / Multiplexers</td>
</tr>
<tr>
<td>6 → 13</td>
<td>74S157</td>
<td>Quad 2- To 1-Line Data Selectors / MUX's</td>
</tr>
<tr>
<td>24 → 31</td>
<td>74LS244</td>
<td>Octal Buffers / Line Drivers/Line Receivers</td>
</tr>
<tr>
<td>18,19,40,41</td>
<td>74S373</td>
<td>Octal D-Type Latches</td>
</tr>
<tr>
<td>24 → 17,28 → 23</td>
<td>74S374</td>
<td>Octal D-Type Flip-Flops</td>
</tr>
<tr>
<td>2 → 5</td>
<td>74LS251</td>
<td>8 Bit Digital Comparator</td>
</tr>
<tr>
<td>32 → 39,42 → 49</td>
<td>HM6148-LP</td>
<td>1024-word X 4-bit High Speed CMOS RAM</td>
</tr>
</tbody>
</table>
MEMORY, LATCH, MULTIPLEXER AND COMPARATOR BOARD
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.

<table>
<thead>
<tr>
<th>Number</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33-46</td>
<td>74LS126</td>
<td>Quadruple Buffers with Tri-State Outputs</td>
</tr>
<tr>
<td>1-32</td>
<td>LM361</td>
<td>High Speed Differential Comparator</td>
</tr>
<tr>
<td>41-44</td>
<td>LM1368</td>
<td>Programmable Power Op Amp</td>
</tr>
<tr>
<td>SA,SB,SC,SD</td>
<td>SWITCHES</td>
<td>Dual in Line switches</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Number</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33-40</td>
<td>74LS126</td>
<td>Quadruple Buffers with Tri-State Outputs</td>
</tr>
<tr>
<td>1-32</td>
<td>LM361</td>
<td>High Speed Differential Comparator</td>
</tr>
<tr>
<td>41-44</td>
<td>LM138BB</td>
<td>Programmable Power Op Amp</td>
</tr>
<tr>
<td>SA,SB,SC,SD</td>
<td>SWITCHES</td>
<td>6 Dual In Line switches</td>
</tr>
</tbody>
</table>
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 actual 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
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 actual 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 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 actual 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
APPENDIX C

CONNECTOR PINOUTS
## Timing and Control Connector

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

<table>
<thead>
<tr>
<th>Component Side</th>
<th>Solder Side</th>
<th></th>
<th>Component Side</th>
<th>Solder Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLOCK1</td>
<td>2</td>
<td>CLOCK2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Manual Trig</td>
<td>4</td>
<td>-</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>OUT 25</td>
<td>6</td>
<td>OUT 17</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>OUT 26</td>
<td>8</td>
<td>OUT 18</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>OUT 27</td>
<td>10</td>
<td>OUT 19</td>
<td>51</td>
</tr>
<tr>
<td>11</td>
<td>OUT 28</td>
<td>12</td>
<td>OUT 20</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>OUT 29</td>
<td>14</td>
<td>OUT 21</td>
<td>57</td>
</tr>
<tr>
<td>15</td>
<td>OUT 30</td>
<td>16</td>
<td>OUT 22</td>
<td>59</td>
</tr>
<tr>
<td>17</td>
<td>OUT 31</td>
<td>18</td>
<td>OUT 23</td>
<td>61</td>
</tr>
<tr>
<td>19</td>
<td>OUT 32</td>
<td>20</td>
<td>OUT 24</td>
<td>63</td>
</tr>
<tr>
<td>21</td>
<td>OUT 9</td>
<td>22</td>
<td>OUT 1</td>
<td>65</td>
</tr>
<tr>
<td>23</td>
<td>OUT 10</td>
<td>24</td>
<td>OUT 2</td>
<td>67</td>
</tr>
<tr>
<td>25</td>
<td>OUT 11</td>
<td>26</td>
<td>OUT 3</td>
<td>69</td>
</tr>
<tr>
<td>27</td>
<td>OUT 12</td>
<td>28</td>
<td>OUT 4</td>
<td>71</td>
</tr>
<tr>
<td>29</td>
<td>OUT 13</td>
<td>30</td>
<td>OUT 5</td>
<td>73</td>
</tr>
<tr>
<td>31</td>
<td>OUT 14</td>
<td>32</td>
<td>OUT 6</td>
<td>75</td>
</tr>
<tr>
<td>33</td>
<td>OUT 15</td>
<td>34</td>
<td>OUT 7</td>
<td>77</td>
</tr>
<tr>
<td>35</td>
<td>OUT 16</td>
<td>36</td>
<td>OUT 8</td>
<td>79</td>
</tr>
<tr>
<td>37</td>
<td>OUT 18</td>
<td>38</td>
<td>IN 17</td>
<td>81</td>
</tr>
<tr>
<td>39</td>
<td>OUT 18</td>
<td>40</td>
<td>PAT 17</td>
<td>83</td>
</tr>
<tr>
<td>41</td>
<td>OUT 20</td>
<td>42</td>
<td>IN 19</td>
<td>85</td>
</tr>
<tr>
<td>43</td>
<td>OUT 20</td>
<td>44</td>
<td>PAT 19</td>
<td>87</td>
</tr>
<tr>
<td>45</td>
<td>OUT 22</td>
<td>46</td>
<td>IN 21</td>
<td>89</td>
</tr>
<tr>
<td>47</td>
<td>OUT 22</td>
<td>48</td>
<td>PAT 21</td>
<td>91</td>
</tr>
<tr>
<td>49</td>
<td>OUT 24</td>
<td>50</td>
<td>IN 23</td>
<td>93</td>
</tr>
</tbody>
</table>
## Analog I/O Connector

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

MULTIBUS INTERFACE SWITCH SELECTION / MEMORY MAP
**Default Memory Map**

Dedicated I/O Memory:
- BANK 1: B:8080H -> B:87FFH
- BANK 2: B:8880H -> B:BFFFH

Control/Status-Register Address: 186H -> 181H

**Switch Numbering**

---Card Edge---

<table>
<thead>
<tr>
<th>A (34)</th>
<th>B (23)</th>
<th>C (12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>1 2 3 4 5 6 7 8</td>
<td>1 2 3 4 5 6 7 8</td>
</tr>
</tbody>
</table>

**Default Switch Settings**

<table>
<thead>
<tr>
<th>A (34)</th>
<th>B (23)</th>
<th>C (12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>1 1 1 1 1 1 1 1</td>
<td>0 0 0 1 0 0 0</td>
</tr>
</tbody>
</table>

0 = Closed
1 = Open
### Dedicated I/O Memory Base Address

<table>
<thead>
<tr>
<th>ADR</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>B4</td>
<td>Open for 16 address bits</td>
</tr>
<tr>
<td></td>
<td>Closed for 20 address bits</td>
</tr>
<tr>
<td>/ADR13 -&gt; C1</td>
<td>0 = Closed</td>
</tr>
<tr>
<td>/ADR12 -&gt; C2</td>
<td>1 = Open for /ADR</td>
</tr>
<tr>
<td>/ADR11 -&gt; C3</td>
<td></td>
</tr>
<tr>
<td>/ADR10 -&gt; C4</td>
<td></td>
</tr>
<tr>
<td>/ADR F -&gt; C5</td>
<td>0 = Open for ADR</td>
</tr>
<tr>
<td>/ADR E -&gt; C6</td>
<td>1 = Closed for ADR</td>
</tr>
<tr>
<td>/ADR D -&gt; C7</td>
<td></td>
</tr>
<tr>
<td>/ADR C -&gt; C8</td>
<td></td>
</tr>
</tbody>
</table>

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

### Control / Status-Register Address

<table>
<thead>
<tr>
<th>A1</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Open for 8 address bits</td>
</tr>
<tr>
<td></td>
<td>Closed for 12 address bits</td>
</tr>
<tr>
<td>/ADR B -&gt; B5</td>
<td>0 = Closed</td>
</tr>
<tr>
<td>/ADR A -&gt; B4</td>
<td>1 = Open for /ADR</td>
</tr>
<tr>
<td>/ADR 9 -&gt; B7</td>
<td></td>
</tr>
<tr>
<td>/ADR 8 -&gt; B6</td>
<td></td>
</tr>
<tr>
<td>/ADR 7 -&gt; A2</td>
<td>0 = Open for ADR</td>
</tr>
<tr>
<td>/ADR 6 -&gt; A3</td>
<td>1 = Closed for ADR</td>
</tr>
<tr>
<td>/ADR 5 -&gt; A4</td>
<td></td>
</tr>
<tr>
<td>/ADR 4 -&gt; A5</td>
<td></td>
</tr>
<tr>
<td>/ADR 3 -&gt; A6</td>
<td></td>
</tr>
<tr>
<td>/ADR 2 -&gt; A7</td>
<td></td>
</tr>
<tr>
<td>/ADR 1 -&gt; A8</td>
<td></td>
</tr>
</tbody>
</table>

/ADR is inverted ADR as it appears on the bus.
## Wait States Switch Selection

<table>
<thead>
<tr>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>Closed</td>
<td>Closed</td>
<td>1 Clock Cycle</td>
</tr>
<tr>
<td>Closed</td>
<td>Closed</td>
<td>Open</td>
<td>2 Clock Cycles</td>
</tr>
<tr>
<td>Closed</td>
<td>Open</td>
<td>Closed</td>
<td>3 Clock Cycles</td>
</tr>
<tr>
<td>Closed</td>
<td>Open</td>
<td>Open</td>
<td>4 Clock Cycles</td>
</tr>
<tr>
<td>Open</td>
<td>Closed</td>
<td>Closed</td>
<td>5 Clock Cycles</td>
</tr>
<tr>
<td>Open</td>
<td>Closed</td>
<td>Open</td>
<td>6 Clock Cycles</td>
</tr>
<tr>
<td>Open</td>
<td>Open</td>
<td>Closed</td>
<td>7 Clock Cycles</td>
</tr>
<tr>
<td>Open</td>
<td>Open</td>
<td>Open</td>
<td>8 Clock Cycles</td>
</tr>
</tbody>
</table>
APPENDIX E

CONTROL / STATUS-REGISTERS AND 1/0 MEMORY STRUCTURE
### CONTROL REGISTER

| Bit 15 | STONTRIG | 0 = Normal Start  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1 = Start on Trigger</th>
</tr>
</thead>
</table>
| Bit 14 | CLEAR | 0 = Clear  
|       |         | 1 = Go |
| Bit 13 | NORMLA | 0 = Normal Single Scan  
|       |          | 1 = Logic-Analyzer On |
| Bit 12 | EPOS | 0 = Disable Pos Edge  
|       |         | 1 = Enable Pos Edge |
| Bit 11 | ENED | 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 Clock 2  
|       |         | 1 = Disable Clock 2 |
| Bit 7-5 | C2,B2,A2 | 0,0,0 = Single Step  
|       |         | 1,0,1 = 50.25 MHz  
|       |         | 0,1,0 = 50.50 MHz  
|       |         | 0,1,1 = 61.00 MHz  
|       |         | 1,0,0 = 62.00 MHz  
|       |         | 1,0,1 = 64.00 MHz  
|       |         | 1,1,0 = 88.00 MHz  
|       |         | 1,1,1 = 16.00 MHz |
| Bit 4  | G1 | 0 = Enable Clock 1  
|       |         | 1 = Disable Clock 1 |
| Bit 3-1 | C1,B1,A1 | 0,0,0 = Single Step  
|       |         | 1,0,1 = 50.25 MHz  
|       |         | 0,1,0 = 50.50 MHz  
|       |         | 0,1,1 = 61.00 MHz  
|       |         | 1,0,0 = 62.00 MHz  
|       |         | 1,0,1 = 64.00 MHz  
|       |         | 1,1,0 = 88.00 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

<table>
<thead>
<tr>
<th>Tn = Tri-State Control Bit n</th>
<th>0 = Put Pin in Tri-State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Drive pin according to Un</td>
</tr>
<tr>
<td>Un = Drive Control Bit n</td>
<td>0 = Drive Pin Low</td>
</tr>
<tr>
<td></td>
<td>1 = Drive Pin High</td>
</tr>
</tbody>
</table>

**Speed = 8 (Non-Multiplexed Mode)**

At offset B80H (default 0:B80H)

<table>
<thead>
<tr>
<th>Low Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T4</th>
<th>V4</th>
<th>T3</th>
<th>V3</th>
<th>T2</th>
<th>V2</th>
<th>T1</th>
<th>V1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

At offset B82H (default 0:B82H)

<table>
<thead>
<tr>
<th>Low Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T12</th>
<th>U12</th>
<th>T11</th>
<th>V11</th>
<th>T10</th>
<th>U10</th>
<th>T9</th>
<th>V9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Speed = 1 (Multiplexed Mode)**

At offset B80H (default 0:B80H) and offset B82H (default 0:B82H)

<table>
<thead>
<tr>
<th>Low Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 2 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T4</th>
<th>V4</th>
<th>T3</th>
<th>V3</th>
<th>T2</th>
<th>V2</th>
<th>T1</th>
<th>U1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T8</th>
<th>V8</th>
<th>T7</th>
<th>V7</th>
<th>T6</th>
<th>V6</th>
<th>T5</th>
<th>V5</th>
</tr>
</thead>
</table>
**PATTERN-INPUT-MEMORY**

<table>
<thead>
<tr>
<th>$H_n = \text{Input High Bit } n$</th>
<th>$0 = \text{Below High Threshold}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$1 = \text{Above High Threshold}$</td>
</tr>
<tr>
<td>$L_n = \text{Input Low Bit } n$</td>
<td>$0 = \text{Below Low Threshold}$</td>
</tr>
<tr>
<td></td>
<td>$1 = \text{Above Low Threshold}$</td>
</tr>
</tbody>
</table>

**Speed = 0 (Non-Multiplexed Mode)**

At offset B800H (default 0:8000)

<table>
<thead>
<tr>
<th>Low Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>H4 L4 H3 L3 H2 L2 H1 L1</td>
<td>H8 L8 H7 L7 H6 L6 H5 L5</td>
</tr>
</tbody>
</table>

At offset B800H (default B:8000)

<table>
<thead>
<tr>
<th>Low Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

**Speed = 1 (Multiplexed Mode)**

At offset 8800H (default 0:8000 and offset B800H (default 0:8000)

<table>
<thead>
<tr>
<th>Low Address</th>
<th>High Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 6 5 4 3 2 1 0</td>
<td>7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>H4 L4 H3 L3 H2 L2 H1 L1</td>
<td>H8 L8 H7 L7 H6 L6 H5 L5</td>
</tr>
</tbody>
</table>
**TRIGGER-PATTERN**

<table>
<thead>
<tr>
<th>Hn = Pattern High Bit n</th>
<th>0 = Below High Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Above High Threshold</td>
</tr>
<tr>
<td>Ln = Pattern Low Bit n</td>
<td>0 = Below Low Threshold</td>
</tr>
<tr>
<td></td>
<td>1 = Above Low Threshold</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>H16 H15 H14 H13 H12 H11 H10 H9 H8 H7 H6 H5 H4 H3 H2 H1 L1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>H8 H7 H6 H5 H4 H3 H2 H1 L1 H9 H8 H7 H6 H5 H4 H3 H2 H1 L1</td>
</tr>
</tbody>
</table>
APPENDIX F

SERIAL COMMUNICATION BOARD CIRCUIT DIAGRAM
SERIAL COMMUNICATION BOARD

This card implements the serial link on the Apple II 4 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.

<table>
<thead>
<tr>
<th>Number</th>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7404</td>
<td>Hex Inverters</td>
</tr>
<tr>
<td>2</td>
<td>74LS04</td>
<td>Hex Inverters</td>
</tr>
<tr>
<td>3</td>
<td>74LS08</td>
<td>Quadruple 2-Input Positive AND Gates</td>
</tr>
<tr>
<td>4</td>
<td>74LS98</td>
<td>Decade Counter</td>
</tr>
<tr>
<td>5,6</td>
<td>74LS192</td>
<td>Presettable Binary Counter</td>
</tr>
<tr>
<td>7</td>
<td>74LS244</td>
<td>Octal Buffers / Tri-State Line Drivers</td>
</tr>
<tr>
<td>8</td>
<td>75LS245</td>
<td>Octal Bus Transceivers</td>
</tr>
<tr>
<td>9</td>
<td>74LS251</td>
<td>1 of 8 Data Selector / Multiplexer</td>
</tr>
<tr>
<td>10</td>
<td>MC6850</td>
<td>Asynchronous Communications Interface</td>
</tr>
<tr>
<td>11</td>
<td>MC1488</td>
<td>Dual RS 232 C Line Driver</td>
</tr>
<tr>
<td>12</td>
<td>HD1489</td>
<td>Dual RS 232 C Line Receiver</td>
</tr>
</tbody>
</table>
SERIAL COMMUNICATION BOARD
APPENDIX G

SERIAL COMMUNICATION BOARD CONTROL
Baud Rates (with /16 Clock)

<table>
<thead>
<tr>
<th>Switch 1</th>
<th>Switch 2</th>
<th>Switch 3</th>
<th>Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Open</td>
<td>Open</td>
<td>150 baud</td>
</tr>
<tr>
<td>Open</td>
<td>Open</td>
<td>Closed</td>
<td>300 baud</td>
</tr>
<tr>
<td>Open</td>
<td>Closed</td>
<td>Open</td>
<td>600 baud</td>
</tr>
<tr>
<td>Open</td>
<td>Closed</td>
<td>Closed</td>
<td>1200 baud</td>
</tr>
<tr>
<td>Closed</td>
<td>Open</td>
<td>Open</td>
<td>2400 baud</td>
</tr>
<tr>
<td>Closed</td>
<td>Open</td>
<td>Closed</td>
<td>4800 baud</td>
</tr>
<tr>
<td>Closed</td>
<td>Closed</td>
<td>Open</td>
<td>9600 baud</td>
</tr>
<tr>
<td>Closed</td>
<td>Closed</td>
<td>Closed</td>
<td>19200 baud</td>
</tr>
</tbody>
</table>

Parameter Switches

<table>
<thead>
<tr>
<th>Switch 4</th>
<th>Switch 5</th>
<th>Switch 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>Bit 1</td>
<td>Bit 2</td>
</tr>
<tr>
<td>Open</td>
<td>= Logic 1</td>
<td></td>
</tr>
<tr>
<td>Closed</td>
<td>= Logic 0</td>
<td></td>
</tr>
</tbody>
</table>

Interrupt Control Switch

<table>
<thead>
<tr>
<th>Switch 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
</tr>
<tr>
<td>Closed</td>
</tr>
</tbody>
</table>

RTS - CTS Control

<table>
<thead>
<tr>
<th>Switch 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
</tr>
<tr>
<td>Closed</td>
</tr>
</tbody>
</table>
Memory Map

<table>
<thead>
<tr>
<th>Data-Register</th>
<th>$C6n7H$ / $C8nFH$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control / Status-Register</td>
<td>$C6n6H$ / $C8nEH$</td>
</tr>
<tr>
<td>Parameter-Register</td>
<td>$C6n0H$ / $C8n1H$ / $C8n4H$ / $C8n5H$ / $C8n8H$ / $C8n9H$ / $C8nCH$ / $C8nDH$</td>
</tr>
</tbody>
</table>

$n = $ Slot Number $+ 8$

Parameter-Register Structure

<table>
<thead>
<tr>
<th>Bit Numbers</th>
<th>7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>0 1 0 1 0 56 55 54</td>
</tr>
</tbody>
</table>

Upper Bits Fixed and lower bits switch selectable.

Control-Register Programming Information

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

Status-Register Programming Information

<table>
<thead>
<tr>
<th>Bit 0</th>
<th>- Transmit Data Register Empty</th>
<th>0 = Not Empty</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 = Empty (Send next Char)</td>
<td></td>
</tr>
<tr>
<td>Bit 1</td>
<td>- Receive Data Register Full</td>
<td>0 = Not Full</td>
</tr>
<tr>
<td></td>
<td>1 = Full (Fetch Char)</td>
<td></td>
</tr>
</tbody>
</table>

For more information see the Motorola 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 6800 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 directly 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 Communication Utility Program

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD00</td>
<td>Cold Start Entry Point</td>
</tr>
<tr>
<td>CD03</td>
<td>Warm Start Entry Point</td>
</tr>
<tr>
<td>CD06</td>
<td>DOS Main Loop Re-entry Pt.</td>
</tr>
<tr>
<td>CD09</td>
<td>Input Character</td>
</tr>
<tr>
<td>CD0C</td>
<td>Input Character</td>
</tr>
<tr>
<td>CD0F</td>
<td>Output Character</td>
</tr>
<tr>
<td>CD12</td>
<td>Output Character</td>
</tr>
<tr>
<td>CD15</td>
<td>Get Character</td>
</tr>
<tr>
<td>CD18</td>
<td>Put Character</td>
</tr>
<tr>
<td>CD1B</td>
<td>Input into Line Buffer</td>
</tr>
<tr>
<td>CD1E</td>
<td>Print String</td>
</tr>
<tr>
<td>CD21</td>
<td>Classify Character</td>
</tr>
<tr>
<td>CD24</td>
<td>Print CR and LF</td>
</tr>
<tr>
<td>CD27</td>
<td>Get Next Buffer Character</td>
</tr>
<tr>
<td>CD2A</td>
<td>Restore I/O Vectors</td>
</tr>
<tr>
<td>CD2D</td>
<td>Get File Specification</td>
</tr>
<tr>
<td>CD30</td>
<td>File Loader</td>
</tr>
<tr>
<td>CD33</td>
<td>Set Extension</td>
</tr>
<tr>
<td>Line</td>
<td>ASSEMBLER Instruction</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>37</td>
<td>ADDXX EQU $CD36</td>
</tr>
<tr>
<td>38</td>
<td>OUTEE EQU $CD39</td>
</tr>
<tr>
<td>39</td>
<td>OUTHE EQU $CD3C</td>
</tr>
<tr>
<td>40</td>
<td>RTPERR EQU $CD3F</td>
</tr>
<tr>
<td>41</td>
<td>GETHE EQU $CD42</td>
</tr>
<tr>
<td>42</td>
<td>OUTADR EQU $CD45</td>
</tr>
<tr>
<td>43</td>
<td>INBEC EQU $CD48</td>
</tr>
<tr>
<td>44</td>
<td>DOCHND EQU $CD4B</td>
</tr>
<tr>
<td>45</td>
<td>STAT EQU $CD4E</td>
</tr>
<tr>
<td>57</td>
<td>D400 FMSINIT EQU $D400</td>
</tr>
<tr>
<td>58</td>
<td>D403 FMSCLOSE EQU $D403</td>
</tr>
<tr>
<td>59</td>
<td>D406 FMS EQU $D406</td>
</tr>
<tr>
<td>67</td>
<td>RXFF EQU $60</td>
</tr>
<tr>
<td>68</td>
<td>OPENR EQU $1</td>
</tr>
<tr>
<td>69</td>
<td>OPENW EQU $2</td>
</tr>
<tr>
<td>70</td>
<td>OPENU EQU $3</td>
</tr>
<tr>
<td>71</td>
<td>CLOSE EQU $4</td>
</tr>
<tr>
<td>72</td>
<td>REWIND EQU $5</td>
</tr>
<tr>
<td>73</td>
<td>OPDIR EQU $6</td>
</tr>
<tr>
<td>74</td>
<td>GETINF EQU $7</td>
</tr>
<tr>
<td>75</td>
<td>PUTINF EQU $8</td>
</tr>
<tr>
<td>76</td>
<td>GETSEC EQU $9</td>
</tr>
<tr>
<td>77</td>
<td>PUTSEC EQU $A</td>
</tr>
<tr>
<td>78</td>
<td>RES1 EQU $B</td>
</tr>
<tr>
<td>79</td>
<td>DELATE EQU $C</td>
</tr>
<tr>
<td>80</td>
<td>RENAME EQU $D</td>
</tr>
<tr>
<td>81</td>
<td>RES2 EQU $E</td>
</tr>
<tr>
<td>82</td>
<td>NEXSEQ EQU $F</td>
</tr>
<tr>
<td>83</td>
<td>OSTREC EQU $10</td>
</tr>
<tr>
<td>84</td>
<td>GETRN EQU $11</td>
</tr>
<tr>
<td>85</td>
<td>PUTRN EQU $12</td>
</tr>
<tr>
<td>86</td>
<td>RES3 EQU $13</td>
</tr>
<tr>
<td>87</td>
<td>NEXDRV EQU $14</td>
</tr>
<tr>
<td>88</td>
<td>REGON EQU $15</td>
</tr>
<tr>
<td>89</td>
<td>BASE2 EQU $16</td>
</tr>
<tr>
<td>100</td>
<td>FUNCN EQU $0</td>
</tr>
<tr>
<td>101</td>
<td>ERSTAT EQU $1</td>
</tr>
<tr>
<td>102</td>
<td>ACTVST EQU $2</td>
</tr>
<tr>
<td>103</td>
<td>DRVNUM EQU $3</td>
</tr>
<tr>
<td>104</td>
<td>FRAME EQU $4</td>
</tr>
<tr>
<td>105</td>
<td>FILEN EQU $12</td>
</tr>
<tr>
<td>106</td>
<td>FATTRB EQU $15</td>
</tr>
<tr>
<td>107</td>
<td>RES4 EQU $16</td>
</tr>
</tbody>
</table>
EQU Starting Track of File
BEGSEC EQU Starting Sector of File
ENUTRK EQU Ending Track of File
ENDSEC EQU Ending Sector of File
-22 File Size
FSECMI EQU File Sector Map Indicator
-27 File Creation Date
-46 Name Work Buffer
-49 Current Dir. Add
-52 First Del Dir Ptr.
-63 Scratch Bytes
-30 Conlbk Lib Flag
-319 Sector Buffer (256)

* Special Values

CC2B MEMEND EQU $CC2B End User Mem For Flex
CC16 ESCRIG EQU $CC16 Escape Return Register
OC60 DELAY EQU $0060 Delay Parameter
OC13 CTRL EQU $13 Wait Character

* File Control Blocks

C100 FCB EQU $C100 Utility Area
C240 FCB EQU $C240 Utility Area
C380 FCB EQU $C380 Utility Area

* Macro Definitions

SUBIN MACRO
PSHS A,B,X,Y,CC
ENIN

SUBQUE MACRO
PULL A,B,X,Y,CC,PC

* Start of Program

0000 DR. $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
Serial Communication Utility 3-1-84 TSC Assembly PAGE 3

111 0011 BEGTRK EQU 17 Starting Track of File
112 0012 BEGSEC EQU 18 Starting Sector of File
113 0013 ENDTTRK EQU 19 Ending Track of File
114 0014 ENDTSEC EQU 20 Ending Sector of File
115 0015 FLSIZE EQU 21 ~22 File Size
116 0017 FILECHI EQU 23 File Sector Map Indicator
117 0018 RES55 EQU 24 Reserved for Future
118 0019 DCFDATE EQU 25 ~27 File Creation Date
119 0016 FCBLST EQU 28 ~29 FCB List Pointer
120 0012 TRACK EQU 30 Current Track
121 0017 SECTOR EQU 31 Current Sector
122 0020 RECORD EQU 32 ~33 Current Record No.
123 0021 DAINDX EQU 34 Data Index
124 0023 RMNDAX EQU 35 Random Index
125 0024 RAMERS EQU 36 ~46 Name Work Buffer
126 002F DIAAUD EQU 47 ~49 Current Dir. Add
127 0032 FDELPD EQU 50 ~52 First Del Dir Ptr.
128 0033 SCRITCH EQU 53 ~63 Scratch Bytes
129 003B COWRHS EQU 59 Space Compression Flag
130 0010 BUFFER EQU 64 ~319 Sector Buffer (256)
131

* Special Values

133 0028 CC2B EQU $CC2B End User Mem For Flex
135 0018 ESRIRG EQU $CC16 Escape Return Register
136 006D DELAY EQU $0060 Delay Parameter
137 0013 CTRLS EQU $13 Wait Character

138

* File Control Blocks

140

* ————————————

141 0C10 FCB1 EQU $0C10 Utility Area
142 0C20 FCB2 EQU $0C24 Utility Area
143 0C30 FCB2 EQU $0C38 Utility Area
144

* ————

145 0000 SUBIN MACRO
148 PSHS A,B,X,Y,CC
149 ENCM
150

151 0000 SUBOUT MACRO
152 PULS A,B,X,Y,CC,PC
153 ENCM
154

* ————

155 0000 ORG $0000
156

157 0000 40 01 TERM BRA TERM1 Get Around Terms
158 0002 01 VN FCB 1 Version Number
159

160 0003 6D 09 TERM1 BSR INIT Initialise
161 *

162 *

163 *

164 *

* Main Loop
Serial Communication Utility

3-1-84 TSC ASSEMBLER PAGE 4

166 0005 BD 41 LOOP BSR KBDCRK Check Keyboard
167 0007 17 0354 LABSR FILCHK Check For File Input
168 000A BD 7B BSR URTCHK Check Uart
169 000C 20 F7 BRA LOOP Loop Always

170 *
171 *
172 Initialize Subroutine

174 000E INIT SUBIN Subroutine Beginning
175 0010 7F 0475 CLR HALFLUP Clear Half Dup
176 0013 7F 0476 CLR NESTRL Clear No BS Trl
177 0016 7F 0477 CLR F1FLAG Clear File 1 Flag
178 0019 7F 0478 CLR F2FLAG Clear File 2 Flag
179 001C 7F 0479 CLR FOUTFLAG Clear File Out Flag
180 001F 7F 047A CLR WAT Clear Wait Flag

181 0022 CC 0060 LDD #DELAY Initial Delay
182 0025 FD 0472 STD SPEED Store in Speed
183 0028 7F 0474 CLR F1FL Type

184 *
185 0030 CC 04A4 LDD #ENDPRO End of Program
186 0032 FD 047D STD BUFSP Start of Buffer
187 0031 FD 0481 STD PTR File Pointer
188 0034 PC CC28 LDD NMEM End of Memory
189 0037 FD 047F STD BUFLP End of Buffer

190 *
191 003A CC 043A LDD #EXIT Patch Escape Ret Reg
192 003D FD CC16 STD BSRTLG Store Address
193 *
194 0040 17 010B LSR #SETST Set Status Of Utility
195 0043 17 0A55 LSR SUBIN Initialize Uart
196 0046 *
197 *
198 Check Keyboard Subroutine

199 *
200 004B KBDCRK SUBIN Subroutine Beginning
201 000A BD CD4E JSR LAT Check Keyboard Status
202 004D 27 26 BNE KBEREXT Exit If No Char
203 *
204 004F 17 0A79 LSR KEYIN Get Char No Echo
205 *
206 0052 81 1B CMPA #ESC Check For Escape
207 0054 26 05 BNE NSSC Not Escape
208 0056 17 00F5 LSR #SETST Set Status
209 0059 20 1A BCA KBEEXT Exit From Subroutine
210 *
211 005B F6 0476 NESC LDB NBSTRL Check If Translate BS
212 005F 5D TSET
213 005F 26 06 BNE NCTQ If Not Then Don't
214 *
215 0061 21 08 CMPA #8 Check for Back Space
216 0063 26 02 BNE NCTQ Not Back Space
217 0065 66 7F LDA $877 Change to Del
218
219 0067 81 13   NCTQ  CMPA  #CTRLS  Check For Ctrl-S
220 0069 26 05   BNE  NCTS  Not Ctrl-S
221 006B 73 047A  COM  WATF  Compliment Wait Flag
222 006E 20 05   BRA  XPURKE  Consume Character
223 * 0070 17 02B7  NCTS  LSRR  LOGIN  Write Character
224 0073 BD 02   BSR  CLOUT  Character Output
225 * 0075  KBDEXT  SUBOUT  Subroutine End
226 * 0077  CHOUT  SUBIN  Subroutine Beginning
227 0079 F6 0475  LDB  HALFDUP  Check Duplex
228 007C 5D 0071  TSTB  SETF  Set Flags
229 007D 27 03   BEQ  CHKCHR  No Echo Full Duplex
230 007F BD 0D10  JSR  PUTCHR  Console Output
231 * 0082 17 0A33  CHKCHR  LSRR  UOUT  Output to Uart
232 * 0085  SUBOUT  Subroutine End
233 * 0087  UERCHR  SUBIN  Subroutine Beginning
234 0089 17 0A1E  LSRR  USTAT  Check Uart Status
235 008C 27 16   BEEQ  URTEXT  Exit If no Character
236 008E 17 0A23  LSRR  UIN  Get Uart Character
237 0091 7D 0474  TST  FICHTF  Check File Type
238 0094 26 02   BNE  NSTRIF  Do Not Strip
239 0096 84 7F   ANDA  #$3F  Strip Most Significant Bit
240 0098 17 086B  NSTRIP  LSRR  FIFO  Write Character
241 009B BD 0D18  JSR  PUTCHR  Console Output
242 009E CC 0000  LDLR  #0  Set Counter to 0
243 00A1 FD 0485  STL  DELCTR  Store in Delay Ctr
244 * 00A4  URTEXT  SUBOUT  Subroutine End
245 * 00A6  PRINT  Free Memory
246 * 00A9  SUBIN  Subroutine Beginning
247 00AB  BE  04EF  LDX  #FREEEM  Free Memory Message
248 00AD BD 0D1E  JSR  PSTRNG  Print
249 00AE FC 047F  LDLR  BUFEP  Buffer End
250 00B1 B3 0C81  SUBD  FFD  Get Free Memory
251 00B4 6D 1E   BSR  PRINTD  Print D
252 * 00B6  LDX  #USEEM  Used Memory Message
253 00B9 BD 0D1E  JSR  PSTRNG  Print
254 00BC FC 0B81  LDLR  PPTE  Get Pointer
255 00BF B3 047D  SUBD  BUFSP  Get Used Memory
256 00C2 6D 10   BSR  PRINTD  Print D
257 * 00C4  BE  054B  LDX  #ALLMK  Total Memory Message
Seria1 Communication Utility 3-1-84 TSC ASSEMBLER PAGE 6

273 00C7 BD CD1E  JSR  FSTRNG  Print
274 00CA FC 047F  LDD  BUFSZ  End Buffer
275 00CD BD 047D  SUBD  BUFSZ  Get Buffer Size
276 00D0 BD 02  BSR  PRINTD  Print D

277 00D2

*  SUBOUT Subroutine End

279

*  Print D

281

*  PRINTD SUBIN Subroutine Beginning
283 00D6 FD 0487  STD  NUMSTR  Store Number
284 00D9 ED 0487  LDX  #NUMSTR  Point to Number
285 00DC BD CD45  JSR  OUTADR  Print Number

00DF

SUBOUT Subroutine End

287

*  Print File Name Subroutine

289

*  ==========================

00E1

PPNAME SUBIN Subroutine Beginning
291 00E3 10BE 0721  LDY  #SPACER  Point To Spacer
292 00E7 C6 03  LDB  #5  Length of Spacer
293 00E9 ED 15  BSR  PPNAME  Print Spacer
294 00EB AD 03  LDA  DRVNUM,X  Get Drive Number
295 00ED BD CD45  ADDA  #30  Change to Character
296 00EF BD CD1E  JSR  PUTCHR  Output Character
297 00F2 31 04  LEAY  FNAME,X  Point to File Name
298 00F4 C6 06  LDB  #6  Number of Characters
299 00F6 BD 0E  BSR  FPNAME  Print Name
300 00F8 31 0C  LEAY  FEXTEN,X  Point to File Extension
301 00FA C6 03  LDB  #3  Number of Characters
302 00FC BD 08  BSR  FPNAME  Print Name

00FE

SUBOUT Subroutine End

304

*  Print Name

306

*  ==========

0100 0102 1F 21  TFR  Y,X  Put Y -> X
309 0104 20 09  BRA  PLLO0P  Entry No ' '

310

*  PLLOOP

312 0106 1F 21  TFR  Y,X  Put Y -> Y
313 0108 BD 22  LDA  #',  Print '
314 010C BD CD1E  JSR  PUTCHR  Output Character
315 010F AD 80  PKLOOP  LDA ,X+  Get Character
316 0111 BD CD1E  JSR  PUTCHR  Output Character
317 0114 5A  BCD  Decrement Counter
318 0115 26 F8  BNE  PKLOOP  Loop Until Last Char

319

*  SUBOUT Subroutine End

321

*  Print Speed Subroutine

323

*  ==========

0119 011B 8E 08C0  LDX  #SPEED  Speed Message
325 011E BD CD1E  JSR  FSTRNG  Print
Serial Communication Utility

3-1-84 TSC ASSEMBLER PAGE 7

0472 tfSPEED Point to Speed
0124 BD CD45 Output Number
0127 66 20 Print Blank
0129 BD CD18 Print Bracket
012C 66 29
012E BD CD18
0131 SUBOUT Subroutine End

* Set Speed

013F tfSPEED SUBIN Subroutine Beginning
0135 BE 074F SSDAGN LDX #$FED Speed Message
0138 BD CD1E JBR PSTRING Print
0139 BD CD1B JBR INBUFF Input Buffer
013E BD CD42 JBR GETHEX Get Hex Number
0141 25 F2 BCS SSDAGN Try Again
0143 5D TSTB Is There a No
0144 26 BNE ISA0 There is a no
0146 BE LDX #$DELAY Default for return
0149 BE ISA0 STX SPEED Store Speed
014C SUBOUT Subroutine End

* Set Status Subroutine

015E SETSTT SUBIN Subroutine Beginning
0150 8E 0469 SSDAGN LDX #$ATH1 Status Message
0153 BD CD1E JBR PSTRING Print
0155 17 FF40 LBXR PRTFEE Print Free Space
0158 BD 579 LUX # ATH2 Status Message
015C BD CD1E JBR PSTRING Print
015F BD 0475 LDA HALFUP Look at Duplex Flag
0162 4D TSTA Set Flags
0163 26 03 BNE SHFDUP Half Duplex Jump
0165 8E 0776 LDX #FDUP Full Duplex Message
0168 20 03 BRA BSTCK Now Check Backspace
016A BE 07AC SHFDUP LDX #FDUP Half Duplex Message
016D BD CD1E BSTCK JBR PSTRING Print
0170 8E 0476 LDA NSHRCL Look at Backspace Flag
0173 4D TSTA Set Flags
0174 26 05 BNE SNSST No Backspace T Jump
0176 8E 07B4 LDX #$BSC Backspace Trans Mask
0179 20 03 BRA LOGPS Now Print Log State
017B 8E 081B NSBSC LDX #BNT No Backspace Trans
017E BD CD1E LOGPS JBR PSTRING Print
0181 BD 047A LDA FILTYP Check File Type
0184 4D TSTA Set Flags
0185 26 05 BNE NSCOM No Space Compress
0187 8E 0889 LDX #$SPCM Has Space Compress
018A 20 03 BRA SPDCHK Speed Check
018C 8E 0852 NSC0N LDX #$SPCM No Space Compress
018F BD CD1E SPDCHK JBR PSTRING Print
0192 BD 85 BSBS PFEED Print Speed
0194 8E 0477 LDA FIFLAGS Log File Flag
0197 4D TSTA Set Flags
381 0196 26 08  BNE  F10  File 1 Open Jump
382 019F 8E 6F1  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  F10  LDX  #F1OM  File 1 Open Mess
386 01A5 BD CD1E  JSR  PSTRNG  Print
387 01A8 8E C100  LDX  #FC81  Get FCB Address
388 01AB 17 F233  LDSR  PFFNAME  "Print File Name"
389 01AE B6 0476  F2PS  LD A  FPFLAG  Storage File Flag
390 01B1 4D  TSTA  Set Flags
391 01B2 26 08  BNE  F20  File 2 Open Jump
392 01B4 8E 095F  LDX  #F2CM  File 2 Closed Mess
393 01B7 BD CD1E  JSR  PSTRNG  Print
394 01B9 20 0C  BRA  FOPS  File Out Print Mess
395 01BC 8E 0996  F20  LDX  #F2OH  File 2 Open Mess
396 01BE BD CD1E  JSR  PSTRNG  Print
397 01C2 8E C240  LD X  #FC82  Get FCB Address
398 01C5 1F FF19  LDSR  PFFNAME  "Print File Name"
399 01C8 B6 0479  FOPS  LD A  FPFLAGS  Output File Flag
400 01C9 4D  TSTA  Set Flags
401 01CC 26 08  BNE  F00  File Output Open Jump
402 01CD 8E 09CD  LDX  #F0CM  File Output Closed Mess
403 01D1 BD CD1E  JSR  PSTRNG  Print
404 01D4 20 0C  BRA  NOWSS  Now Set Status
405 01D6 8E 0A04  F00  LDX  #F0OH  File Output Open Mess
406 01D9 BD CD1E  JSR  PSTRNG  Print
407 01DC 8E C380  LDX  #FCEO  Get FCB Address
408 01DF 17 F3FF  LDSR  PFFNAME  "Print File Name"
409 01E0 8E 0A38  NOWSS  LD X  #MENU  Menu Message
410 01E3 BD CD1E  JSR  PSTRNG  Print
411 01E8 BD CD15  JSR  GEICHR  Input Character
412 01E9 26 0E  BNE  TR2  Try 2
413 01EF 5C 01  CHF  #13  Carriage Ret
414 01ED 27 3F  BLO  SEXIT  Subroutine Exit
415 01EE 81 31  TR1  CHF  #1  Subroutine Exit
416 01F1 26 06  BNE  TR2  Try 2
417 01F3 73 0475  COM  HALFWP  Invert Duplex
418 01F6 16 FP57  LIBRA  SSAGN  Do Again
419 01F9 81 32  TR2  CHF  #12
420 01FB 26 06  BNE  TR3  Try 3
421 01FD 73 0476  COM  NBSTPL  Invert BS Translation
422 0200 16 FP40  LIBRA  SSAGN  Do Again
423 0203 81 33  TR3  CHF  #13
424 0205 26 06  BNE  TR4  Try 4
425 0207 73 0474  COM  FLTTYP  Invert File Type
426 020A 16 FP43  LIBRA  SSAGN  Do Again
427 020C 81 34  TR4  CHF  #14
428 020E 26 06  BNE  TR  Try 5
429 0210 77 0476  COM  NBSTPL  Invert BS Translation
430 0212 17 FP1F  LIBSR  SSPEED  Set Speed
431 0214 16 FP39  LIBRA  SSAGN  Do Again
432 0217 81 35  TR5  CHF  #15
433 0219 26 05  BNE  TR6  Try 6
434 021B BD 39  BSR  FUN5  Function J - Log File
<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>435</td>
<td>021D 16</td>
<td>FF30&lt;br&gt;LBRA SSAGN</td>
<td>Do Again</td>
</tr>
<tr>
<td>436</td>
<td>0220 81</td>
<td>36&lt;br&gt;TR6&lt;br&gt;CMPA 8'6</td>
<td>Try 7</td>
</tr>
<tr>
<td>437</td>
<td>0222 26</td>
<td>05&lt;br&gt;BNE TR7</td>
<td>Try 8</td>
</tr>
<tr>
<td>438</td>
<td>0224 8D</td>
<td>43&lt;br&gt;BSR FUn6&lt;br&gt;Function 6 - Storage File</td>
<td></td>
</tr>
<tr>
<td>439</td>
<td>0226 16</td>
<td>FF27&lt;br&gt;LBRA SSAGN</td>
<td>Do Again</td>
</tr>
<tr>
<td>440</td>
<td>0228 81</td>
<td>37&lt;br&gt;TR7&lt;br&gt;CMPA 8'7</td>
<td>Try 9</td>
</tr>
<tr>
<td>441</td>
<td>022A 26</td>
<td>05&lt;br&gt;BNE TR8</td>
<td>Try 10</td>
</tr>
<tr>
<td>442</td>
<td>022C 8D</td>
<td>53&lt;br&gt;BSR FUn7&lt;br&gt;Function 7 - Output File</td>
<td></td>
</tr>
<tr>
<td>443</td>
<td>022E 16</td>
<td>FF1E&lt;br&gt;LBRA SSAGN</td>
<td>Do Again</td>
</tr>
<tr>
<td>444</td>
<td>0230 81</td>
<td>36&lt;br&gt;TR8&lt;br&gt;CMPA 8'8</td>
<td>Function 8 - Exit</td>
</tr>
<tr>
<td>445</td>
<td>0232 26</td>
<td>03&lt;br&gt;BNE TR9</td>
<td>Character Error</td>
</tr>
<tr>
<td>446</td>
<td>0234 16</td>
<td>0201&lt;br&gt;LBRA EXIT</td>
<td>Function 9 - Abort</td>
</tr>
<tr>
<td>447</td>
<td>0236 81</td>
<td>39&lt;br&gt;TR9&lt;br&gt;CMPA 8'9</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>448</td>
<td>0238 26</td>
<td>03&lt;br&gt;BNE CERRR</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>449</td>
<td>023A 16</td>
<td>0216&lt;br&gt;LBRA ABORT</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>450</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>451</td>
<td>0240 8E</td>
<td>07&lt;br&gt;CERRR&lt;br&gt;LDA 007</td>
<td>Beep</td>
</tr>
<tr>
<td>452</td>
<td>0242 8B</td>
<td>CD18&lt;br&gt;JSR PUTCCHR</td>
<td>Output</td>
</tr>
<tr>
<td>453</td>
<td>0244 8D</td>
<td>CD18&lt;br&gt;JSR PUTCCHR</td>
<td>Output</td>
</tr>
<tr>
<td>454</td>
<td>0246 8D</td>
<td>CD18&lt;br&gt;JSR PUTCCHR</td>
<td>Output</td>
</tr>
<tr>
<td>455</td>
<td>0248 16</td>
<td>FF02&lt;br&gt;LBRA SSAGN</td>
<td>Do Again</td>
</tr>
<tr>
<td>456</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>457</td>
<td>024A 8E</td>
<td>0635&lt;br&gt;SEXIT&lt;br&gt;LDA #007</td>
<td>Return Message</td>
</tr>
<tr>
<td>458</td>
<td>0251 BD</td>
<td>CD1E&lt;br&gt;JSR PSTENG</td>
<td>Print</td>
</tr>
<tr>
<td>459</td>
<td>0253 4F</td>
<td></td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>460</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>461</td>
<td></td>
<td>* Open / Close Log File</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>462</td>
<td></td>
<td># Subroutine Beginning</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>463</td>
<td>025A BD</td>
<td>0477&lt;br&gt;LDA F1FLAG</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>464</td>
<td>025C 4D</td>
<td># TSTA</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>465</td>
<td>025E 26</td>
<td>04&lt;br&gt;BNE FUn5B</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>466</td>
<td>0260 8D</td>
<td>64&lt;br&gt;PBR OPENF</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>467</td>
<td>0262 20</td>
<td>03&lt;br&gt;BRA FUn5C</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>468</td>
<td></td>
<td></td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>469</td>
<td></td>
<td></td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>470</td>
<td></td>
<td></td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>471</td>
<td>0264 17</td>
<td>0172&lt;br&gt;FUn3B&lt;br&gt;LBR CLoUT</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>472</td>
<td>0266 17</td>
<td>0172&lt;br&gt;FUn3E&lt;br&gt;SUBOUT Subroutine End</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>473</td>
<td></td>
<td></td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>474</td>
<td></td>
<td>* Open / Close Storage File</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>475</td>
<td></td>
<td># Subroutine Beginning</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>476</td>
<td>026A 8D</td>
<td>39&lt;br&gt;BSR F1FILE2</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>477</td>
<td>026C 16</td>
<td>0478&lt;br&gt;LDA F2FLAG</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>478</td>
<td>0270 4D</td>
<td># TSTA</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>479</td>
<td>0271 26</td>
<td>04&lt;br&gt;BNE FUn6B</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>480</td>
<td>0273 8D</td>
<td>51&lt;br&gt;BSR OPENF</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>481</td>
<td>0275 2D</td>
<td>03&lt;br&gt;BRA FUn6E</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>482</td>
<td></td>
<td></td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>483</td>
<td></td>
<td></td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>484</td>
<td>0277 17</td>
<td>0083&lt;br&gt;FUn6B&lt;br&gt;LBR CLoSEF</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>485</td>
<td>027A FC</td>
<td>0470&lt;br&gt;FUn6C&lt;br&gt;LAD BUFSP</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>486</td>
<td>027D 4D</td>
<td>04d1&lt;br&gt;STD PFTX</td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>487</td>
<td>0280 8D</td>
<td></td>
<td>SUBOUT Subroutine End</td>
</tr>
<tr>
<td>488</td>
<td></td>
<td></td>
<td>SUBOUT Subroutine End</td>
</tr>
</tbody>
</table>
489
490
0282
0284 BD 30
0286 B6 0479
0289 4D
029A 26 03
029C 17 0112
029F 20 03

499
0291 17 0145

501
502
503
0296
0298 CC C100
029B FD 0478
029E CC 0477
02A1 FD 0483

507
508

510
511
512
02A4
02AC CC C240
02AD FD 047B
02AE CC 0478
02B1 FD 0483

514
515
516
517

519
520
521
02B6
02BA CC C380
02BB FD 047D
02BC CC 0479
02C1 FD 0483

523
524
525
526

529
530

02C6
02CB BD C11E
02CD BD C61B
02D1 BE 047B
02DA BD C02D

531
532
533
534
535
536
537
538
539
540
541
542

02CB 8E 0726
02CB BD C11D
02DE BD C61B
02DF BD C02D
02D7 LO5 014A
02D8 86 01
02DD BD C333
02DE 8E 00
02E2 A7 64
02E4 BD D406

535
541
542

543
544
545
546
548
550

02CB BD C11E
02CD BD C61B
02DE BD C02D
02D7 LO5 014A
02D8 86 01
02DD BD C333

549
550
551
552
553
554
555
556
557
558
559
560
561
562

531
532
533
534
535
536
537
538
539
540
541
542

* Open / Close Output File
* Subroutine Beginning

FILE1 SUBIN Subroutine Beginning
LDO #FCB1 Store FCB
STD FCB
LDD #F1FLAG File 1 Flag
STD FFLAG
SUBOUT Subroutine End

FILE2 SUBIN Subroutine Beginning
LDO #FCB2 Store FCB
STD FCB
LDD #F2FLAG File 2 Flag
STD FFLAG
SUBOUT Subroutine End

FILE3 SUBIN Subroutine Beginning
LDO #F3CB Store FCB
STD FCB
LDD #F3FLAG File 3 Flag
STD FFLAG
SUBOUT Subroutine End

* Open Relevant File
* Subroutine Beginning

OPENF SUBIN Subroutine Beginning
LUX #FNAME File Name Message
JSR PSTRING Print
JSR INDUFF Read Into Buffer
LDX FCB Get FCB Address
JSR GETFIL Get File Spec
LACS ERROR Any Errors
LDA #1 Set to TXT Default
JSR SETEXT Set Extension
LDA #OPENWR Open for Write
STA FUNCYN, X Store Function
JSR FNS Open for Write
졌다

LNEZ ERROR Any Errors

LDA #RWR Read/Write

STA FUNKTX Save in PCB

LDA FILL Space Compression

STA COMPSX Store

LDA #1 Open File Flag

STA [FFLAG] Store in Flag Place

SUBOUT Subroutine End

* Close Storage File

* ---------------------

CLOSEX SUBIN Subroutine Beginning

LDA F2FLAG Check If File Open

TSTA Set Flags

BEQ CLGEEXT Exit If Not Open

LDX BUFSP Point To Beginning

CMPX FFCR See If Reached End

BEQ CLCLOSE Close File If Case

LDA KX+ Get Character

PSHS X Store X

LDA #FCBZ Point to PCB

JSR FNS Write Character

LBNL ERROR Any Errors

PSLS X Store X

BRA CLFLP Loop

CLSFILE BSR FILE2 Set File 2

LBSR CLOUT Close File

* Write To Storage File

* ---------------------

FIFILE SUBIN Subroutine Beginning

LDB F2FLAG Check If File Open

TSTB Set Flags

BQ PIEXIT Exit If Not Open

LDB FTR Get Pointer Value

STA 0 X Store Character

LEAX #X Increment Pointer

STX FTER Store New Value

CMPX BUFSP See If Reached End

BNC FILEX Exit If Not Equal

LDB #OVSAWM Overflow Save Mess

JSR SRING Print

BSR CLOSEX Close File

LBSR OVERR Overflow Error

PIEXIT SUBOUT Subroutine End

* Write To Log File

* ---------------------
SUBROUTINE Beginning

See If File Open

BEQ Exit If Closed

LDX #FCB1 Point To FCB

JSR FMS Write Char

LDA ERROR Any Errors?

* LOGOUT Subroutine End

* Check For File Output

* SUBROUTINE Beginning

LDA FOFLAG Check If Open

TSTA Set Flags

BEQ FILEX Exit If File Not Open

LDA WAIT Check Wait Flag

TSTA Set Flags

BNE FILEX Exit in Wait Mode

LDX DELCTR Get Counter

BEQ FILEX Exit If Not Equal

GETCH LDAX #FCB0 Output From File

JSR FMS Get Character

BNE ENDFIL Check If File End

LDR #0 Set Counter to Zero

STD DELCTR Store in DELCTR

BRA FILEX Exit After Getting Char

ENDFIL LDA ERSTAT,X Get Error Status

CMPLA #0 Is it E0F Error

BEQ NORMCLS Normal Close

CLR FOFLAG Set File Closed Flag

LDBA ERROR Error Exit

NORCLS LSR FILEX Set File

BSR CMBUT Close Output File

FILEX SUBROUTINE Beginning

* Open Output File

* SUBROUTINE Beginning

LDX #FCB0 File Name Message

JSR PSTRING Print

JSR THBUFF Read Into Buffer

LDX #FCB0 Get FCB Address
**Serial Communication Utility**

3-1-84 TSC ASSEMBLER PAGE 13

---

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>03AF BD</td>
<td>CD2D</td>
<td>JSR GETFIL</td>
</tr>
<tr>
<td>03B2 25</td>
<td>71</td>
<td>BCS ERROR</td>
</tr>
<tr>
<td>03B4 86</td>
<td>01</td>
<td>LDA $1</td>
</tr>
<tr>
<td>03B6 8D</td>
<td>CD33</td>
<td>JSR SETEXT</td>
</tr>
<tr>
<td>03B9 86</td>
<td>01</td>
<td>LDA OPENRD</td>
</tr>
<tr>
<td>03BB A7</td>
<td>84</td>
<td>STA FUNCTN,X</td>
</tr>
<tr>
<td>03BD BE</td>
<td>D406</td>
<td>JSR FNS</td>
</tr>
<tr>
<td>03C0 26</td>
<td>63</td>
<td>BNE ERROR</td>
</tr>
<tr>
<td>03C2 86</td>
<td>00</td>
<td>LDA #RDWR</td>
</tr>
<tr>
<td>03C4 A7</td>
<td>84</td>
<td>STA FUNCTN,X</td>
</tr>
<tr>
<td>03C6 86</td>
<td>0474</td>
<td>LDA FLYTP</td>
</tr>
<tr>
<td>03C9 A7</td>
<td>88 3B</td>
<td>STA COMPRES,X</td>
</tr>
<tr>
<td>03CC 86</td>
<td>01</td>
<td>LDA $1</td>
</tr>
<tr>
<td>03CE B4</td>
<td>0479</td>
<td>STA FOPLAG</td>
</tr>
<tr>
<td>03D1 CC</td>
<td>0000</td>
<td>LDD $0</td>
</tr>
<tr>
<td>03D4 FD</td>
<td>0485</td>
<td>STD DELCTR</td>
</tr>
</tbody>
</table>

* SUBOUT Subroutine End

---

* Close Relevant File

---

CLOUT SUBSUB Subroutine Beginning

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>03BB A6</td>
<td>9F 0483</td>
<td>LDA [PFLAG]</td>
</tr>
<tr>
<td>03BF 4D</td>
<td></td>
<td>TSTA</td>
</tr>
<tr>
<td>03BE 27</td>
<td>10</td>
<td>BEQ CIDTEX</td>
</tr>
<tr>
<td>03BE 6F</td>
<td>9F 0483</td>
<td>CLR [PFLAG]</td>
</tr>
<tr>
<td>03EB BE</td>
<td>047B</td>
<td>LDX FCB</td>
</tr>
<tr>
<td>03EB 86</td>
<td>04</td>
<td>LDA $CLOSE</td>
</tr>
<tr>
<td>03EB A7</td>
<td>84</td>
<td>STA FUNCTN,X</td>
</tr>
<tr>
<td>03ED BD</td>
<td>D406</td>
<td>JSR FNS</td>
</tr>
<tr>
<td>03F0 26</td>
<td>33</td>
<td>BNE ERROR</td>
</tr>
</tbody>
</table>

* CIDTEX SUBOUT Subroutine End

---

* Purge Relevant File

---

FURGE SUBSUB Subroutine Beginning

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>03FE A6</td>
<td>9F 0483</td>
<td>LDA [PFLAG]</td>
</tr>
<tr>
<td>03FA 4D</td>
<td></td>
<td>TSTA</td>
</tr>
<tr>
<td>03FB 27</td>
<td>19</td>
<td>BEQ FUREXT</td>
</tr>
<tr>
<td>03FD 5F</td>
<td>9F 0483</td>
<td>CLR [PFLAG]</td>
</tr>
<tr>
<td>0401 8X</td>
<td>0473</td>
<td>LDX FCB</td>
</tr>
<tr>
<td>0404 86</td>
<td>04</td>
<td>LDA $CLOSE</td>
</tr>
<tr>
<td>0406 A7</td>
<td>84</td>
<td>STA FUNCTN,X</td>
</tr>
<tr>
<td>0408 BD</td>
<td>D406</td>
<td>JSR FNS</td>
</tr>
<tr>
<td>040B 26</td>
<td>18</td>
<td>BNE ERROR</td>
</tr>
<tr>
<td>040D 86</td>
<td>0C</td>
<td>LDA $DELETE</td>
</tr>
<tr>
<td>040F A7</td>
<td>84</td>
<td>STA FUNCTN,X</td>
</tr>
<tr>
<td>0411 BD</td>
<td>D406</td>
<td>JSR FNS</td>
</tr>
<tr>
<td>0414 26</td>
<td>0F</td>
<td>BNE ERROR</td>
</tr>
</tbody>
</table>

* FUREXT SUBOUT Subroutine End

---

* Overflow Error
SERIAL COMMUNICATION UTILITY 3-1-84 TSC ASSEMBLER PAGE 14

705 0418 OVEROR SUBIN Subroutine Beginning
707 041A BE 06A1 LDX #Mess Overflow Error
708 041D BD CD1E JSR PSTENG Print
709 0420 BD CD15 JSR GECHR Wait for Char
710 0423 SUBOUT Subroutine End

711 * Error Exit From Program

714 0425 BD CD3F ERROR JSR RTPEMR Report Error
715 0428 BE 0667 LDX #MESS Error Message
716 042B BD CD1E JSR PSTENG Print
717 042E BD 04 PLA ACLOSE Close File Function
718 0430 A7 84 STA FUNCTIN,X Store Function
719 0432 BD DA06 JSR RMS Do Closure
720 0435 BD CD15 JSR GECHR Wait for Char
721 0438 SUBOUT Subroutine End

722 * Exit Subroutine

725 043A 17 FE59 EXIT LSR FILE1 Close File 1
726 043D 8D 9A BSR CLOUT
727 043F 17 FE64 LSR FILE2 Close File 2
728 0442 17 FE68 LSR CLOSEP Close Output File
729 0445 17 FE6E LSR FILE3 Output File
730 0448 BD 8F BSR CLOUT Close Output File
731 044A BD DA03 JSR FMSCLS Close All Files
732 044D 8E 05D7 LDX #MESS Finish Message
733 0450 BD CD1E JSR PSTENG Print
734 0453 7E CD03 JMP WARNS Warm Start

735 * Abort Exit

736 0456 17 FE3D ABORT LSR FILE1 Purge File 1
737 0459 8D 99 BSR PURGE
738 045B 17 FE48 LSR FILE2 Purge File 2
739 045E BD 94 BSR PURGE
740 0460 17 FE53 LSR FILE3 Close Output File
741 0463 17 FF73 LSR CLOUT
742 0466 BD DA03 JSR FMSCLS Close All Files
743 0469 8E 0807 LDX #MESS Abort Message
744 046C BD CD1E JSR PSTENG Print
745 046F 7E CD03 JMP WARNS Warm Start

748 * Storage Locations

750 * SPEED FCR 00 Speed
751 0472 0000
752 0474 00 FILTYP FCR 00 File Type
753 0475 00 HALFDUP FCR 00 Duplex Flag
754 0476 00 NBSTRL FCR 00 Back Space Translation Flag
755 0477 00 F1FLAG FCR 00 File 1 Flag
756 0478 00 F2FLAG FCR 00 File 2 Flag
757 0479 00 FOFLAG FCR 00 File Out Flag
758 047A 00 WALT FCR 00 Wait Flag
### Serial Communication Utility

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>047B 0000</td>
<td>FCB Pointer Store</td>
</tr>
<tr>
<td>047D 0000</td>
<td>Buffer Start Pointer</td>
</tr>
<tr>
<td>047F 0000</td>
<td>Buffer End Pointer</td>
</tr>
<tr>
<td>0481 0000</td>
<td>File Pointer</td>
</tr>
<tr>
<td>0483 0000</td>
<td>Flag Pointer</td>
</tr>
<tr>
<td>0485 0000</td>
<td>Delay Counter</td>
</tr>
<tr>
<td>0487 0000</td>
<td>Storage for Printed No.</td>
</tr>
</tbody>
</table>

### Status Codes

- **$/4**: Buffer Free
- **$/,4**: Buffer Used
- **/,13**: Status
- **- --------*/**: Finished Utility
- **- Error - Hit Any Key/**
- **- Overflow - Hit Any Key/**
- **- Overflow - Saving File/**
- **- Space Compression/**

### Functions

- **FMAM**: Existing Name?
- **FMAMH**: New File Name?
- **ISREDN**: Input Delay Number
- **MDUP**: Set Full Duplex
- **MFDUP**: Set Half Duplex
- **MFS**: Don't Translate BS
- **MFSN**: Translate BS to DEL
- **MFSN**: Don't Translate BS to DEL
- **MFSN**: Space Compression

### Escape-Control Menu

- **Ctrl-S**: Pause or Go
- **Ctrl-D**: In Emulation Mode

### Other Commands

- **Retrieve**: Hit Any Key
- **Send**: Hit Any Key

---

**Notes:**
- **0000**: FCB 
- **000**: FCB Pointer Store
- **00**: Buffer Start Pointer
- **00**: Buffer End Pointer
- **00**: File Pointer
- **00**: Flag Pointer
- **00**: Delay Counter
- **00**: Storage for Printed No.
Serial Communication Utility

```
813 0871 28 20 4E 6P  FCC  // No Space Compress \,4
814 0889 33 20 20 20  HSFCM  FCC  // No Space Compress \\
815 08A8 28 20 53 70  FCC  // Space Compression \,4
816 08C0 34 20 20 20  SPEEM  FCC  // Change Delay Speed \\
817 08DF 28 20 44 65  FCC  // Delay Number \,4
818 08F1 35 20 20 20  FLCH  FCC  // Open Log File \\
819 0910 28 20 4C 6F  FCC  // Log File Closed \,4
820 0928 35 20 20 20  FLOM  FCC  // Close Log File \\
821 0947 28 20 4C 6F  FCC  // Log File Closed \,4
822 095F 36 20 20 20  F2CH  FCC  // Open Storage File \\
823 097F 28 20 33 74  FCC  // Storage File Closed \,4
824 0996 36 20 20 20  F2OM  FCC  // Close Storage File \\
825 09B5 28 20 33 74  FCC  // Storage File Opened \,4
826 09CD 37 20 20 20  FUCN  FCC  // Open Output File \\
827 09EC 28 20 4F 75  FCC  // Output File Closed \,4
828 0A04 37 20 20 20  FUCN  FCC  // Close Output File \\
829 0A23 26 20 4F 75  FCC  // Output File Opened \,4
830 0A3B 38 20 20 20  MENU  FCC  // Exit to Flex System \,13
831 0A55 39 20 20 20  FCC  // Abort this Utility \,13
832 0A6F 52 45 54 20  FCC  // Back to Emulation \,13
833 0A89 0D 43 6E 74  FCC  \,15, Enter Command \,04
834
835  * New Insert user Routines (LIB UART)
836  * -------------------------------------
```
**Serial Communication Utility**

839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891

S e r i a l  C o m m u n i c a t i o n  U t i l i t y 3-1-84 TSC ASSEMBLER PAGE 17

****************************************************

* UART Specific Routines for File Transfer, Terminal
* Written For a MC6850 UART
* By M. K. Hawes
* Last Updated 3 January 1984
*
* SERIAL CARD ROUTINES
*

856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891

CO1B EQU $1B
CO05 SLOT EQU $5
AO05 PARREG EQU $AO05+(SLOT+$10) Parameter Switch Add
AO06 CTREG EQU $AO06+(SLOT+$10) Control/Status Reg
AO07 DATREG EQU $AO07+(SLOT+$10) Data Register Add
AO00 XMODAT EQU $AO00 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.
*
873 OAA5 B4 02  
874 OAA4 B6 03  
875 OAA5 B7 A006  
876 OAA2 B6 11  
877 OAA4 B7 A006  
878 OAA7 35 02  
879 OAA9 39  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  

UNIT PSHS A
LDA $603 Reset UART
STA CTREG
LDA $611 Set for /16 and 2 Stop Bits
STA CTREG
PULS A
RES

* 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

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>892</td>
<td></td>
<td></td>
</tr>
<tr>
<td>893</td>
<td></td>
<td></td>
</tr>
<tr>
<td>894</td>
<td>OAAA 34 02</td>
<td>USTAT PSHS A</td>
</tr>
<tr>
<td>895</td>
<td>OAC B6 A0D6</td>
<td>LDA CTLREG Check Control Register</td>
</tr>
<tr>
<td>896</td>
<td>OAF B4 21</td>
<td>ANDA #521 Check Overflow and Char</td>
</tr>
<tr>
<td>897</td>
<td>OAB1 35 02</td>
<td>PULS A</td>
</tr>
<tr>
<td>898</td>
<td>OAB3 39</td>
<td>RTS</td>
</tr>
<tr>
<td>899</td>
<td></td>
<td></td>
</tr>
<tr>
<td>900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>901</td>
<td></td>
<td></td>
</tr>
<tr>
<td>902</td>
<td></td>
<td></td>
</tr>
<tr>
<td>903</td>
<td></td>
<td></td>
</tr>
<tr>
<td>904</td>
<td></td>
<td></td>
</tr>
<tr>
<td>905</td>
<td></td>
<td></td>
</tr>
<tr>
<td>906</td>
<td></td>
<td></td>
</tr>
<tr>
<td>907</td>
<td>OAB4 B6 A0D7</td>
<td>UIN LDA DATREG Get Character</td>
</tr>
<tr>
<td>908</td>
<td>OAB7 39</td>
<td>RFS</td>
</tr>
<tr>
<td>909</td>
<td></td>
<td></td>
</tr>
<tr>
<td>910</td>
<td></td>
<td></td>
</tr>
<tr>
<td>911</td>
<td></td>
<td></td>
</tr>
<tr>
<td>912</td>
<td></td>
<td></td>
</tr>
<tr>
<td>913</td>
<td></td>
<td></td>
</tr>
<tr>
<td>914</td>
<td></td>
<td></td>
</tr>
<tr>
<td>915</td>
<td></td>
<td></td>
</tr>
<tr>
<td>916</td>
<td></td>
<td></td>
</tr>
<tr>
<td>917</td>
<td></td>
<td></td>
</tr>
<tr>
<td>918</td>
<td></td>
<td></td>
</tr>
<tr>
<td>919</td>
<td></td>
<td></td>
</tr>
<tr>
<td>920</td>
<td>OAB8 34 02</td>
<td>UOUT PSHS A</td>
</tr>
<tr>
<td>921</td>
<td>OABA B6 A0D6</td>
<td>UOUT1 LDA CTLREG Check If Last Char Sent</td>
</tr>
<tr>
<td>922</td>
<td>OABD B4 02</td>
<td>ANDA #02</td>
</tr>
<tr>
<td>923</td>
<td>OABF 27 F9</td>
<td>BEQ UOUT1 If Not Then Wait</td>
</tr>
<tr>
<td>924</td>
<td>OAC1 35 02</td>
<td>PULS A</td>
</tr>
<tr>
<td>925</td>
<td>OAC3 B7 A0D7</td>
<td>STA DATREG Output Character</td>
</tr>
<tr>
<td>926</td>
<td>OAC6 39</td>
<td>RFS</td>
</tr>
<tr>
<td>927</td>
<td></td>
<td></td>
</tr>
<tr>
<td>928</td>
<td></td>
<td></td>
</tr>
<tr>
<td>929</td>
<td></td>
<td></td>
</tr>
<tr>
<td>930</td>
<td></td>
<td></td>
</tr>
<tr>
<td>931</td>
<td></td>
<td></td>
</tr>
<tr>
<td>932</td>
<td></td>
<td></td>
</tr>
<tr>
<td>933</td>
<td></td>
<td></td>
</tr>
<tr>
<td>934</td>
<td></td>
<td></td>
</tr>
<tr>
<td>935</td>
<td></td>
<td></td>
</tr>
<tr>
<td>936</td>
<td></td>
<td></td>
</tr>
<tr>
<td>937</td>
<td></td>
<td></td>
</tr>
<tr>
<td>938</td>
<td></td>
<td></td>
</tr>
<tr>
<td>939</td>
<td></td>
<td></td>
</tr>
<tr>
<td>940</td>
<td>OAG7 B6 A0D5</td>
<td>PARIN LDA PARREG Input from Parameter Reg</td>
</tr>
<tr>
<td>941</td>
<td>OACA 39</td>
<td>RTS Return from Subroutine</td>
</tr>
<tr>
<td>942</td>
<td></td>
<td></td>
</tr>
<tr>
<td>943</td>
<td></td>
<td></td>
</tr>
<tr>
<td>944</td>
<td></td>
<td></td>
</tr>
<tr>
<td>945</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This Routine Inputs a Key From the Keyboard
without echoing it on the Screen. The Key value is placed in the A Reg.

951 OACB B6 A000 KEYIN LDA KBDPAT Input From Keyboard
952 OACE 84 7F ANDA #67F Strib MSB
953 OADD B7 A010 STA KBCLRA Clear Keyboard Strobe
954 OADD 39 RTS Return From Subroutine

* End Of Uart Routines

**********************************************************************************************************
961 * End of Program
962
963
964 OAD4 00 ENDFCM FCE 00 Dummy Byte
965 END TERM

0 ERROR(S) DETECTED
### Symbol Table:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABMESS</td>
<td>ABORT</td>
</tr>
<tr>
<td>BACREC</td>
<td>BAC COMP</td>
</tr>
<tr>
<td>BUFFER</td>
<td>BUFFER</td>
</tr>
<tr>
<td>CLASS</td>
<td>CLASS</td>
</tr>
<tr>
<td>CLOSEF</td>
<td>CLOSEF</td>
</tr>
<tr>
<td>CTRLREG</td>
<td>CTRL REG</td>
</tr>
<tr>
<td>DLETOT</td>
<td>DLETOT</td>
</tr>
<tr>
<td>ENDFIL</td>
<td>ENDFIL</td>
</tr>
<tr>
<td>ERROR</td>
<td>ERROR</td>
</tr>
<tr>
<td>FIOI</td>
<td>FIOI</td>
</tr>
<tr>
<td>FZFLAG</td>
<td>FZ FLAG</td>
</tr>
<tr>
<td>FC5</td>
<td>FC5</td>
</tr>
<tr>
<td>FCDATE</td>
<td>FC DATE</td>
</tr>
<tr>
<td>FILECHK</td>
<td>FILE CHK</td>
</tr>
<tr>
<td>FILESIG</td>
<td>FILE SIG</td>
</tr>
<tr>
<td>FILESIZE</td>
<td>FILE SIZE</td>
</tr>
<tr>
<td>FILER</td>
<td>FILER</td>
</tr>
<tr>
<td>FILERND</td>
<td>FILER ND</td>
</tr>
<tr>
<td>FILERI</td>
<td>FILER I</td>
</tr>
<tr>
<td>FILERMO</td>
<td>FILER MO</td>
</tr>
<tr>
<td>FILERPM</td>
<td>FILER PM</td>
</tr>
<tr>
<td>FILERU</td>
<td>FILER U</td>
</tr>
<tr>
<td>FILERV</td>
<td>FILER V</td>
</tr>
<tr>
<td>FILERX</td>
<td>FILER X</td>
</tr>
<tr>
<td>FILERZ</td>
<td>FILER Z</td>
</tr>
<tr>
<td>FILLREG</td>
<td>FILL REG</td>
</tr>
<tr>
<td>FILLREGX</td>
<td>FILL REG X</td>
</tr>
<tr>
<td>FILLREGY</td>
<td>FILL REG Y</td>
</tr>
<tr>
<td>FILLREGZ</td>
<td>FILL REG Z</td>
</tr>
<tr>
<td>FILLREGD</td>
<td>FILL REG D</td>
</tr>
<tr>
<td>FILLREGS</td>
<td>FILL REG S</td>
</tr>
<tr>
<td>FILLREGT</td>
<td>FILL REG T</td>
</tr>
<tr>
<td>FILLREGU</td>
<td>FILL REG U</td>
</tr>
<tr>
<td>FILLREGV</td>
<td>FILL REG V</td>
</tr>
<tr>
<td>FILLREGW</td>
<td>FILL REG W</td>
</tr>
<tr>
<td>FILLREGX</td>
<td>FILL REG X</td>
</tr>
<tr>
<td>FILLREGY</td>
<td>FILL REG Y</td>
</tr>
<tr>
<td>FILLREGZ</td>
<td>FILL REG Z</td>
</tr>
<tr>
<td>FILLREGD</td>
<td>FILL REG D</td>
</tr>
<tr>
<td>FILLREGS</td>
<td>FILL REG S</td>
</tr>
<tr>
<td>FILLREGT</td>
<td>FILL REG T</td>
</tr>
<tr>
<td>FILLREGU</td>
<td>FILL REG U</td>
</tr>
<tr>
<td>FILLREGV</td>
<td>FILL REG V</td>
</tr>
<tr>
<td>FILLREGW</td>
<td>FILL REG W</td>
</tr>
<tr>
<td>FILLREGX</td>
<td>FILL REG X</td>
</tr>
<tr>
<td>FILLREGY</td>
<td>FILL REG Y</td>
</tr>
<tr>
<td>FILLREGZ</td>
<td>FILL REG Z</td>
</tr>
<tr>
<td>FILLREGD</td>
<td>FILL REG D</td>
</tr>
<tr>
<td>FILLREGS</td>
<td>FILL REG S</td>
</tr>
<tr>
<td>FILLREGT</td>
<td>FILL REG T</td>
</tr>
<tr>
<td>FILLREGU</td>
<td>FILL REG U</td>
</tr>
<tr>
<td>FILLREGV</td>
<td>FILL REG V</td>
</tr>
<tr>
<td>FILLREGW</td>
<td>FILL REG W</td>
</tr>
<tr>
<td>FILLREGX</td>
<td>FILL REG X</td>
</tr>
<tr>
<td>FILLREGY</td>
<td>FILL REG Y</td>
</tr>
<tr>
<td>FILLREGZ</td>
<td>FILL REG Z</td>
</tr>
<tr>
<td>FILLREGD</td>
<td>FILL REG D</td>
</tr>
<tr>
<td>FILLREGS</td>
<td>FILL REG S</td>
</tr>
<tr>
<td>FILLREGT</td>
<td>FILL REG T</td>
</tr>
<tr>
<td>FILLREGU</td>
<td>FILL REG U</td>
</tr>
<tr>
<td>FILLREGV</td>
<td>FILL REG V</td>
</tr>
<tr>
<td>FILLREGW</td>
<td>FILL REG W</td>
</tr>
<tr>
<td>FILLREGX</td>
<td>FILL REG X</td>
</tr>
<tr>
<td>FILLREGY</td>
<td>FILL REG Y</td>
</tr>
<tr>
<td>FILLREGZ</td>
<td>FILL REG Z</td>
</tr>
<tr>
<td>FILLREGD</td>
<td>FILL REG D</td>
</tr>
<tr>
<td>FILLREGS</td>
<td>FILL REG S</td>
</tr>
<tr>
<td>FILLREGT</td>
<td>FILL REG T</td>
</tr>
<tr>
<td>FILLREGU</td>
<td>FILL REG U</td>
</tr>
<tr>
<td>FILLREGV</td>
<td>FILL REG V</td>
</tr>
<tr>
<td>FILLREGW</td>
<td>FILL REG W</td>
</tr>
<tr>
<td>FILLREGX</td>
<td>FILL REG X</td>
</tr>
<tr>
<td>FILLREGY</td>
<td>FILL REG Y</td>
</tr>
<tr>
<td>FILLREGZ</td>
<td>FILL REG Z</td>
</tr>
<tr>
<td>FILLREGD</td>
<td>FILL REG D</td>
</tr>
<tr>
<td>FILLREGS</td>
<td>FILL REG S</td>
</tr>
<tr>
<td>FILLREGT</td>
<td>FILL REG T</td>
</tr>
<tr>
<td>FILLREGU</td>
<td>FILL REG U</td>
</tr>
<tr>
<td>FILLREGV</td>
<td>FILL REG V</td>
</tr>
<tr>
<td>FILLREGW</td>
<td>FILL REG W</td>
</tr>
<tr>
<td>FILLREGX</td>
<td>FILL REG X</td>
</tr>
<tr>
<td>FILLREGY</td>
<td>FILL REG Y</td>
</tr>
<tr>
<td>FILLREGZ</td>
<td>FILL REG Z</td>
</tr>
<tr>
<td>FILLREGD</td>
<td>FILL REG D</td>
</tr>
<tr>
<td>FILLREGS</td>
<td>FILL REG S</td>
</tr>
<tr>
<td>FILLREGT</td>
<td>FILL REG T</td>
</tr>
<tr>
<td>FILLREGU</td>
<td>FILL REG U</td>
</tr>
<tr>
<td>FILLREGV</td>
<td>FILL REG V</td>
</tr>
<tr>
<td>FILLREGW</td>
<td>FILL REG W</td>
</tr>
<tr>
<td>FILLREGX</td>
<td>FILL REG X</td>
</tr>
<tr>
<td>FILLREGY</td>
<td>FILL REG Y</td>
</tr>
<tr>
<td>FILLREGZ</td>
<td>FILL REG Z</td>
</tr>
<tr>
<td>FILLREGD</td>
<td>FILL REG D</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>Logic Type</th>
<th>Logic Equation</th>
<th>Truth Table</th>
<th>Logic Symbol</th>
<th>Logic Equation</th>
<th>Truth Table</th>
<th>Logic Symbol</th>
</tr>
</thead>
</table>
| AND Gate         | \( A \land (B \land C) \land D \) | \[
|                  |                | \begin{array}{c|c|c|c|c}
|                  | 0 & 0 & 0 & 0 & 0 \\
|                  | 0 & 1 & 1 & 1 & 1 \\
|                  | 1 & 0 & 0 & 0 & 0 \\
|                  | 1 & 1 & 1 & 1 & 1 \\
|                  | \end{array} | | \( A \rightarrow (B \land C \land D) \rightarrow E \) | | \[
|                  |                | \begin{array}{c|c|c|c|c}
|                  | 0 & 0 & 0 & 0 & 0 \\
|                  | 0 & 1 & 1 & 1 & 1 \\
|                  | 1 & 0 & 0 & 0 & 0 \\
|                  | 1 & 1 & 1 & 1 & 1 \\
|                  | \end{array} | | \( A \rightarrow (B \land C \land D) \rightarrow E \) | | \[
|                  |                | \begin{array}{c|c|c|c|c}
|                  | 0 & 0 & 0 & 0 & 0 \\
|                  | 0 & 1 & 1 & 1 & 1 \\
|                  | 1 & 0 & 0 & 0 & 0 \\
|                  | 1 & 1 & 1 & 1 & 1 \\
|                  | \end{array} | |

Notes: ( ) denotes optional fields, "o" if not used.

<table>
<thead>
<tr>
<th>Logic Type</th>
<th>Logic Symbol</th>
<th>Logic Equations</th>
<th>Truth Table</th>
<th>Logic Symbol</th>
<th>Logic Equations</th>
<th>Truth Table</th>
<th>Logic Symbol</th>
</tr>
</thead>
</table>
| OR Gate          | \( A \lor (B \lor C) \lor D \lor E \) | \[
|                  |                | \begin{array}{c|c|c|c|c}
|                  | 0 & 0 & 0 & 0 & 0 \\
|                  | 0 & 1 & 1 & 1 & 1 \\
|                  | 1 & 0 & 0 & 0 & 0 \\
|                  | 1 & 1 & 1 & 1 & 1 \\
|                  | \end{array} | | \( A \rightarrow (B \lor C \lor D \lor E) \rightarrow O \) | | \[
|                  |                | \begin{array}{c|c|c|c|c}
|                  | 0 & 0 & 0 & 0 & 0 \\
|                  | 0 & 1 & 1 & 1 & 1 \\
|                  | 1 & 0 & 0 & 0 & 0 \\
|                  | 1 & 1 & 1 & 1 & 1 \\
|                  | \end{array} | | \( A \rightarrow (B \lor C \lor D \lor E) \rightarrow O \) | | \[
|                  |                | \begin{array}{c|c|c|c|c}
|                  | 0 & 0 & 0 & 0 & 0 \\
|                  | 0 & 1 & 1 & 1 & 1 \\
|                  | 1 & 0 & 0 & 0 & 0 \\
|                  | 1 & 1 & 1 & 1 & 1 \\
|                  | \end{array} | |

Notes: ( ) denotes optional fields, "o" if not used.

<table>
<thead>
<tr>
<th>Logic Type</th>
<th>Logic Symbol</th>
<th>Logic Equations</th>
<th>Truth Table</th>
<th>Logic Symbol</th>
<th>Logic Equations</th>
<th>Truth Table</th>
<th>Logic Symbol</th>
</tr>
</thead>
</table>
| EXCLUSIVE OR Gate| \( A \oplus B \)   | \[
|                  |                | \begin{array}{c|c|c|c|c}
|                  | 0 & 0 & 0 & 0 & 0 \\
|                  | 0 & 1 & 1 & 1 & 1 \\
|                  | 1 & 0 & 0 & 0 & 0 \\
|                  | 1 & 1 & 1 & 1 & 1 \\
|                  | \end{array} | | \( A \rightarrow (B \oplus C) \rightarrow \) | | \[
|                  |                | \begin{array}{c|c|c|c|c}
|                  | 0 & 0 & 0 & 0 & 0 \\
|                  | 0 & 1 & 1 & 1 & 1 \\
|                  | 1 & 0 & 0 & 0 & 0 \\
|                  | 1 & 1 & 1 & 1 & 1 \\
|                  | \end{array} | | \( A \rightarrow (B \oplus C) \rightarrow \) | | \[
|                  |                | \begin{array}{c|c|c|c|c}
|                  | 0 & 0 & 0 & 0 & 0 \\
|                  | 0 & 1 & 1 & 1 & 1 \\
|                  | 1 & 0 & 0 & 0 & 0 \\
|                  | 1 & 1 & 1 & 1 & 1 \\
|                  | \end{array} | |

Notes: ( ) denotes optional fields, "o" if not used.
LOGIC TYPE: D-TYPE FLIP-FLOP

LOGIC SYMBOL:

TRUTH TABLE:

<table>
<thead>
<tr>
<th>DATA</th>
<th>CLOCK</th>
<th>RESET</th>
<th>SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

ELEMENT CODING: OFF DATA & CLOCK (RESET) SET 0 (0)

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:

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>SET</th>
<th>RESET</th>
<th>NEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

ELEMENT CODING: J = J WHEN CLOCK (RESET) SET 0 (0)

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:

<table>
<thead>
<tr>
<th>DATA</th>
<th>LOAD</th>
<th>ENABLE</th>
<th>RESET</th>
<th>COUNTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>NO CHANGE</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>NO CHANGE</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NO CHANGE</td>
</tr>
</tbody>
</table>

ELEMENT CODING: CNTNR = LOAD | ENABLE | CLOCK (LOAD) | RESET | 0 (0)

NOTES: X = DON'T CARE, LOAD AND ENABLE ARE ACTIVE HIGH AND LOAD OVERRIDES ENABLE.
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 Appendix.
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 Havens. (June 1983)
90 REM
100 REM -------------------------------------------------------------
110 REM FOR M ATRIX
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 XXX
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 'O'. *
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 CLEAR100:WIDTH235:PRINTCHR$(12)
510 REM *********************************************
520 REM * Automatic Continue *
530 REM *********************************************
540 PRINT"Do You Want Automatic Continuing ? ";
Logic Simulator Program

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$="N"THEN610
640 IFQ$="N"THENAC=0:TN=6:GOTO760
650 START+;HEADING="Digital Simulator Output"
660 PRC+
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:MPRAM=30:MSAMPLES=14
770 MTPFS=50
780 PRINT:PRINTTAB(15*(1+PR));"LOGIC SIMULATOR"
790 PRINTTAB(15*(1+PR));"-------------":PRINT
800 PRINTTM;"NAMES":PRINTTM;"LOGIC DEVICES"
810 PRINTTM;"TYPES":PRINTTM;"EXTERNAL ST IMUX"
820 PRINT:
830 DIMITMSS(MNAMS),MRT(MLGIC)
840 DIMITM(MLGIC,8)
850 DIMITMEXTRNALS(MEXTRNALS,10)
860 DIMITMTPF(MTPFS)
870 DIMITMTEP(MTPFS)
880 DIMITMS(MS)
890 DIMITMS(MS)
900 DIMITM(10),LM$=10,S(672)
910 REM ***************************
920 REM * Start of Program *
920 REM ***************************
930 PRINT:PRINT"*** Initializing Arrays ***"
940 PRINT:PRINT"------------- ***"
950 GOSUB1300
960 PRINT:PRINT"*** Reading Any Macros ***"
970 PRINT:PRINT"------------- ***"
980 GOSUB1300
990 PRINT:PRINT"*** Read In Logic Network ***"
1000 PRINT:PRINT"------------- ***"
1010 PRINT:PRINT:GOSUB1990
1020 PRINT:PRINT"*** Expanding Any Macros ***"
1030 PRINT:PRINT"------------- ***"
1040 PRINT:PRINT:GOSUB2200
1050 PRINT:PRINT"*** Read External Stimuli ***"
1060 PRINT:PRINT:GOSUB2330
1070 PRINT:PRINT:GOSUB2330
1080 PRINT:PRINT:GOSUB2330
1090 PRINT: GOSUB2640
1100 PRINT: PRINT "*** Checking for Errors ***"
1110 GOSUB2930
1120 PRINT: PRINT "*** Read Nodes To Sample ***"
1130 PRINT "*** ----------------------------------- ***"
1140 PRINT: GOSUB2930
1150 PRINT: PRINT "*** Setting Up Simulation ***"
1160 PRINT "*** ----------------------------------- ***"
1170 GOSUB3920
1180 IFFRM10: THEN PRINT: CHR$(12): PRINT "START-
1190 END
1200 REM ********** Special Input Routine **********
1210 REM * Initialize Pointers *
1220 REM ********** Initialize Pointers **********
1230 SC=O: PRINT "":
1240 GET: IFQ<OTEN13LO
1250 SC=SC+1: IFSC>600ANDACTHEN130G
1260 SH=SN+1
1270 IFSH>ANDSF=OTENPRINTCHR$(8); "": SF=1: SH=0
1280 IFSH>ANDSF=OTENPRINTCHR$(8); "": SF=0: SH=0
1290 GOTO1240
1300 Q=ASC(SI$)
1310 Q$=CHR$(Q): PRINTCHR$(8): Q$
1320 RETURN
1330 REM ********** Initialize Pointers **********
1340 REM * Initialize Pointers *
1350 REM * [ 500 ] *
1360 REM ********** Initialize Pointers **********
1370 NL=OsNTYPE=O
1380 READQS
1390 IPQ<"LAST" "TEN$(NT)=Q$: NT=NT+I: GOTO 1280
1400 DATA "END", "AND", "NAND", "OR", "NOR", "XOR"
1410 DATA "NOR", "DPP", "JKPP", "GMR", "PLSR"
1420 DATA "SREG", "", "LAST"
1430 NTYPE=NTYPE+NTYPE+1
1440 PRINT Logic Print Values and Inversions
1450 LO=0: I=0: LX=L
1460 DMLVS$(3), LPS$(3), LW(3)
1470 LVS(LO)="O": LPS$(LO)="": LW(LO)=L
1480 LVS(LL)="1": LPS$(LL)="1": LW(LL)=L
1490 LVS(LX)="X": LPS$(LX)="X": LW(LX)=L
1500 DTMTEX$(I): FORI=GTO I: READTEX$(I): NEXT I
1510 DATA "A", "P"
1520 RETURN
1530 REM ********** Initialize Pointers **********
1540 REM * Read In Macros *
1550 REM * [ 1000 ] *
1560 REM ********** Initialize Pointers **********
1570 GOSUB6190
1580 IFLEFT5 (L$, 3)<> "MAC" THEN RETURN
1590 X$=L$(1)
1600 PRINT "**** Reading in Macro : "; X$; "****"
1610 PRINT: TFCR=-1
1630 GOSUB 6380
1640 IF POUND = "THEN 1710
1650 IF PTR = "THEN 1710
1660 PRINT "This Macro Name Already Exists."
1670 PRINT "Enter R (replace) or N (replace) ";
1680 SI$ = "R"; GOSUB 1200
1690 IF SI$ = "R" THEN 1760
1700 IF SI$ = "N" THEN 1670
1710 PTR = NLGC
1720 GOSUB 6470; MT = FO
1730 GOSUB 2070
1740 GOSUB 1820
1750 GOTO 1530
1760 REM ****************************
1770 REM * Skip This Macro *
1780 REM [* 1200 ] *
1790 REM ****************************
1800 GOSUB 6190; IF L$ (0) = "END" THEN 1800
1810 GOTO 1200
1820 REM ****************************
1830 REM * Check for Macro Errors *
1840 REM [* 1300 ] *
1850 REM ****************************
1860 PRINT; PRINT "*** Checking Macro " ; TY$(MT) ;
1870 PRINT " for Errors ***"
1880 XI = TY$(MT) ; GOSUB 6810
1890 FOR I = 0 TO
1900 XI = LO (XI, I) ; GOSUB 6870
1910 NEXT I
1920 XI = XI + 1
1930 XI = LO (XI, 0) ; IF TY$(XI) = "* " THEN 1890
1940 XI = XI
1950 GOSUB 7010
1960 GOSUB 7100
1970 ER = 0
1980 RETURN
1990 REM ****************************
2000 REM * Read In Logic Description *
2010 REM [* 2000 ] *
2020 REM ****************************
2030 IF L$(6, 8) = "ELEMENTS THEN 2070
2040 PRINT; PRINT "Error, ELEMENTS Expected."
2050 IF B = THEN FRINT "IO" ; PRINT CH$(12) ; START
2060 END
2070 SL = XL; MA$(0) = "O" ; NAM = 1
2080 FORI = 1 TO N; MA$(I) = "" ; NEXT I
2090 GOSUB 6190
2100 X$ = L$I (0)
2110 GOSUB 6380; IF PO = 12 THEN GOSUB 6470
2120 T (0) = FO
2130 FORI = 1 TO 7
2140 X$ = L$I (I) ; GOSUB 6560
2150 IF PO = 1 THEN GOSUB 6730
2160 T (I) = FO; NEXT I
LOGIC SIMULATOR PROGRAM

2170 GOSUB 7300
2180 IF T(0) = 0 THEN ZI 90
2190 RETURN
2200 REM ******************************************
2210 REM * Expand Macros *
2220 REM * [ 3000 ] *
2230 REM ******************************************
2240 ST = SL
2250 FOR I = ST ONL - 1
2260 IF L0(I, 0) > NPRIM THEN 2290
2270 NEXT I
2280 RETURN
2290 GOSUB 7590
2300 GOSUB 8000
2310 REM Repeat Proc to Find Next Macro
2320 GOTO 2200
2330 REM ******************************************
2340 REM * List Expanded Logic Description *
2350 REM * [ 3500 ] *
2360 REM ******************************************
2370 PRINT: PRINT
2380 PRINT "*** Expanded Logic Description ***"
2390 PRINT "---------------------------------------------------------------------"
2400 FOR I = 1 TO 5: PRINT TAB(I*TN); "No."; TAB(2*TN); "Type";
2410 PRINT TAB(TN); "Name"; TAB(3*TN); "Val"; TAB(7/2*TN+1); "Bgn";
2420 PRINT TAB(7*TN); "Outpus";
2430 FOR I = 1 TO TOTN - 1: PRINT TAB(TN*14TN - I); "Q";
2440 PRINT TAB(TN); "Q-";
2450 FOR I = 1 TO TOTN - 1: PRINT "-"; NEXT I;
2460 PRINT "+
2470 NEXT I: NEXT I: PRINT "+
2480 NAS(0) = ";
2490 FOR I = 1 TO TOTN - 1
2500 IF L0(I, 0) = 1 THEN PRINT "(\"; PRINT RIGHT$(STR$(I-SL+1), LEN$(STR$(I-SL+1))) - 1);
2510 PRINT "\"
2520 PRINT "\"
2530 PRINT TAB(TN);
2540 PRINT$(LO(I, 0));
2550 FOR I = 1 TO 7
2560 NEXT I
2570 PRINT$(TN*14TN+1);
2580 XX = LO(I, II)
2590 IF XX = 0 THEN PRINT "+"; "XX: GOTO 10
2600 PRINT$(XX); " ";
2610 PRINT$(XX+1); PRINT $(XX)
2620 FOR I = 1 TO 7
2630 PRINT$(XX)
2640 RETURN
2650 REM ******************************************
2660 REM * Read In External Description *
2670 REM * [ 4000 ] *
2680 REM ******************************************
2690 PRINT "No."; TAB(TN); "Type"; TAB(2*TN); "Val"; TAB(7/2*TN+1); "Bgn";
2700 FOR I = 1 TO 8: PRINT$(I*TN/2+1); "Qs";
2710 PRINTRIGHT$(STR$(I),1);NEXT I:PRINT
2720 FOR I=1 TO 3:PRINT"+";FOR I=1 TO TN-1
2730 PRINT"-";NEXT I:NEXT I:PRINT"-";
2740 FOR I=1 TO 7:PRINT"+";FOR I=1 TO TN/2-1
2750 PRINT"-";NEXT I:NEXT I:PRINT"+
2760 NEXT X:NEXT 6190
2770 IFLEFT$(L$,9);="EXTERNALS"THEN 2810
2780 PRINT"Error, EXTERNALS Expected."
2790 IFPRINTTHENPRINT$;PRINTCHR$(12):START-
2800 END
2810 GOSUB6190
2820 IF L IS(0);="END" THEN PRINTTAB(TN); "END"; RETURN
2830 FOR I=O TO 1:IF L IS(0) THEN I(O)=I
2840 NEXT I
2850 IF I(EQ 1) THEN 2810
2860 T(I)=VAL(L$(I)):GOSUB6560
2870 IF I¬(I)+TTHEN 2810
2880 FOR I=0 TO 3:IF L IS(0) THEN I(2)=L V$(I)
2890 NEXT I
2900 FOR I=3 TO 9: T(I)=VAL(L$(I)):NEXT I
2910 GOSUB 7430
2920 GOTO2810
2930 REM ***************
2940 REM * Check for Errors *
2950 REM * [ 5000 ] *
2960 REM ***************
2970 GOSUB6810;MM=0
2980 FORI=0 TO TN
2990 OL(EX(I,1))=1
3000 NEXT I
3010 ST=SL;GOSUB7010
3020 GOSUB7100
3030 RETURN
3040 REM **********************
3050 REM * Read in Sample Points *
3060 REM * [ 6000 ] *
3070 REM **********************
3080 NS=O
3090 GOSUB6190
3100 IF L IS(7);="OUTPUTS"THEN 3140
3110 PRINT"Error, OUTPUTS Expected."
3120 IFPRINTTHENPRINT$;PRINTCHR$(12):START-
3130 END
3140 GOSUB6190
3150 IF L IS(0);="END"THEN 31220
3160 FORI=0 TO 10
3170 LS=L$(I):IF X=" THEN 3220
3180 GOSUB6560:IF $="1THLM3200
3190 IF NS<14THENNS=NS+1:SA(NS)=PO
3200 NEXT I
3210 GOTO3140
3220 IF NS>7THEN 3260
3230 FORI=0 TO NS
3240 PRINTTAB((I-1)*TN);NAS(SA(I));
Logic Simulator Program

3250 NEXT I; PRINT; PRINT "END" : RETURN
3260 FOR I=1 TO 7
3270 PRINT AS(('I-1)*TN); NA$(SA(I));
3280 NEXT I; PRINT; PRINT "END": RETURN
3290 PRINT AS(('I-6)*TN); NA$(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+CT+NH))*2
3370 A2=PRE(O)-200
3380 IF A2<CAH THEN RETURN
3390 PRINT
3400 PRINT "*** Build Selective Trace Table ***"
3410 PRINT "*** --------------------------- ***"
3420 PRINT
3430 STF=1: DIMSO(NL), S1(NN), S2(TL+NN)
3440 T1=0
3450 FOR I=1 TO S1(I)=T1
3460 FOR J=S1(NN): FOR K=1 TO S5
3470 IF LF(J,K)-nUENS2(T1)=J: T1=T1+1
3480 NEXT K: NEXT J
3490 S2(T1)=-1; T1=T1+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=1 TO S5
3580 IF LF$ (SA(I),1)<>"&" THEN SE=1
3590 SRT(I)=1
3600 NEXT I
3610 FL=0
3620 FOR I=1 TO SE-1
3630 IF NAS(SRT(I))<NAS(SRT(I+1)) THEN 3680
3640 I=SRT(I)
3650 SRT(I)=SRT(I+1)
3660 SRT(I+1)=I
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=LX NYN=PRINTNAS(SRT(JJ)); TAB(TN);
3770 X$=NAS(SRT(JJ)); GOSUB 3560
3780 IF X$ THEN 3680
Logic Simulator Program

3790 FOR I=SL-TML-1
3800 IFLO(I,6)=THEN PRINT "[ Q- ];Q=I-SL+1:GOTO 3860
3810 IFLO(I,7)=THEN PRINT "[ Q- ];Q=I-SL+1:GOTO 3860
3820 NEXT I
3830 FOR I=ONEX
3840 IFEX(I,1)=THEN PRINT "[ EXT ];EXIT(I+1;0):GOTO 3860
3850 NEXT I
3860 OS-RIGHT($(" "+STR$(Q)+")",6)
3870 PRINT Q;TAB(3*TN);T1=SL(SF(TJ));
3880 IF$2(T1)=I-THEN 3910
3890 PRINT$2(T1)-SL+1;T1=T1+1;GOTO 3860
3900 NEXT JJ:RETURN
3910 PRINT NEXT JJ:RETURN
3920 REM ********************************
3930 REM * Do Simulation *
3940 REM * [7000 ] *
3950 REM ********************************
3960 DIMHE(NN);ME(MM);REM()=HEN THEN RETURN
3970 NN=NN-1;NL=NL-1;NE=NE-1;QA=0
3980 GOSUB 3310
3990 GOSUB 4080
4000 GOSUB 4830
4010 GOSUB 5030
4020 GOSUB 5890
4030 IF$="Q"THEN 3990
4040 IF$="R"THEN 3990
4050 GOSUB 8230
4060 GOSUB 4390
4070 GOTO 4000
4080 REM ********************************
4090 REM * Initialize for Simulation *
4100 REM *
4110 REM ********************************
4120 FOR I=ON:NE(I)=LX: NEXT I
4130 FOR I=SL-TML:UL(I,8)=LX:NEXT I
4140 PRINT CHR$(12):E3=0
4150 PRINT "Default for Start & Increment ? ";
4160 SI$="Y":GOSUB 1200
4170 IFQS=$"Y"ANDQ$<>"N"THEN 4150
4180 S9=0:S2=1
4190 IFQS=$"Y"THEN 4220
4200 INPUT "Enter Sample Start Time ";S9
4210 INPUT "Enter Increment Time ";S2
4220 SI=INT(S9):S2=INT(S2)
4230 IFPE$3690:
4240 IF$2=OTEN THEN =S1
4250 ES=32767:FOR I=OTEN:E3=EX(I,3)
4260 GOSUB 4670:RETURN
4270 IFNS=OTHEN GOSUB 5890
4280 IFMM=OTHEN GOSUB 4300
4290 Q$="":RETURN
4300 REM ********************************
4310 REM * Set Aside Mem for SREGs *
4320 REM *

Logic Simulator Program 2-29-84

4330 REM *****************************************************
4340 M1=0
4350 PRI=0:MB(I,0)=LX:ME(I,1)=LX:NEXT I
4360 PRI=SLTONL
4370 IFY$(LO(I,0))="SEC"THENLO(I,8)=HI:ML=LO(I,2)
4380 NEXT I:RETURN
4390 REM *****************************************************
4400 REM * Recalculate Ripple Time *
4410 REM * [ 7500 ] *
4420 REM *****************************************************
4430 IF9=THENRI=RI+1:RETURN
4440 RI=E9:IFRI<THENRI=9
4450 RETURN
4460 REM *****************************************************
4470 REM * Get Next Scheduled External *
4480 REM * [ 8000 ] *
4490 REM *****************************************************
4500 IFRI=THENRETURN
4510 E9=32767
4520 PRI=0:TNLX
4530 E3=EX(I,10)
4540 IFRI<THENGO TO SUB4580
4550 COSUB4760
4560 NEXT I
4570 RETURN
4580 REM *****************************************************
4590 REM * Find Next Scheduled Change *
4600 REM * [ 8100 ] *
4610 REM *****************************************************
4620 E0=EX(I,0);E1=EX(I,1)
4630 E2=EX(I,2);E3=EX(I,3)
4640 Y=NE(E1)
4650 IFRI<THENLNE(Y)
4660 NE(E1)=LN(Y)
4670 REM *****************************************************
4680 REM * Schedule Next Change *
4690 REM * [ 8150 ] *
4700 REM *****************************************************
4710 IFEX$(E0)="P"THENES=RX(I,10)+EX(I,4):RETURN
4720 J=4;E3=32767
4730 IFJ>=THENRETURN
4740 IFEX$(I,J)<THENJ=J+1:GOT04730
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 IFE3>E3THENMEM9=E3
4820 RETURN
4830 REM *****************************************************
4840 REM * Move New to Old, Mark changes *
4850 REM * [ 8500 ] *
4860 REM *****************************************************
Logic Simulator Program

4870 C9=0:IFSTF<0THENGOSUB4970
4880 FORI=1TONN
4890 IFNE(I)<COL(I)THENGOSUB4920
4900 NEXTI
4910 RETURN
4920 C9=I:OL(I)=NE(I)
4930 IFSTF=THENRETURNN
4940 J=S2(I)
4950 IFJ<>THENRBURK
4960 II=SL(I):IFJ— THENRBURK
4970 REM ****************************
4980 REM * Clear SO *
4990 REM ****************************
5000 FORZI»SLrONL:SO(II)=0:NEXrH
5010 RETURN
5020 REM ****************************
5030 REM * Display Logic Values *
5040 REM 9000 *
5050 REM ****************************
5060 REM **********************
5070 SCREEN1,16:FRBfT "Time" ;R1;:GETQB:QA-QA+QB
5080 IFQBOOTHENSCREEN1,1:PRDfl"Stopped"
5090 IFPC=CriHENGOSUB5150
5100 IFPC=THENGOSUB5150
5110 GDOSUB5380
5120 S9=S9+S2
5130 PC=PC+1:IF(PC>45)OR(QA<>0)THENGOSUB5640
5140 QA=0:RETURNN
5150 REM ****************************
5160 REM * Set up Display (Names & Tics) *
5170 REM 9100 *
5180 REM ****************************
5190 FORI=1TOT47:SCREEN1,16
5200 PRINT":":NEXTI
5210 PC=1:FORI=IT015:SCREEN1,1
5220 PRINTCHR$(27):NEXTI:GOSUB5260
5230 FORI=IT0N$:SCREEN1,1+1:PRINTMA$(SA(I));
5240 NEXTI
5250 RETURN
5260 REM ****************************
5270 REM * Print Tic Marks *
5280 REM 9200 *
5290 REM ****************************
5300 SCREEN1,1:PRINTCHR$(27)
5310 FDEX=L2T043:SCREENX,1
5320 PRINTCHR$(152):NEXTX
5330 FORI=IT06:SCREEN11+5*I,1:PRINTCHR$(153);
5340 NEXTI
5350 SCREEN11,1:PRINTCHR$(146);
5360 SCREEN46,1:PRINTCHR$(147);
5370 REM ****************************
5380 REM * Print Values Down Column *
5390 REM 9500 *
5400 REM ****************************
5410 REM ******************************
5420 IF INT((PC-1)/5)*5<(PC-1) THEN 5440
5430 SCREEN PC-1, 16: PRINT "!";
5440 FOR I=1 TO I+1: PRINT NAS(I); NEXT I
5460 RETURN
5470 REM ******************************
5480 REM * Print Screen *
5490 REM * [???? ] *
5500 REM ******************************
5510 CT=1: HP=1
5520 FOR I=1 TO 2954: PEEK(I)
5530 FOR J=1 TO 46
5540 ST(CT)=PEEK(J): CT=CT+1
5550 NEXT J
5560 ST(CT)=13: CT=CT+1
5570 NEXT I: SCREEN 1, 15: PRINT "FULL SCREEN, GET RESPONSE";
5580 IF Q$="C" OR Q$="R" OR Q$="Q" THEN 5770
5590 IF Q$="M" OR Q$="N" THEN 5770
5600 IF Q$="Q" THEN GOSUB 5260: RETURN
5610 PC=0
5620 IF Q$="C" THEN RETURN
5630 PRINT CHR$(12)
5640 PRINT "CHANGE SIGNALS (DISPLAYED) OR VALUES";
5650 PRINT "VALUES";
5660 INPUT X$: PRINT X$;
5670 IF X$="D" AND X$<"V" THEN 5830
5680 IF X$="V" THEN 5990
5690 PRINT "AS EACH CURRENT SAMPLE NODE IS";
5700 PRINT "DISPLAYED, HIT RETURN FOR NO"
5710 PRINT "CHANGE, NEW NAME FOR CHANGE"
5720 FOR I=1 TO 14
5730 X$="": PRINT NAS(I); INPUT X$
5740 IF X$="" THEN 5970
Logic Simulator Program

5950 GOSUB 6560: IF FO = "THEN" GOTO 5930
5960 SA (1) = FO
5970 IFSA(I) < " THEN NS = I
5980 NEXT I: RETURN
5990 PRINT
6000 PRINT: "Type Signal Name or Return: ";
6010 X$ = "**INPUT**$
6020 IFX$ = "*THEN*": RETURN
6030 GOSUB 6560: IF FO = "THEN" GOTO 6000
6040 OV = NE (FO)
6050 PRINT$ (FO, "="; LPS(NE(FO), "-> ";
6060 X$ = "**INPUT**$
6070 IFX$ = "*THEN*(FO) = 0: GOTO 6120
6080 IFX$ = "*THEN*(FO) = 3: GOTO 6120
6090 IFX$ = "*THEN*(FO) = 1: GOTO 6120
6100 IFij$="*THEN*6000
6110 GOTO 6050 
6120 I = FO: IF TYPE(I) < "END": GOTO 4920
6130 IPOV = NE(FO) THEN 6000
6140 IF THENBEGIN +
6150 PRINT$ (FO, "TAB(10)"); ("; LPS(OV)
6160 PRINT") * " ("; LPS(NE(FO), "); ");
6170 IF THENBEGIN -
6180 GOTO 6000
6190 REM ****************************
6200 REM * Read and Parse Line *
6210 REM [ 10000 ] *
6220 REM ****************************
6230 IFIE, -1 THEN L$ = "END": GOTO 6250
6240 READ L$
6250 IF LEFT$(L$, 1 ) = "X": THEN I = 1: L$ = "END"
6260 I = 1: I3 = 1: L1 = 0: I2 = LEN(L$)
6270 I = I1: IF IZ THEN 6290
6280 IF MID$(L$, I, 1 ) < " THEN 6270
6290 L1$(I3) = MID$(L$, I, 1 - I3)
6300 I3 = I3 + 1
6310 IFL1 > 10 THEN RETURN
6320 I = I1: IFI1 = I3 THEN 6350
6330 IF MID$(L$, I, 1 ) = " THEN 6320
6340 I3 = I1: GOTO 6270
6350 FOR I = 1 TO 10
6360 LI$(I) = " : NEXT I
6370 RETURN
6380 REM ****************************
6390 REM * Look Up Type *
6400 REM [ 11000 ] *
6410 REM ****************************
6420 FO = -1
6430 FOR II = 0 TO TYPE -1
6440 IFX$ = "*TYPE*" (II) THEN FO = II: TP = TYPE (II)
6450 NEXT II
6460 RETURN
6470 REM ****************************
6480 REM * ADD TYPE *
Logic Simulator Program  

6490 REM *  [ 11100 ] *
6500 REM ********************************
6510 IFF0—ITHENFO—NTYP—NTYP+1
6520 IFF0—NTYPTHENNTYP—FO
6530 TF$(FO) »X$
6540 TYPE(FO) »TFR
6550 RETURN
6560 REM ********************************
6570 REM * LOOK UP SIGNAL NAME *
6580 REM *
6590 REM ********************************
6600 FO—1:FX$»""THENX$»"0"
6610 IFLETS$(X$,1)»""THEN6660
6620 FORII—OTON(NAM
6630 IFX$»NAM$(II)THEMF0»I1
6640 NEXT II
6650 RETURN
6660 REM ********************************
6670 REM "NAME" IS A NUMBER STORE AS NEG *
6680 REM *
6690 REM ********************************
6700 XX$»RIGHT$(X$,LEN(X$)-1)
6710 FO»VAL(XX$):IFF0—ITHENFO=0
6720 RETURN
6730 REM ********************************
6740 REM * Add Name to List *
6750 REM *
6760 REM ********************************
6770 IFF0—ITHENFO—NYAM$NYAM+1
6780 IFPO—NYAMTHENNYAM—FO+1
6790 NAM$(FO) »X$
6800 RETURN
6810 REM ********************************
6820 REM * Clear Old Array *
6830 REM *
6840 REM ********************************
6850 FORi=OTONM:OL(I) »0:NEXTI
6860 TL=0:RETURN
6870 REM ********************************
6880 REM * Mark and Check for Duplicates *
6890 REM *
6900 REM ********************************
6910 IFX»OTHENRETURN
6920 IFOL(X)>OTHENGOSUB6940
6930 OL(X)»1:RETURN
6940 PRINT "Error Multiple Defn Output ":NAM$(X)
6950 GOSUB6960:RETURN
6960 PRINT "Enter I(gnore) To Ignore Error,"
6970 PRINT "Any Other To Prevent Simulation";
6980 SIG»"1":GOSUB1200
6990 IFQ$»"1"THENERR=1
7000 RETURN
7010 REM ********************************
7020 REM * Check for Multiple Outputs *
7030 REM *
7040 REM *************************************************
7050 XI = ST
7060 FOR I = X1 TO N1 - 1
7070 FOR J = 1 TO 7
7080 X[LO(I, J)] = GOSUB 6870
7090 NEXT J
7100 REM *************************************************
7110 REM * Check for Defined Inputs *
7120 REM *
7130 REM *************************************************
7140 XI = ST
7150 FOR I = X1 TO N1 - 1
7160 FOR J = 1 TO 5
7170 X[LO(I, J)] = GOSUB 7200
7180 IF X < 0 THEN RETURN
7190 NEXT J
7200 REM ************************************
7210 REM * Check For Mark *
7220 REM *
7230 REM ************************************
7240 IF X < 0 THEN RETURN
7250 IF OL(X) = 0 OR HC SUB 7270
7260 PRINT "Error Undefined Input "; NA$(X)
7270 GOSUB 6960; RETURN
7280 ER = 1; RETURN
7290 REM ************************************
7300 REM * Add T(0) - T(7) To Logic Array *
7310 REM *
7320 REM *
7330 REM ************************************
7340 FOR II = 0 TO 7
7350 LO(NL, II) = T[II]
7360 NEXT II
7370 PRINT$(LO(NL, 0));" ";
7380 IF TY$(LO(NL, 0)) = "END" THEN 7420
7390 FOR II = 0 TO 7: XX = LO(NL, II); PRINTTAB(0\$*II)
7400 IF XX = THEN PRINT " "; XX: " "; GOTO 7420
7410 PRINT NA$(XX): "; ": NEXT II
7420 PRINT NL = NL + 1; RETURN
7430 REM ************************************
7440 REM * Add T(0) - T(9) To Extern Array *
7450 REM *
7460 REM *
7470 FOR II = 0 TO 9
7480 EX(NE, II) = T[II]
7490 NEXT II
7500 Q$ = STR$(NE + 1)
7510 PRINT (" "; RIGHT$(Q$, LEN(Q$) - 1));" ";
7520 PRINTTAB(TN); TEXT$(EX(NE, 0)): " ";
7530 PRINTTAB(2*TN); NA$(EX(NE, 1)): " ";
7540 PRINTTAB(3*TN); LV$(EX(NE, 2)): " ";
7550 FOR II = 3 TO 9
7560 PRINTTAB((II + 4)*TN; STR$(EX(NE, II))
Logic Simulator Program

7570 NEXT II: PRINT
7580 NL=NL+1: RETURN
7590 REM *************************************
7600 REM * Build Macro Correspondence Array *
7610 REM * [ 30000 ] *
7620 REM *************************************
7630 I0=I:X=LO(I,0): J=I:TYPE(X):Y=LO(J,0)
7640 M1=I:LO(0)=0: PRINT
7650 PRINT "*** Expanding Macro: \"";TY$(X);\" *** "
7660 PRINT : FORK=ITONAM: LO(K)=1: NEXT K
7670 IFX=0 THEN 7920
7680 FORK=IT03
7690 I1=LO(M1,K): I2=LO(LO,K)
7700 IF I1<>0 AND I2<>0 THEN 7720
7710 LO(I1)=I2
7720 NEXT K
7730 FORK=GTO7
7740 I1=LO(M1,K): I2=LO(LO,K)
7750 IF I1<>0 AND I2=0 THEN 7770
7760 LO(I1)=I2
7770 NEXT K
7780 QOSUM7820
7790 M1=M1+1: X=LO(M1,0): X=LO(LO,0)
7800 IFTY$(Y)="**THEN7670
7810 RETURN
7820 REM *************************************
7830 REM * Move El Array Up One *
7840 REM * [ 30500 ] *
7850 REM *************************************
7860 NL=NL-1
7870 FORK=ITONL-1
7880 FORK1=GTO8
7890 LO(K,K1)=LO(K+1,K1)
7900 NEXT K1; NEXT K
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 IFPR0RTHENPRINT: 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=IT07
8070 X=LO(M1,K)
8080 IFX=0 THEN GOTO8110
8090 IPOL(X)=-1: GOSUB8150
8100 T(K)=LO(X)
Logic Simulator Program

```
7570 NEXT II: PRINT
7580 NS=NS+1: RETURN
7590 REM ************************************************************************
7600 REM * Build Macro Correspondence Array *
7610 REM *
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$(X), "
7660 PRINT FORK=ITONAM:OL(K)=1:NEXT K
7670 IFX<THEN7920
7680 FORK=IT05
7690 I1=LO(M1,K): I2=LO(L0,K)
7700 IFI1=AND I2<>THEN7720
7710 OL(I1)=I2
7720 NEXT K
7730 FORK=ST07
7740 I1=LO(M1,K): I2=LO(L0,K)
7750 IFI1<>ANDI2=THEN7720
7760 OL(I1)=I2
7770 NEXT K
7780 GOSUB7820
7790 H1=M1+1: Y=LO(M1,0) X=LO(L0,0)
7800 IFYS(Y)=""THEN7670
7810 RETURN
7820 REM ************************************************************************
7830 REM * Move El Array Up One *
7840 REM *
7850 REM ************************************************************************
7860 NL=NL-1
7870 FORK=LOT0NL-1
7880 FORK=LOT08
7890 LO(K,KL)=LO(K+1,KL)
7900 NEXT K: NEXT K
7910 RETURN
7920 REM ************************************************************************
7930 REM * Mismatch on Macro Call *
7940 REM *
7950 REM ************************************************************************
7960 REM Should Not Happen But Never Know
7970 PRINT "Mismatch on Macro Call, Cant cont"
7980 IFPR0PTHENPRF#1;PR INTCHR$(12):START-7990 END
8000 REM ************************************************************************
8010 REM * Add Expanded Macro to Elx Ary *
8020 REM *
8030 REM ************************************************************************
8040 NL=NL-1
8050 T(U)=LO(M1,C)
8060 FORK=IT07
8070 X=LO(M1,K)
8080 IFX<THEN7(K)=X;GOTO8110
8090 IFOL(X)=-1THEN GOSUB8150
8100 T(K)=OL(X)
```
8110 NEXT K
8120 GOSUB 7300
8130 ML=ML+1:IFT(ML)
8140 RETURN
8150 REM ****************************
8160 REM * Add Unique Name, Number *
8170 REM ****************************
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 ****************************
8260 LT=LT0(L,O):FORI=1TO5:IN(I)=LT0(L,I):NEXT I
8270 QT=LT0(L,6):Q1=LT0(L,7)
8280 NHT=L0(L,50,8540,8590,8580,8630,8620,8660,8660,9000,9130
8290 REM *** AND NAND OR NOR XOR XNOR ***
8300 GOSUB 8440
8310 RETURN
8320 REM ****************************
8330 REM * Store Q and Q- *
8340 REM ****************************
8350 OL(0)-LN(OL(Q1)):XQ=OL(QQ):RETURN
8360 X=Q0:Q1=Q0:X=REM *** NAND GATE ***
8370 OL(Q)=L1:T=L1:REM *** AND GATE ***
8380 FORI=1TO5:Y=YAND0L(IN(I)):NEXT I
8390 RETURN
8400 X=Q0:Q1=Q0:X=REM *** NOR GATE ***
8410 OL(Q)=L1:T=L1:REM *** OR GATE ***
8420 FORI=1TO5:Y=YOR0L(IN(I)):NEXT I
8430 RETURN
8440 X=Q0:Q1=Q0:X=REM *** XNOR GATE ***
8450 OL(Q)=L1:REM *** XOR GATE ***
8460 Y=OL(IN(1)):Y1=L1(Y):Y2=OL(IN(2))
Logic Simulator Program

8650 GOSUB 9260: RETURN
8660 REM *******************************
8670 REM * D Flip Flop *
8680 REM * [ 41500 ] *
8690 REM *******************************
8700 GOSUB 8490: 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 GOSUB 9320: LO(L,8)=YI
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 GOSUB 8490: XX=OL(0): LX=OL(IN(3))
8820 DO=OL(IN(1)): DL=LN(OL(IN(2)))
8830 OL(0)=L1: XR=OL(IN(4)): XS=OL(IN(5))
8840 XL=LO(L,8)
8850 GOSUB 9450: LO(L,8)=YI: RETURN
8860 REM *******************************
8870 REM * Parallel Load Counter *
8880 REM * [ 41700 ] *
8890 REM *******************************
8900 LO(L,0)=LX: YE=OL(IN(2)): XC=OL(IN(3))
8910 OL(0)=LO: XL=OL(IN(4))
8920 OL(0)=L1: XR=OL(IN(5)): XS=L1
8930 GOSUB 8490: XL=LO(L,8)
8940 YC=YE: YO=XC: Y1=LN(YO)
8950 GOSUB 9260: DO=Y: DL=OL(IN(1)): GOSUB 9450
8960 LO(L,8)=YI: IFTYS(LO(L+1,0))<"*" THEN RETURN
8970 GOSUB 8440: LX=IN(1): LO(L,1)
8980 QO=LO(L,6): Q1=LO(L,7)
8990 YE=YEANDY: GOTO 8930
9000 REM *******************************
9010 REM * Parallel Load Shift Register *
9020 REM * [ 41800 ] *
9030 REM *******************************
9040 OL(0)=LX: DO=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)=LO: DL=OL(IN(1)): GOSUB 8490
9080 XL=LO(L,8): GOSUB 9450
9090 LO(L,8)=YI: IFTYS(LO(L+1,0))<"*" THEN RETURN
9100 GOSUB 8440: LX=IN(1): LO(L,1): QO=LO(L,6)
9110 Q1=LO(L,7): DO=Y
9120 GOTO 9070
9130 REM *******************************
9140 REM * N Bit Shift Register *
9150 REM * [ 41900 ] *
9160 REM *******************************
9170 IF IN(2)=0 THEN 8660
9180 OL(0)=LX: XD=OL(IN(1)): XC=OL(IN(3))
Logic Simulation Program

9190 OL(0)=L1:XR=OL(IN(4)):XS=OL(IN(5))
9200 M1=OL(L,6)
9210 FOR I=1 TO IN(2)
9220 XL=ME(M1,0):XQ=ME(M1,1)
9230 GOSUB9320:XD=XQ:YL=ME(M1,0)
9240 M1=M1+1:NEXT I
9250 RETURN
9260 REM *****************************************************
9270 REM * MUX Function *
9280 REM * [ 45000 ] *
9290 REM *****************************************************
9300 Y«(YO AND LN(YC)) OR (YL AND YC)
9310 RETURN
9320 REM *****************************************************
9330 REM * Shift Register Function *
9340 REM * [ 45100 ] *
9350 REM *****************************************************
9360 YG= XD:Y Q= XC:YL=XL
9370 GOSUB9250:GOSUB9400
9380 YL=Y Q= X Q:GOSUB9250:GOSUB9400
9390 RETURN
9400 REM *****************************************************
9410 REM * Set and Reset function *
9420 REM * [ 45200 ] *
9430 REM *****************************************************
9440 Y= (YO AND(XS))AND XD: RETURN
9450 REM *****************************************************
9460 REM * MUX and DFF *
9470 REM * [ 45300 ] *
9480 REM *****************************************************
9490 Y0=DO:Y1=DI:XC=XM:GOSUB9250
9500 XD=X: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 G X : PRESET CLEAR Q Q-
9600 DATA JKFF J K G X C LOCK CLEAR PRESET Q Q-
9610 DATA NAND G X : 0 0 0 0 C LOCK
9620 DATA END
9630 DATA MACRO SHIFT
9640 DATA SHIF T G X J K : CL R- SH/LO QA
9650 DATA * 0 0 0 0 0 0 QB
9660 DATA * 0 0 0 0 0 0 QC
9670 DATA * 0 0 0 0 0 0 QD
9680 DATA PLSR C D SER G X LOAD CLR- QA
9690 DATA * 0 0 0 0 0 0 QB
9700 DATA * 0 0 0 0 0 0 QC-
9710 DATA * 0 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 SC
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 TO1
9990 DATA AND EXINT ENUS 0 0 0 TO2
10000 DATA NAND CLK1 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 NKFF HI SPEED STB SPEED HI MXCTL MXCTL-
10100 DATA NKFF MXCTL LO STB HI T16 G T11
11100 DATA NKFF HI LO QC1 HI T16 0 T12
11120 DATA NKFF MXCTL- LO STB HI T16 0 T13
11130 DATA NKFF HI LO QC2 HI T16 G T14
11140 DATA NOR T11 QA1 0 0 0 T15
11150 DATAa NOR EXINT- QA1 0 0 0 T16
11160 DATAa NOR T13 QA2 0 0 0 T17
11170 DATAa NOR EXINT- QA2 0 0 0 T18
11180 DATA SHIFT OSC T15 T15 EXINT HI QA1
11190 DATA * 0 0 0 0 QB1
11200 DATA * 0 0 0 0 QC1
11210 DATA * 0 0 0 0 QB1
11220 DATA SHIFT OSC T17 T17 EXINT HI QA2
11230 DATA * 0 0 0 0 QB2
11240 DATA * 0 0 0 0 QC2
11250 DATA * 0 0 0 0 QB2
11260 DATA OR QA1 QA1 0 0 0 F11
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  
10370 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 ENGG 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 XXX
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.
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 ***

<table>
<thead>
<tr>
<th>NJKFF</th>
<th>J</th>
<th>K</th>
<th>CK-</th>
<th>PRESET</th>
<th>CLEAR</th>
<th>Q</th>
<th>Q-</th>
</tr>
</thead>
<tbody>
<tr>
<td>JKFF</td>
<td>J</td>
<td>K</td>
<td>CK-</td>
<td>CLEAR</td>
<td>PRESET</td>
<td>Q</td>
<td>Q-</td>
</tr>
<tr>
<td>NAND</td>
<td>CK-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*** Checking Macro NJKFF for Errors ***

*** Reading in Macro : SHIFT ***

<table>
<thead>
<tr>
<th>SHIFT</th>
<th>CK</th>
<th>J</th>
<th>K-</th>
<th>CLR-</th>
<th>SH/LD</th>
<th>QA</th>
<th>QA</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PLSR</td>
<td>0</td>
<td>DSER</td>
<td>CK</td>
<td>LOAD</td>
<td>CLR-</td>
<td>QA</td>
<td>QA</td>
</tr>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AND</td>
<td>J</td>
<td>K-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>DSER</td>
<td></td>
</tr>
<tr>
<td>NAND</td>
<td>SH/LD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LOAD</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

*** Checking Macro SHIFT for Errors ***

*** Reading in Macro : MUX ***

<table>
<thead>
<tr>
<th>MUX</th>
<th>1A</th>
<th>1B</th>
<th>G</th>
<th>SEL</th>
<th>1Y</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>2A</td>
<td>2B</td>
<td>0</td>
<td>0</td>
<td>2Y</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td>3A</td>
<td>3B</td>
<td>0</td>
<td>0</td>
<td>3Y</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td>4A</td>
<td>4B</td>
<td>0</td>
<td>0</td>
<td>4Y</td>
<td>0</td>
</tr>
<tr>
<td>AND</td>
<td>1A</td>
<td>SG</td>
<td>0</td>
<td>0</td>
<td>T1</td>
<td>0</td>
</tr>
<tr>
<td>AND</td>
<td>1A</td>
<td>SS</td>
<td>0</td>
<td>0</td>
<td>T2</td>
<td>0</td>
</tr>
<tr>
<td>AND</td>
<td>2A</td>
<td>SG</td>
<td>0</td>
<td>0</td>
<td>T3</td>
<td>0</td>
</tr>
<tr>
<td>AND</td>
<td>2A</td>
<td>SS</td>
<td>0</td>
<td>0</td>
<td>T4</td>
<td>0</td>
</tr>
<tr>
<td>AND</td>
<td>3A</td>
<td>SG</td>
<td>0</td>
<td>0</td>
<td>T5</td>
<td>0</td>
</tr>
<tr>
<td>AND</td>
<td>3A</td>
<td>SS</td>
<td>0</td>
<td>0</td>
<td>T6</td>
<td>0</td>
</tr>
<tr>
<td>AND</td>
<td>4A</td>
<td>SG</td>
<td>0</td>
<td>0</td>
<td>T7</td>
<td>0</td>
</tr>
<tr>
<td>AND</td>
<td>4A</td>
<td>SS</td>
<td>0</td>
<td>0</td>
<td>T8</td>
<td>0</td>
</tr>
<tr>
<td>OR</td>
<td>T1</td>
<td>T2</td>
<td>0</td>
<td>0</td>
<td>1Y</td>
<td>0</td>
</tr>
<tr>
<td>OR</td>
<td>T3</td>
<td>T4</td>
<td>0</td>
<td>0</td>
<td>2Y</td>
<td>0</td>
</tr>
<tr>
<td>OR</td>
<td>T5</td>
<td>T6</td>
<td>0</td>
<td>0</td>
<td>3Y</td>
<td>0</td>
</tr>
<tr>
<td>OR</td>
<td>T7</td>
<td>T8</td>
<td>0</td>
<td>0</td>
<td>4Y</td>
<td>0</td>
</tr>
<tr>
<td>NOR</td>
<td>0</td>
<td>SEL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NAND</td>
<td>SEL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Digital Simulator Output**

```
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 | EPOS | 0 | 0 | 0 | T02 0 |
| NAND | CLK1  | 0 | 0 | 0 | 0 | T03 0 |
| OR   | T02   | T02 | 0 | 0 | 0 | T04 0 |
| OR   | T03   | T03 | 0 | 0 | 0 | T05 0 |
| AND  | T04   | T04 | 0 | 0 | 0 | T06 0 |
| OR   | T05   | T05 | 0 | 0 | 0 | T07 0 |
| AND  | T06   | T06 | 0 | 0 | 0 | T08 0 |
| OR   | T07   | T07 | 0 | 0 | 0 | T09 0 |
| OR   | T08   | T08 | 0 | 0 | 0 | T10 0 |
| OR   | 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 |
| OR   | T11   | QA1 | 0 | 0 | 0 | T15 0 |
| OR   | EXINT- | QA1 | 0 | 0 | 0 | T16 0 |
| OR   | EXINT- | QA2 | 0 | 0 | 0 | T17 0 |
| OR   | T13   | QA2 | 0 | 0 | 0 | T18 0 |
| SHIFT | QSC | T15 | T15 | EXINT | HI | QA1 0 |
| *   | 0 | 0 | 0 | 0 | 0 | QB1 0 |
| *   | 0 | 0 | 0 | 0 | 0 | QC1 0 |
| SHI T | QSC | T17 | T17 | EXINT | HI | QA2 0 |
| *   | 0 | 0 | 0 | 0 | 0 | QB2 0 |
| *   | 0 | 0 | 0 | 0 | 0 | QC2 0 |
| OR   | QA1   | QA1 | 0 | 0 | 0 | FI1 0 |
| OR   | QB1   | QB1 | 0 | 0 | 0 | FI2 0 |
| OR   | QC1   | EXINT- | 0 | 0 | 0 | FI13 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 ***

```
J K F F  HI  SPEED  &51  HI  SPEED  MXCTL  MXCTL-  
NAND  STB  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  0  GB1  0
*  0  0  0  0  0  0  QC1  0
*  0  0  0  0  0  0  OD1  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  0  GB2  0
*  0  0  0  0  0  0  QC2  0
*  0  0  0  0  0  0  OD2  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  0  &65  0
AND  S1  &66  0  0  0  0  &67  0
AND  F12  &64  0  0  0  0  &68  0
AND  S2  &66  0  0  0  0  &69  0
AND  F13  &64  0  0  0  0  &70  0
AND  S3  &66  0  0  0  0  &71  0
AND  W11- &64  0  0  0  0  &72  0
AND  W12  &66  0  0  0  0  &73  0
OR  &65  &67  0  0  0  0  THE1  0
OR  &68  &69  0  0  0  0  THE2  0
OR  &70  &71  0  0  0  0  THE3  0
OR  &72  &73  0  0  0  0  WE2-  0
NOR  LO  SPEED  0  0  0  0  &64  0
NAND  SPEED  0  0  0  0  &74  0
NOR  LO  &74  0  0  0  0  &66  0
END
### Digital Simulator Output

#### Expanded Logic Description

<table>
<thead>
<tr>
<th>No.</th>
<th>Type</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>NAND</td>
<td>EXINT</td>
<td>EXINT-</td>
</tr>
<tr>
<td>(2)</td>
<td>AND</td>
<td>EXINT</td>
<td>ENEG</td>
</tr>
<tr>
<td>(3)</td>
<td>AND</td>
<td>EXINT</td>
<td>EPOS</td>
</tr>
<tr>
<td>(4)</td>
<td>NAND</td>
<td>CLK1</td>
<td></td>
</tr>
<tr>
<td>(5)</td>
<td>NOR</td>
<td>T02</td>
<td>T02</td>
</tr>
<tr>
<td>(6)</td>
<td>NOR</td>
<td>T03</td>
<td>T03</td>
</tr>
<tr>
<td>(7)</td>
<td>OR</td>
<td>T05</td>
<td>T05</td>
</tr>
<tr>
<td>(8)</td>
<td>AND</td>
<td>T01</td>
<td>T06</td>
</tr>
<tr>
<td>(9)</td>
<td>OR</td>
<td>T06</td>
<td>T04</td>
</tr>
<tr>
<td>(10)</td>
<td>AND</td>
<td>T03</td>
<td>T07</td>
</tr>
<tr>
<td>(11)</td>
<td>OR</td>
<td>T09</td>
<td>T03</td>
</tr>
<tr>
<td>(12)</td>
<td>NO</td>
<td>T09</td>
<td>QA1</td>
</tr>
<tr>
<td>(13)</td>
<td>NO</td>
<td>T11</td>
<td>QA2</td>
</tr>
<tr>
<td>(14)</td>
<td>NO</td>
<td>EXINT-</td>
<td>QA2</td>
</tr>
<tr>
<td>(15)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(16)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(17)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(18)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(19)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(20)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(21)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(22)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(23)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(24)</td>
<td>OR</td>
<td>QA2</td>
<td>QA1</td>
</tr>
<tr>
<td>(25)</td>
<td>JKFF</td>
<td>HI</td>
<td>SPEED</td>
</tr>
<tr>
<td>(26)</td>
<td>NAND</td>
<td>STB</td>
<td></td>
</tr>
<tr>
<td>(27)</td>
<td>JKF</td>
<td>MXCTL-</td>
<td>L0</td>
</tr>
<tr>
<td>(28)</td>
<td>NAND</td>
<td>STB</td>
<td></td>
</tr>
<tr>
<td>(29)</td>
<td>JKF</td>
<td>HI</td>
<td>$\delta$4</td>
</tr>
<tr>
<td>(30)</td>
<td>NAND</td>
<td>QCL</td>
<td></td>
</tr>
<tr>
<td>(31)</td>
<td>JKF</td>
<td>MXCTL-</td>
<td>L0</td>
</tr>
<tr>
<td>(32)</td>
<td>NAND</td>
<td>STB</td>
<td></td>
</tr>
<tr>
<td>(33)</td>
<td>JKF</td>
<td>HI</td>
<td>$\delta$6</td>
</tr>
<tr>
<td>(34)</td>
<td>NAND</td>
<td>QCL</td>
<td></td>
</tr>
<tr>
<td>(35)</td>
<td>PLSR</td>
<td></td>
<td>$\delta$8</td>
</tr>
<tr>
<td>(36)</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(37)</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(38)</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(39)</td>
<td>AND</td>
<td>HI</td>
<td></td>
</tr>
<tr>
<td>(40)</td>
<td>PLSR</td>
<td></td>
<td>$\delta$13</td>
</tr>
<tr>
<td>(41)</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(42)</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(43)</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(44)</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(45)</td>
<td>AND</td>
<td>HI</td>
<td></td>
</tr>
<tr>
<td>(46)</td>
<td>AND</td>
<td>FI1</td>
<td>$\delta$4</td>
</tr>
<tr>
<td>(47)</td>
<td>AND</td>
<td>S1</td>
<td>$\delta$6</td>
</tr>
<tr>
<td>(48)</td>
<td>AND</td>
<td>FI2</td>
<td>$\delta$8</td>
</tr>
<tr>
<td>(49)</td>
<td>AND</td>
<td>S2</td>
<td>$\delta$10</td>
</tr>
<tr>
<td>(50)</td>
<td>AND</td>
<td>FI3</td>
<td>$\delta$12</td>
</tr>
<tr>
<td>(51)</td>
<td>AND</td>
<td>S3</td>
<td>$\delta$14</td>
</tr>
<tr>
<td>(52)</td>
<td>AND</td>
<td>WE1-</td>
<td>$\delta$16</td>
</tr>
<tr>
<td>(53)</td>
<td>AND</td>
<td>WE21</td>
<td>$\delta$18</td>
</tr>
<tr>
<td>(54)</td>
<td>OR</td>
<td>$\delta$19</td>
<td></td>
</tr>
<tr>
<td>(55)</td>
<td>OR</td>
<td>$\delta$20</td>
<td></td>
</tr>
</tbody>
</table>
**Digital Simulator Output**

```
(57) OR &70 &71 . . . THET1 .
(58) OR &72 &73 . . . WE2- .
(59) NOR LO SPEED . . . &64 .
(60) NAND SPEED . . . &74 .
(61) NOR LO &74 . . . &66 .
END . . . .
```

*** Read External Stimuli ***

<table>
<thead>
<tr>
<th>No.</th>
<th>Type</th>
<th>Name</th>
<th>Val</th>
<th>Ben</th>
<th>Os1</th>
<th>Os2</th>
<th>Os3</th>
<th>Os4</th>
<th>Os5</th>
<th>Os6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>A</td>
<td>HI</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(2)</td>
<td>A</td>
<td>LO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(3)</td>
<td>P</td>
<td>OSC</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(4)</td>
<td>P</td>
<td>CLK1</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(5)</td>
<td>A</td>
<td>SPEED</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(6)</td>
<td>A</td>
<td>EXINT</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(7)</td>
<td>A</td>
<td>ENEG</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(8)</td>
<td>A</td>
<td>EPOS</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*** Checking for Error ***

*** Read Nodes To Sample ***

```
SPEED EXINT OSC CLK1 STB MXCTL FI1 F12 F13 WE2- THET1 THET2 THET3 WE2-
END
```

*** Setting Up Simulation ***

*** Build Selective Trace Table ***

*** Load Table ***

**NODE DEFINED**

```
<table>
<thead>
<tr>
<th>NODE</th>
<th>DEFINED</th>
<th>DRIVEN ELEMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK1</td>
<td>[ EXT 4 ]</td>
<td>4</td>
</tr>
<tr>
<td>ENEG</td>
<td>[ EXT 7 ]</td>
<td>2</td>
</tr>
<tr>
<td>EPOS</td>
<td>[ EXT 8 ]</td>
<td>3</td>
</tr>
<tr>
<td>EXINT</td>
<td>[ EXT 6 ]</td>
<td>1 2 3 35 41</td>
</tr>
<tr>
<td>EXINT-</td>
<td>[ Q 1 ]</td>
<td>14 16 19 23</td>
</tr>
<tr>
<td>FI1</td>
<td>[ Q 17 ]</td>
<td>47</td>
</tr>
<tr>
<td>FI2</td>
<td>[ Q 18 ]</td>
<td>49</td>
</tr>
<tr>
<td>FI3</td>
<td>[ Q 19 ]</td>
<td>31</td>
</tr>
<tr>
<td>HI</td>
<td>[ EXT 1 ]</td>
<td>25 25 27 29 29 31 33 33 40 46</td>
</tr>
<tr>
<td>LO</td>
<td>[ EXT 2 ]</td>
<td>27 29 31 33 59 61</td>
</tr>
<tr>
<td>MXCTL</td>
<td>[ Q 25 ]</td>
<td>27</td>
</tr>
<tr>
<td>MXCTL-</td>
<td>[ Q- 25 ]</td>
<td>31</td>
</tr>
<tr>
<td>OSC</td>
<td>[ EXT 3 ]</td>
<td>35 41</td>
</tr>
<tr>
<td>QA1</td>
<td>[ Q 35 ]</td>
<td>13 14 17 17 20</td>
</tr>
<tr>
<td>QA2</td>
<td>[ Q 41 ]</td>
<td>15 16 21 21 24</td>
</tr>
<tr>
<td>QB1</td>
<td>[ Q 36 ]</td>
<td>18 18</td>
</tr>
<tr>
<td>QB2</td>
<td>[ Q 42 ]</td>
<td>22 22</td>
</tr>
<tr>
<td>QC1</td>
<td>[ Q 37 ]</td>
<td>19 38</td>
</tr>
</tbody>
</table>
```
### Digital Simulator Output

<table>
<thead>
<tr>
<th>DC2</th>
<th>43</th>
<th>23 34</th>
</tr>
</thead>
<tbody>
<tr>
<td>QD1</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>QD2</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>21</td>
<td>48</td>
</tr>
<tr>
<td>S2</td>
<td>22</td>
<td>50</td>
</tr>
<tr>
<td>S3</td>
<td>23</td>
<td>52</td>
</tr>
<tr>
<td>SPEED</td>
<td>EXT</td>
<td>25 59 60</td>
</tr>
<tr>
<td>S18</td>
<td>12</td>
<td>26 32</td>
</tr>
<tr>
<td>T01</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>T02</td>
<td>3</td>
<td>5 5</td>
</tr>
<tr>
<td>T03</td>
<td>4</td>
<td>6 10 11</td>
</tr>
<tr>
<td>T04</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>T05</td>
<td>6</td>
<td>7 7</td>
</tr>
<tr>
<td>T06</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>T07</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>T08</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>T09</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>T10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>T11</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>T12</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>T13</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>T14</td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>T15</td>
<td>13</td>
<td>39 39</td>
</tr>
<tr>
<td>T16</td>
<td>14</td>
<td>27 29</td>
</tr>
<tr>
<td>T17</td>
<td>15</td>
<td>45 45</td>
</tr>
<tr>
<td>T18</td>
<td>16</td>
<td>21 23</td>
</tr>
<tr>
<td>THE1</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>THE2</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>THE3</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>WE1-</td>
<td>20</td>
<td>53</td>
</tr>
<tr>
<td>WE2-</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>WE21</td>
<td>24</td>
<td>54</td>
</tr>
<tr>
<td>&amp;51</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>&amp;52</td>
<td>28</td>
<td>27</td>
</tr>
<tr>
<td>&amp;53</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>&amp;54</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>&amp;55</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>&amp;56</td>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td>&amp;57</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>&amp;58</td>
<td>34</td>
<td>33</td>
</tr>
<tr>
<td>&amp;59</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>&amp;6</td>
<td>39</td>
<td>35</td>
</tr>
<tr>
<td>&amp;61</td>
<td>40</td>
<td>35</td>
</tr>
<tr>
<td>&amp;62</td>
<td>45</td>
<td>41</td>
</tr>
<tr>
<td>&amp;63</td>
<td>46</td>
<td>41</td>
</tr>
<tr>
<td>&amp;64</td>
<td>59</td>
<td>47 49 51 53</td>
</tr>
<tr>
<td>&amp;65</td>
<td>47</td>
<td>55</td>
</tr>
<tr>
<td>&amp;66</td>
<td>61</td>
<td>48 50 52 54</td>
</tr>
<tr>
<td>&amp;67</td>
<td>48</td>
<td>55</td>
</tr>
<tr>
<td>&amp;68</td>
<td>49</td>
<td>56</td>
</tr>
<tr>
<td>&amp;69</td>
<td>50</td>
<td>56</td>
</tr>
<tr>
<td>&amp;70</td>
<td>51</td>
<td>57</td>
</tr>
<tr>
<td>&amp;71</td>
<td>52</td>
<td>57</td>
</tr>
<tr>
<td>&amp;72</td>
<td>53</td>
<td>56</td>
</tr>
<tr>
<td>&amp;73</td>
<td>54</td>
<td>58</td>
</tr>
<tr>
<td>&amp;74</td>
<td>60</td>
<td>61</td>
</tr>
</tbody>
</table>
Digital Simulator Output

Default for Start & Increment ? N
Enter Sample Start Time ? 0
Enter Increment Time ? 5

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>125</th>
<th>150</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEED</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>EXINT</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>OSC</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CLK1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>STB</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>MXCTL</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F11</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F12</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F13</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE1-</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET2</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET3</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE2-</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>170</th>
<th>200</th>
<th>225</th>
<th>250</th>
<th>275</th>
<th>300</th>
<th>325</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEED</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>EXINT</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>OSC</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CLK1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>STB</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>MXCTL</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F11</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F12</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F13</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE1-</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET2</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET3</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE2-</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>345</th>
<th>200</th>
<th>225</th>
<th>250</th>
<th>275</th>
<th>300</th>
<th>325</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEED</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>EXINT</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>OSC</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CLK1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>STB</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>MXCTL</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F11</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F12</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F13</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE1-</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET2</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET3</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE2-</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>520</th>
<th>350</th>
<th>375</th>
<th>400</th>
<th>425</th>
<th>450</th>
<th>475</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEED</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>EXINT</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>OSC</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CLK1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>STB</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>MXCTL</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F11</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F12</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>F13</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE1-</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET2</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>THET3</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE2-</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

SPEED (1) -> (.)
## Digital Simulator Output

### Time 695

<table>
<thead>
<tr>
<th>Speed</th>
<th>EXINT</th>
<th>OSC</th>
<th>CLK1</th>
<th>STB</th>
<th>MXCTL</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>WE1-</th>
<th>THET1</th>
<th>THET2</th>
<th>THET3</th>
<th>WE2-</th>
</tr>
</thead>
<tbody>
<tr>
<td>525</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>550</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>575</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>625</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>650</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>675</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Time 870

<table>
<thead>
<tr>
<th>Speed</th>
<th>EXINT</th>
<th>OSC</th>
<th>CLK1</th>
<th>STB</th>
<th>MXCTL</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>WE1-</th>
<th>THET1</th>
<th>THET2</th>
<th>THET3</th>
<th>WE2-</th>
</tr>
</thead>
<tbody>
<tr>
<td>700</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>725</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>750</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>775</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>825</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>850</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Time 1045

<table>
<thead>
<tr>
<th>Speed</th>
<th>EXINT</th>
<th>OSC</th>
<th>CLK1</th>
<th>STB</th>
<th>MXCTL</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>WE1-</th>
<th>THET1</th>
<th>THET2</th>
<th>THET3</th>
<th>WE2-</th>
</tr>
</thead>
<tbody>
<tr>
<td>875</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>900</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>925</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>950</td>
<td>1</td>
<td>1..1</td>
<td>1..1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>975</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1025</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 86/12A 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 th
* pattern input from the test. The reason fo.
* 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 - C: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=8/a/d8
* MOV ES, AX ; Clear Extra Segment Register
s0:1005=b8/e0
* CLD ; Clear Direction Flag
s0:1007=fc
* MOV SI, 2000H ; Set Source of Pattern
s0:1008=bc/0/20
* MOV DI, 8000H ; Set Destination Test Hardware
s0:100b=bf/00/80
* MOV CX, 0800H ; Set Length
s0:100e=bf/00/08
* MLOOP MOVSW ; Memory to Memory Move
s0:1011=e5
* LOOP MLOOP ; Loop Until End
s0:1012=e2/fd
* MOV SI, 8000H ; Set Source Test Hardware
s0:1014=bc/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=e3
* LOOP ELOOP ; Loop Until End
s0:101e=e2/Ed
* JMP OFE00:06F1 ; Return to Monitor
s0:1020=ea/f1/06/00/fe

* ===========
* Finished Load
* ===========
This Command File loads Test Patterns for the HM6146LP 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
* 0 = Low
* ? = Undefined
* X = Defined
* = Tri-State

READ

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CS/</td>
<td>11111</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE/</td>
<td>11111</td>
<td>11111</td>
<td>11111</td>
</tr>
<tr>
<td>ADDR</td>
<td>????</td>
<td>XXXXX</td>
<td>XXXXX</td>
</tr>
<tr>
<td>DATA</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
</tbody>
</table>

WRITE

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CS/</td>
<td>11111</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>WE/</td>
<td>11111</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ADDR</td>
<td>????</td>
<td>XXXXX</td>
<td>XXXXX</td>
</tr>
<tr>
<td>DATA</td>
<td>-----</td>
<td>XXXXX</td>
<td>XXXXX</td>
</tr>
</tbody>
</table>

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/ | I01 | I02 | I03 | I04 |

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
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
12 12 11 11 10 10 9 9 16 16 15 15 14 14 13 13

PATTERN RETURNED DATA

s = 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 babababa bsbabab
4 4 3 3 2 2 1 1 8 8 7 7 6 6 5 5

0:3800 babababa bababab
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$fffe,0:2002
m0:2800$fffe,0:2802

Now Load Test Pattern

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

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

Write 000,f
s0:2814=aa/aa/aa/aa/aa/aa
s0:2814=fa/00/aa/aa/aa/aa
* <--- Read 000
s0:201c=aa/aa/aa/aa/aa/aa
s0:281c=00/ba/00/ff/00/00

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

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

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

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

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

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

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

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

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

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

* <--- Write 3FF,1
s0:2074=aa/ff/ff/ff/ff/ff
s0:2874=fa/00/bf/00/bf/00/00

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

* <--- Write 3FF,1
s0:2084=aa/ff/ff/ff/ff/ff
s0:2884=fa/00/bf/00/bf/00/00
s0:2884=fa/00/af/af/af/af/00

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

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

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

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

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

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

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

* Get Access
* ---------
oc100,0110

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

*****

* Reset State
* ------------
oc100,1500
c0100,1501
c0100,0110

* Set System Looping
* -------------------
oc100,7422

* -----------

* 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**

- 000, 0000
- 001, 0010
- 010, 0100
- 011, 0110
- 100, 1000
- 101, 1010
- 110, 1100
- 111, 1110

**Test Run**

- 000, 0000
- 001, 0010
- 010, 0100
- 011, 0110
- 100, 1000
- 101, 1010
- 110, 1100
- 111, 1110

**Set Access**

- 000, 0000
- 001, 0010
- 010, 0100
- 011, 0110
- 100, 1000
- 101, 1010
- 110, 1100
- 111, 1110

**Set Pattern Returned**

- 000, 0000
- 001, 0010
- 010, 0100
- 011, 0110
- 100, 1000
- 101, 1010
- 110, 1100
- 111, 1110

**Read Good Pattern**

- 000, 0000
- 001, 0010
- 010, 0100
- 011, 0110
- 100, 1000
- 101, 1010
- 110, 1100
- 111, 1110

**Reset State**

- 000, 0000
- 001, 0010
- 010, 0100
- 011, 0110
- 100, 1000
- 101, 1010
- 110, 1100
- 111, 1110

**Set System Looping**

- 000, 0000
- 001, 0010
- 010, 0100
- 011, 0110
- 100, 1000
- 101, 1010
- 110, 1100
- 111, 1110
**Retest Command File**

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.

---

**React State**

```
owl00,1500
owl00,1501
owl00,0110
```

**Do Test Run**

```
owl00,5422
```

**Get Access**

```
owl00,0110
```

**Get Pattern Returned**

```
g0:1000
```

**Compare with Good Pattern**

```
c0:3000$d0,0:4000
```

```
c0:3800$d0,0:4000
```

**Reset State**

```
owl00,1500
owl00,1501
owl00,0110
```

**Set System Looping**

```
owl00,7422
```
* End of Re-Test
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 000,5
d0:3008#8
d0:3008#8

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

* => Write 000,1
d0:3018#8
d0:3018#8

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

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

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

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

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

* => Write 000,4
* ====> Read ODD
  d0:3048#8
  d0:3848#8

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

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

* ====> Write 3FF, U
  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, 0
  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 3F7, 8
   d0: 3018#8
   d0: 381A#8

* —> Read 3FF
   d0: 30c0#8
   d0: 38c0#8

* ================
* End of Display
* ================

This Command File Sets the Test Hardware 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
3700H + ( 2 * Lowest 10 bits of number from
status register). This is for the lower
Tri-State bits or Lower 16 Hi-State bits,
\( v, b1 \ldots a8, b0 \). For the Higher 8 Tri-State,
\( b1, \ldots 16 \) Hi-State bits the Address is
3800H + ( 2 * Lowest 17 bits of number read).

Load Tri-State Pattern

Set Address

Load Test Hardware Rx

Reset MU0

Set Logic Analyser Being
* 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 Analyzer
  * ------------------------
* This Command File Starts the logic analyiner to read on Positive Edge and With Speed = 0. Speed = 0 means the Test Hardware is in the * non-multiplexing mode.

* Reset Status
* ----------------
  ow100,1501
  ow100,1901
  ow100,0116

* Set LA Gr:
* ----------------
  ow100,7422
  10<1w100>

* 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
  * ---------------
  ow109,linu
  lw100

* Do Move Routine
  * ---------------
  go:1000

*****

* Display Pattern Input
  * ---------------
  do:2000$30
  do:2800$30

* Display Pattern Returned
  * ---------------
  do:3000$30
  do: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
  * ------------
  ow100,1500
  ow100,1501
  ow100,0110

  * Zero Pattern Memory
  * ------------
  sg:2000=00
  m0:2000f1.000,0:2001
  *****

  * Move Pattern into Test Hardware
  * -----------------------------
  sg:1000
  *****

  * Output First location (Into Tri-State)
  * -------------------------------------
  ow100,5400
  ow100,5401

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

  * ==============
  * End of Reset
  * =============
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 Analyzer mode, where it waits for the trigger and samples on the positive edge of the clock.

* Load Move Program

s0:1000=88/00/00
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=e5
s0:1012=e2/7d
s0:1014=be/00/80
s0:1017=bf/00/30
s0:101a=b9/00/08
s0:101d=a5
s0:101e=e2/7d
s0:1020=es/f1/06/00/fe

* Load Oscillating Pattern

s0:2000=sa/af/ff
s0:2001=1000,0:2004

* Get Access

ov100,0110

* Load Test Hardware Ram

s0:1000

* Reset State
ALL.TXY

ow100,1500
ow100,1501
ow100,0110

* Set Logic Analyser Going

ow100,7422
100000<ow100>

* End of All

*
INTRODUCTION

This is the sequence of commands and the reply from the Test-Hardware used to test a 4Mx1048-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 HN6148LP RAM *
| * *
| *--------------------------------------------------*
| *
| * Load Move Routine *
| * *--------------------------------------------------*

._--------------------------------------------------*
| * "Move" Program *
| * *--------------------------------------------------*
| *
| * The Load File Line: Look at the "Move" Program *
| * This program which starts at offset 0000h *
| * the Pattern moves to the SS:0000h address and *
| * loads it into the Hardware's pattern *
| * note to at the same time it loads the Test *
| * until a return pattern can be set *
| * *
| * to clear the moves. The reason for *
| * the load to set the Test Hardware's Memory *
| * over the Pattern output bank when writing to *
| * it, and it over the Pattern input bank when *
| * really needs it. If you write to the Test *
| * Hardware and read it back you get a different *
| * number which continues the Move and Substitute *
| * command of the move: in other words *
| *
| * 0124Fh - 012EFF Pattern Input Buffer Area *
| * 013500h - 013EFF Pattern Returned Buffer Area *
| * 014000h - 014EFF Test Hardware Address Area *
| *
| *--------------------------------------------------*
| *
| MOV AX,0000h ; Clear AX
| s0:1000=88/00/00
| *
| MOV DS,AX ; Clear Data Segment Register
| s0:1003=86/d6
| *
| MOV ES,AX ; Clear Extra Segment Register
| s0:1005=86/c6
| *
| CLO ; Clear Direction Flag
| s0:1007=fc
| *
| MOV SI,1000h ; Set Source of Pattern
| s0:100b=be/00/20
| *
| MOV DI,8000h ; Set Destination Test Hardware
| s0:100f=bf/00/80
| *
| MOV CX,08000h ; Set Length
HM6148LP RAM Test Example

2-24-84 PAGE 2

; 60:100e=b9/00/08
; MLOOP NOVSW ; Memory to Memory Move
; 60:1011=a5
; LOOP MLOOP ; Loop Until End
; 60:1012=a2/1d
; MOV SI,8000H ; Set Source Test Hardware
; 60:1014=be/00/80
; MOV DI,3000H ; Set Destination Pattern Returnet
; 60:1017=bf/00/30
; MOV CX,0800H ; Set Length
; 60:101a=b9/00/08
; LOOP ELOOP ; Memory to Memory Move
; 60:101d=a5
; MOV SI, 8000H ; Memory to Memory Move
; 60:101e=a2/1d
; JMP OFEO:00F1 ; Return to Monitor
; 60:1020=af/f1/06/00/f

; ===============
; * Finished Load
; ===============

; ================
; * Load Ram Test
; ================

; ************************************************************************
; * This Command File loads Test Patterns for
; * the HM6148L 1K by 4 Static Ram Chip. This
; * test is not 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
; * 0 = Low
; * ? = Undefined
; * X = Defined
; * - = Tri-State

; READ
HM6148LBP RAM Test Example

WRITE

RAM LINE DEFINITIONS

PATTERN OUTPUT CONTROL

PATTERN RETURNED DATA

a = Low Input (Referenced to 0.8 Volts)
b = High Input (Referenced to 2.4 Volts)
HH6148LP RAM Test Example

.** Setup Default Pattern

.** ---------------------

.s0:2000=aa/aa
.s0:2800=fa/00
.m0:2000=fe,0:2002
.*****
.m0:2800=fe,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/aa/aa

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

.** <<< Write 000,F
.s0:2114=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2814=fa/00/aa/ff/aa/ff/ff/ff

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

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

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

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

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

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

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

.** <<< Write 000,8
.s0:2054=aa/aa/aa/aa/aa/aa/aa/aa

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

.** <<< Write 000,9
.s0:2064=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2864=fa/00/aa/ba/aa/ba/ba/00

.** <<< Read 000
.s0:206c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:286c=fa/00/ba/00/ba/00/fe/00

.** <<< Write 000,10
.s0:2074=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2874=fa/00/aa/ae/aa/ae/ae/ae

.** <<< Read 000
.s0:207c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:287c=fa/00/ba/00/ba/00/fe/00

.** <<< Write 000,11
.s0:2084=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2884=fa/00/aa/ba/aa/ba/ba/00

.** <<< Read 000
.s0:208c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:288c=fa/00/ba/00/ba/00/fe/00

.** <<< Write 000,12
.s0:2094=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2894=fa/00/aa/ae/aa/ae/ae/ae

.** <<< Read 000
.s0:209c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:289c=fa/00/ba/00/ba/00/fe/00

.** <<< Write 000,13
.s0:20a4=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28a4=fa/00/aa/ba/aa/ba/ba/00

.** <<< Read 000
.s0:20ac=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28ac=fa/00/ba/00/ba/00/fe/00

.** <<< Write 000,14
.s0:20b4=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28b4=fa/00/aa/ae/aa/ae/ae/ae

.** <<< Read 000
.s0:20bc=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28bc=fa/00/ba/00/ba/00/fe/00

.** <<< Write 000,15
.s0:20c4=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28c4=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:20cc=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28cc=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,F
.s0:20d4=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28d4=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:20dc=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28dc=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,0
.s0:20e4=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28e4=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:20ec=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28ec=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,1
.s0:20f4=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28f4=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:20fc=aa/aa/aa/aa/aa/aa/aa/aa
.s0:28fc=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,2
.s0:2104=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2814=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:210c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:281c=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,3
.s0:2114=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2814=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:211c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:281c=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,4
.s0:2124=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2824=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:212c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:282c=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,5
.s0:2134=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2834=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:213c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:283c=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,6
.s0:2144=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2844=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:214c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:284c=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,7
.s0:2154=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2854=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Read 000
.s0:215c=aa/aa/aa/aa/aa/aa/aa/aa
.s0:285c=fa/00/ae/aa/ae/ae/ae/ae

.** <<< Write 000,8
.s0:2164=aa/aa/aa/aa/aa/aa/aa/aa
.s0:2864=fa/00/ae/aa/ae/ae/ae/ae
.0:2834=fa/00/aa/aa/aa/aa/fa/00

* <==== Read 000
.0:205c=aa/aa/aa/aa/aa/aa/aa
.0:285c=fe/00/aa/00/ba/00/fa/00

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

* <==== Read 3FF
.0:2074=aa/aa/ff/ff/ff/ff/aa
.0:2874=fe/00/af/ff/af/ff/fa/00

* <==== Write 3FF,1
.0:2084=aa/aa/ff/ff/ff/ff/aa
.0:2884=fe/00/af/ab/af/ab/fa/00

* <==== Read 3FF
.0:2094=aa/aa/ff/ff/ff/ff/aa
.0:2894=fe/00/af/ae/af/ae/fa/00

* <==== Write 3FF,2
.0:20a4=aa/aa/ff/ff/ff/ff/aa
.0:28a4=fe/00/bf/00/bf/00/fa/00

* <==== Read 3FF
.0:20b4=aa/aa/ff/ff/ff/ff/aa
.0:28b4=fe/00/bf/00/bf/00/fa/00

* <==== Write 3FF,3
.0:20c4=aa/aa/ff/ff/ff/ff/aa
.0:28c4=fe/00/bf/00/bf/00/fa/00

* <==== Read 3FF
.0:20d4=aa/aa/ff/ff/ff/ff/aa
.0:28d4=fe/00/bf/00/bf/00/fa/00

* <==== Write 3FF,4
.0:20e4=aa/aa/ff/ff/ff/ff/aa
.0:28e4=fe/00/bf/00/bf/00/fa/00

* <==== Read 3FF
.0:20f4=aa/aa/ff/ff/ff/ff/aa
.0:28f4=fe/00/bf/00/bf/00/fa/00

* <==== Write 3FF,5
.0:20g4=aa/aa/ff/ff/ff/ff/aa
.0:28g4=fe/00/bf/00/bf/00/fa/00

* <==== Read 3FF
.0:20h4=aa/aa/ff/ff/ff/ff/aa
.0:28h4=fe/00/bf/00/bf/00/fa/00

* <==== Write 3FF,6
.0:20i4=aa/aa/ff/ff/ff/ff/aa
.0:28i4=fe/00/bf/00/bf/00/fa/00

* <==== Read 3FF
.0:20j4=aa/aa/ff/ff/ff/ff/aa
.0:28j4=fe/00/bf/00/bf/00/fa/00
* Get Access
  
* ----
  
.owl00,0110
  
* Load Test Hardware Ram
  
* -----------------------------------
  
.g0:1000

ISBC 86/12 MONITOR V2.0

* ****

* Reset State
  
* -------
  
.owl00,1500
  
.owl00,1501
  
.owl00,0110

* Set System Looping
  
* ----------------
  
.owl00,7422

* ================

* 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
  
* -------
  
.owl00,1500
  
.owl00,1501
  
.owl00,0110

* Do Test Run
  
* -------
  
.owl00,5422
\* Get Access
\* --------------
\#0:1000

\* Get Pattern Returned
\* ----------------------
\#0:1000

\* ISC 86/12 MONITOR V2.0
\* ********
\* Save Good Pattern
\* ----------------
\#0:3000\#100,0:4000

\* Reset State
\* ------------
\#0:1000,1500
\#0:100,1501
\#0:100,0110

\* Set System Looping
\* -------------------
\#0:100,7422

\* 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
\* -----------
\#0:1000,1500
• Do Test Run
  * 
  .owl00,1501
  .owl00,0110
  
  * Get Access
  * 
  .owl00,0110
  
  * Get Pattern Returned
  * 
  .owl00,0110
  .owl00,1501
  .owl00,1500
  .owl00,0110
  .owl00,3422
  
  * Set System Looping
  * 
  .owl00,7422
  
  * End of Re-Test
  * 
  
  **************************************************

* 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 FO 55 00 00 00 FO 55

Read 000
.d0:3010#8
u000:3010 00 00 00 00 00 00 00 00
.d0:3810#8
u000:3810 FO 55 30 00 30 00 FO 55

Write 000,F
.d0:3018#8
1.1000:3018 00 00 00 00 00 00 00 00
.d0:3818#8
1.1000:3818 FO 55 00 FF 00 FF FO 55

Read 000
.d0:3020#8
0000:3020 00 00 00 00 00 00 00 00
.d0:3820#8
0000:3820 FO 55 30 03 30 03 FO 55

Write 000, 1
.d0:3028#8
0000:3028 00 00 00 00 00 00 00 00
.d0:3828#8
0000:3828 FO 55 00 03 00 03 FO 55

Read 000
.d0:3030#8
0000:3030 00 00 00 00 00 00 00 00
.d0:3830#8
0000:3830 FO 55 30 03 30 03 FO 55

Write 000, 2
.d0:3038#8
0000:3038 00 00 00 00 00 00 00 00
.d0:3838#8
0000:3838 FO 55 00 00 00 OC FO 55

Read 000
.d0:3040#8
0000:3040 00 00 00 00 00 00 00 00

........
HM6148LP RAM Test Example

-00:3040 FO 55 30 OC 30 OC FO 55
  * UG.O..U*
  * === Write 000,4
  .d0:3048#F
  0000:3048 00 00 00 00 00 00 00
  .d0:3848#F
  0000:3848 FO 55 00 30 00 30 FO 55
  *U.O.O.U*
  * === Read 000
  .d0:3050#F
  0000:3050 00 00 00 00 00 00 00
  .d0:3850#F
  0000:3850 FO 55 30 30 30 FO 55
  *U.O.O.U*
  * === Write 000,6
  .d0:3058#F
  0000:3058 00 00 00 00 00 00 00
  .d0:3858#F
  0000:3858 FO 55 00 00 00 00 FO 55
  *U.O.O.U*
  * === Read 000
  .d0:3060#F
  0000:3060 00 00 00 00 00 00 00
  .d0:3860#F
  0000:3860 FO 55 30 30 30 FO 55
  *U.O.O.U*
  * === Write 3FF,0
  .d0:3068#F
  0000:3068 00 00 FF FF FF FF 00 00
  .d0:3868#F
  0000:3868 FO 55 0F OF 0F 0F FO 55
  *U.U.U*
  * === Read 3FF
  .d0:3070#F
  0000:3070 00 00 FF FF FF FF 00 00
  .d0:3870#F
  0000:3870 FO 55 3F 00 3F 00 FO 55
  *U?.?..U*
  * === Write 3FF,F
  .d0:3078#F
  0000:3078 00 00 FF FF FF FF 00 00
  .d0:3878#F
  0000:3878 FO 55 OF OF FF FO 55
  *U.U.U*
  * === Read 3FF
  .d0:3080#F
  0000:3080 00 00 FF FF FF FF 00 00
  .d0:3880#F
  0000:3880 FO 55 3F FF 3F FF FO 55
  *U?.?..U*
  * === Write 3FF,1
  .d0:3088#F
  0000:3088 00 00 FF FF FF FF 00 00
  *........*
HM6146LP RAM Test Example

2-24-84 PAGE 11

.d0:3088#8
0000:3888 F0 55 OF 03 OF 03 F0 55
.* U....U*

.* ===> Read 3FF
.d0:3090#8
0000:3090 00 00 FF FF FF 00 00
*. U??....U*

.* ===> Write 3FF, 2
.d0:3098#8
0000:3898 00 00 FF FF FF 00 00
*. U??....U*

.* ===> Read 3FF
.d0:30A0#8
0000:30A0 00 00 FF FF FF FF 00 00
*. U??....U*

.* ===> Write 3FF, 4
.d0:38A8#8
0000:38A8 00 00 FF FF FF FF 00 00
*. U....U*

.* ===> Read 3FF
.d0:30B0#8
0000:30B0 00 00 FF FF FF FF 00 00
*. U???

.* ===> Write 3FF, 8
.d0:30B8#8
0000:38B8 00 00 FF FF FF FF 00 00
*. U??....U*

.* ===> Read 3FF
.d0:30C0#8
0000:30C0 00 00 FF FF FF FF 00 00
*. U??....U*

.* ===> Read 3FF
.d0:30D0#8
0000:38D0 00 00 FF FF FF FF 00 00
*. 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

Do Test Run

Get Access

Get Pattern Returned

==============================================

Compare with Good Pattern

( No Differences Lower 8 Bits )
HM6148LP RAM Test Example 2-24-84 PAGE 13

0000:384B 00 30 0000:484B
0000:384D 00 30 0000:484D
0000:3853 00 30 0000:4853
0000:3855 00 30 0000:4855
0000:385B 00 CO 0000:485B
0000:385D 00 CO 0000:485D
0000:3863 00 CO 0000:4863
0000:3865 00 CO 0000:4865
0000:38AB 00 30 0000:48AB
0000:38AD 00 30 0000:48AD
0000:38BB 00 30 0000:48BB
0000:38BD 00 30 0000:48BD
0000:38C3 00 CO 0000:48C3
0000:38C5 00 CO 0000:48C5

*****

* Reset State
* -----------
:ow100,1500
:ow200,1501
: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.

**************************************************
HM6148LP RAM Test Example

* ==> Write 000,0
.d0:3008#8
0000:3008 00 00 00 00 00 00 00
* ....... *

* ==> Read 000
.d0:3010#8
0000:3010 00 00 00 00 00 00 00
* ....... *

* ==> Write 000,1
.d0:3018#8
0000:3018 00 00 00 00 00 00 00
* ....... *

* ===> Read 000
.d0:3020#8
0000:3020 00 00 00 00 00 00 00
* ....... *

* ==> Write 000,2
.d0:3028#8
0000:3028 00 00 00 00 00 00 00
* ....... *

* ===> Read 000
.d0:3030#8
0000:3030 00 00 00 00 00 00 00
* ....... *

* ==> Write 000,3
.d0:3038#8
0000:3038 00 00 00 00 00 00 00
* ....... *

* ===> Read 000
.d0:3040#8
0000:3040 00 00 00 00 00 00 00
* ....... *

* ==> Write 000,4
.d0:3048#8
0000:3048 00 00 00 00 00 00 00
* ....... *
HM6148LF RAM Test Example

* 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
* 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 00 00 00 00 00 00 00 00
* Should Be CO CO

* 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
* Should Be CO CO

* Write 3FF, 0
.d0:3068#8
0000:3068 00 00 00 00 00 00 00 00
.d0:3868#8
0000:3868 F0 55 0F 00 0F 00 F0 55
* Should Be 3F FF FF FF F0 55

* Read 3FF
.d0:3070#8
0000:3070 00 00 00 00 00 00 00 00
.d0:3870#8
0000:3870 F0 55 3F 00 3F 00 F0 55
* Should Be 3F FF FF FF F0 55

* Write 3FF, 7
.d0:3078#8
0000:3078 00 00 00 00 00 00 00 00
.d0:3878#8
0000:3878 05 00 0F FF FF F0 55
* Should Be 3F FF FF FF F0 55

* Read 3FF
.d0:3080#8
0000:3080 00 00 00 00 00 00 00 00
.d0:3880#8
0000:3880 F0 55 3F FF 3F FF F0 55
* Should Be 3F FF FF FF F0 55

* Write 3FF, 1
.d0:3088#8
0000:3088 00 00 00 00 00 00 00 00
.d0:3888#8
0000:3888 F0 55 3F FF 3F FF F0 55
* Should Be 3F FF FF FF F0 55
Write 3FF,1

Read 3FF

Write 3FF,2

Read 3FF

Write 3FF,4

Read 3FF

Write 3FF,8

Read 3FF
HM6148LP RAM Test Example

0000:3000 00 00 FF FF FF FF 00 00
.00:38c0#8
0000:38c0 FO 55 3F 00 3F 00 FO 55
.*
.* Should Be CO CO
.*
.* End of Display
.*

********************
.*
.* End of RAM Test
.*

********************

2-24-84 PAGE 17
APPENDIX 0

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.
Load Move Routine

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
* MOV DS, AX ; Clear Data Segment Register
* MOV ES, AX ; Clear Extra Segment Register
* CLD ; Clear Direction Flag
* MOV SI, 2000H ; Set Source of Pattern
* MOV DI, 8000H ; Set Destination Test Hardware
```
* Finished Load

* Run Logic Analyser Mode

* Low Tri-State Pattern
Logic Analyzer Example (Counter) 2-24-84 PAGE 3

*---------------------
.s0:2000=00
.w0:200001000,0:2001
***
.* Get Access
.* ----------------
.sw100,0:10
.wz100
.wA8A
.
.* Load Test Hardware Ram
.* ---------------------
.sg0:1000

ISBC 86/12 MONITOR V2.0
****
.
.* Reset State
.* ----------------
.sw100,1300
.sow100,1501
.sow100,0110
.
.* Set Logic Analyzer Going
.* -----------------------
.sw100,6c22
.100000<1w100>
.ADA7
.AFD7
.AF31
.AE87
.AD06
.AF34
.AEC4
.AD29
.AF91
.AE7
.AB44
.AD9E
.AC5E
.AG3F
.AF96
.AE6D
.A53F
.AD9B
.ACF1
.AC3D
.A5A4
.ADF5
.AD4A
.AC9D
.AFF5
.AFP4
CDAA (← Trigger Point)

1. Get Access
2. Get Pattern Returned
3. Display Returned Pattern (Lower 16 Bits)

SBC 86/12 MONITOR V2.0

0000:3000 C3 3C CC 3C CF 3C F0 3C F3 3C FC 3C FF 3C 00 3C
0000:3010 03 3C 00 3C OF 3C 3C 3C 3C 00 3C 00 3C
0000:3020 03 3C 00 3C OF 3C 3C 3C 3C 00 3C 00 3C
0000:3030 03 3C 00 3C OF 3C 3C 3C 3C 00 3C 00 3C
0000:3040 03 3C 00 3C OF 3C 3C 3C 3C 00 3C 00 3C
0000:3050 03 3C 00 3C OF 3C 3C 3C 3C 00 3C 00 3C
0000:3060 03 3C 00 3C OF 3C 3C 3C 3C 00 3C 00 3C
0000:3070 03 3C 00 3C OF 3C 3C 3C 3C 00 3C 00 3C
0000:3080 03 3C 00 3C OF 3C 3C 3C 3C 00 3C 00 3C
0000:3090 03 3C 00 3C OF 3C 3C 3C 3C 00 3C 00 3C

*Control C*

* Get Access

* Get Pattern Returned

* Display Returned Pattern (Lower 16 Bits)
Logic Analyser Example (Counter) 2-24-84

<table>
<thead>
<tr>
<th>Counter</th>
<th>Pattern</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000:30A0</td>
<td>C3 CF CC CP CP CP F0 CF F3 CF PC CF PF CF OO CF</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:30B0</td>
<td>C3 CP CC CP OP CP 30 CP 33 CP 3C CP 3P CF G0 F0</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:30C0</td>
<td>C3 F0 CF F0 CP F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0 F0</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:30D0</td>
<td>C3 FO CF FO CP FO FO FO FO FO FO FO FO FO FO FO FO</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:30E0</td>
<td>C3 CP CC CP CP F3 CP F3 CP F3 CP F3 CP F3 CP F3 CP F0</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:30F0</td>
<td>C3 CP CP CP CP F3 CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3100</td>
<td>C3 CP CC CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3110</td>
<td>C3 CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3120</td>
<td>C3 CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3130</td>
<td>C3 CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3140</td>
<td>C3 CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3150</td>
<td>C3 CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3160</td>
<td>C3 CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3170</td>
<td>C3 CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3180</td>
<td>C3 CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
<tr>
<td>0000:3190</td>
<td>C3 CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP CP</td>
<td>.........*</td>
<td>0.3.&lt;.??*</td>
</tr>
</tbody>
</table>

*Control C*
**Logic Analyzer Example (Counter)**

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
</table>
| 0:3000 | 3F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FFFF

---

**Note**: Trigger was on Value 00 00
.* End of Logic Analyser Example *
.*
**************************************************