A self-checking multiprocessor system for train-borne operation

This item was submitted to Loughborough University’s Institutional Repository by the/an author.

Additional Information:

• A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/13789

Publisher: © William George Marshall

Please cite the published version.
This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
A SELF-CHECKING MULTIPROCESSOR SYSTEM FOR
TRAIN-BORNE OPERATION

BY

WILLIAM GEORGE MARSHALL, B.Sc., D.I.S.

A doctoral thesis submitted in partial
fulfilment of the requirements for the
award of Doctor of Philosophy of
Loughborough University of Technology.


SUPERVISOR:-

W. FORSYTHE, M.Sc.

Department of Electronic and Electrical Engineering

© William George Marshall 1981
SYNOPSIS

Since the construction of the first electronic computer, forty years ago, reliability has been an important consideration in the minds of computer designers. However, the available technology limited early efforts at fault-tolerance, error-checking and fault recovery. The early valve and discrete transistor machines tended to be large and heavy, unsuitable for applications such as on-board automatic train control, even with no error detection and safety systems.

With improvements in component technology, vehicle-borne control electronics has become a practical reality, but until the emergence of large scale integrated circuits (LSI) constraints on fault detection systems remained. The review sections of this thesis trace the development of high-reliability computing techniques, in particular those applied to vehicle control.

Control designs are becoming more ambitious, and as microprocessors rival minicomputers for speed and instruction set power, multiprocessing at low cost is easily attainable. In this atmosphere of cheap computer power, the British Rail Automatic Train Operation (BRATO) project was born. Seven microprocessor systems were to run three programs to effect (almost) driverless train control and more. The research for this thesis involved examining the alternative methods for tackling the problems of safety and availability in a train-borne system.

The design and development of a multiprocessing module, Cyclone I will be described. Cyclone I contains three 16-bit microprocessors and three program stores and runs as a compact, self-checking multiprocessor. When fully developed, the design will be suitable not only for train systems, but for other applications requiring a high confidence level in the output control signals.
Acknowledgements

I would like to thank the following people:-

At the Railway Technical Centre, Derby:-

Dr. P. D. Thomas, my supervisor at BR for most of the project period. I spent many hours discussing BRATO, CYCLONE and railways in general with Pete, giving me much valuable knowledge on train automation. His effort in setting up the project is very much appreciated.

Mr P. W. Parkin, who played a major part in obtaining BR approval for the project.

Mr. D. Hutton, who was always ready to answer my questions on hardware, and to put up with me scrounging parts!

Mr. R. H. Evans, who took over my project supervision at BR when Pete left.

At Loughborough University:-

Mr. W. Forsythe, my university supervisor, without whose guidance and advice, this work would not have been possible.

Finally, I would like to extend my thanks to all friends and colleagues, both at Derby and at Loughborough who made three years' work so enjoyable.
3.4 HARDWARE STRUCTURE TO ACHIEVE RELIABILITY PERFORMANCE

3.4.1 Masking Redundancy
3.4.2 Duplicated Modules and Voting
3.4.3 Triple Modular Redundancy
3.4.4 System Partitioning
3.4.5 Fault Detection and System Reconfiguration
3.4.6 System Timing
3.4.7 System Busses
3.4.8 Voters and Switches

3.5 SOFTWARE TECHNIQUES TO ACHIEVE RELIABILITY PERFORMANCE

3.5.1 Code Checking While Running Programs
3.5.2 Self-Test While Not Running Programs
3.5.3 Rollback and Retry

3.6 SPECIAL HARDWARE RECOVERY TECHNIQUES

3.6.1 Watchdog Timers
3.6.2 Illegal Address Detection
3.6.3 AC Output Coupling

3.7 CONCLUSION

REFERENCES FOR CHAPTER 3

CHAPTER 4 : AUTOMATIC VEHICLE CONTROL BY MEANS OF ON-BOARD COMPUTERS

4.1 COMPARISON OF MOBILE AND STATIC COMPUTER INSTALLATIONS

4.1.1 Physical Size and Weight Restrictions
4.1.2 Electrical Noise Problems
4.1.3 Mechanical Problems
4.1.4 Power Supply Requirements

4.2 PROCESSING REQUIREMENTS - INPUTS AND OUTPUTS

4.2.1 Motor Control
4.2.2 Brake Control
4.2.3 Speed and Distance Determination
4.2.4 Speed Supervision

4.3 PROCESSING REQUIREMENTS - PROGRAM SOFTWARE

4.3.1 Automatic Driver Computer
4.3.2 Tachometer Computer
4.3.3 Safety Supervisor Computer
# RELIABILITY CONSIDERATIONS

4.4.1 Self-Testing  103
4.4.2 Self-Checking  103
4.4.3 Fault-Tolerant  104

# CONCLUSION

4.5 REFERENCES FOR CHAPTER 4  107

## CHAPTER 5 : SUMMARY OF THE BRATO PROJECT

5.1 INTRODUCTION  108
5.2 THE BRATO SUBSYSTEMS  109
5.2.1 The Autodriver  109
5.2.2 The Supervisor  112
5.2.3 The Tachometer Processor  113
5.2.4 The Receiver  114
5.3 THE SYSTEM MONITORS  114
5.4 CONCLUSION  116

## REFERENCES FOR CHAPTER 5  117

## CHAPTER 6 : THE DEVELOPMENT OF CYCLONE, A COMPACT, SELF-CHECKING MULTIPROCESSOR

6.1 INTRODUCTION  118
6.2 DESIGN OBJECTIVES  118
6.3 THE CYCLONE CONCEPT  120
6.3.1 Time Redundant Multiprocessing  120
6.3.2 The Choice of Bus Structure  122
6.3.3 Optimizing for Throughput  127
6.3.4 Task Switching at Sector Boundaries  130
6.4 HARDWARE DEVELOPMENT  132
6.4.1 The Choice of Microprocessor  132
6.4.2 Synchronizing Problems and Solutions  133
6.4.3 Loss-of-Processor Tolerance  135
6.5 SOFTWARE DEVELOPMENT TOOLS  136
6.5.1 The "Soft" Front Panel  136
6.5.2 The Bootstrap Loader  138
6.5.3 The Hex Editor  138
6.5.4 Magnetic Tape Cartridge Handler  139
6.6 CHECKING SOFTWARE DEVELOPMENT
   6.6.1 Checking Philosophy 139
   6.6.2 Version 1 Software 142
   6.6.3 Version 2 Software 142
   6.6.4 Version 3 Software 142
   6.6.5 Reset Recovery and Checkout 143

6.7 DUPLEX CYCLONE
   6.7.1 Single Hard Fault Tolerance 143
   6.7.2 Theoretical Reliability Improvement 144
   6.7.3 Switching Monitor Complexity 146

6.8 CONCLUSION 146

REFERENCES FOR CHAPTER 6 148

CHAPTER 7: HARDWARE DESCRIPTION OF THE PROTOTYPE
   MULTIPROCESSOR MODULE - CYCLONE I 149 - 176

7.1 INTRODUCTION 149

7.2 FUNCTIONAL DESCRIPTION 151
   7.2.1 Timing and Synchronization 151
   7.2.2 Bus Control and Priority Assignment 158
   7.2.3 The Processor 163
   7.2.4 Address Decode and Memory 165
   7.2.5 Input/Output 169
   7.2.6 Control Panel Interface 174

7.3 CONCLUSION 176

CHAPTER 8: CYCLONE I SOFTWARE DEVELOPMENT 177 - 239

8.1 INTRODUCTION 177

8.2 SYSTEM STRUCTURE - PROGRAMMER'S VIEWPOINT 177
   8.2.1 System Memory Map 177
   8.2.2 The INS8900 Instruction Set and Addressing Modes 179

8.3 PROGRAM DESIGN RULES 184
   8.3.1 Input/Output Operation 184
   8.3.2 Scratchpad RAM Usage 185

8.4 CYCLIC OPERATION 188
   8.4.1 Software Overview 188
   8.4.2 The Level 5 Interrupt Handler 188
   8.4.3 The Level 4 Interrupt Handler 190
   8.4.4 The Reset Handler Routine 195
8.5 TEST PROGRAMS
8.5.1 Simple Cycle Check Program
8.5.2 The Multi-Task Tachometer Program
8.5.3 The Level 2 Timer Interrupt Handler
8.6 CONCLUSION
FLOW CHARTS AND PROGRAM LISTINGS
REFERENCES FOR CHAPTER 8

CHAPTER 9: AN EXAMINATION OF MODULE PERFORMANCE

9.1 INTRODUCTION
9.2 OPERATION WITH FAULT-FREE CONDITIONS
9.3 PERFORMANCE WITH FAULTS
9.3.1 CPU Failure
9.3.2 Transient Faults - The Interference Box
9.3.3 Cyclone Response to Power Supply Transients
9.4 CONCLUSION
REFERENCES FOR CHAPTER 9

CHAPTER 10: CONCLUSIONS, COMMENTS AND SUGGESTIONS
FOR FURTHER WORK

10.1 CONCLUSIONS AND COMMENTS
10.2 SUGGESTIONS FOR FURTHER WORK
REFERENCES FOR CHAPTER 10

THE PLATES
APPENDIX I: UTILITÍ SOFTWARE

APPENDIX II: INS800 INSTRUCTION SET
**LIST OF ILLUSTRATIONS**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>&quot;Bath-Tub&quot; Failure Rate Curve</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Reliability Curve for Single Module</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>Duplex System</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Triple Modular Redundancy</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>Triplicated TMR</td>
<td>14</td>
</tr>
<tr>
<td>2.6</td>
<td>System-Voter Configurations</td>
<td>17</td>
</tr>
<tr>
<td>2.7</td>
<td>Plots of Simplex and TMR Reliability</td>
<td>19</td>
</tr>
<tr>
<td>2.8</td>
<td>Plots of Voter-Switch Reliability</td>
<td>19</td>
</tr>
<tr>
<td>2.9</td>
<td>Majority and Threshold Voters</td>
<td>27</td>
</tr>
<tr>
<td>2.10</td>
<td>Spare Assignment Strategies</td>
<td>29</td>
</tr>
<tr>
<td>3.1</td>
<td>Example of Quadding</td>
<td>41</td>
</tr>
<tr>
<td>3.2</td>
<td>Duplex Redundancy</td>
<td>41</td>
</tr>
<tr>
<td>3.3</td>
<td>Basic C-APT Duplex System</td>
<td>44</td>
</tr>
<tr>
<td>3.4</td>
<td>C.vmp Configuration</td>
<td>44</td>
</tr>
<tr>
<td>3.5</td>
<td>Cm* Structure</td>
<td>47</td>
</tr>
<tr>
<td>3.6</td>
<td>FTMP Structure</td>
<td>47</td>
</tr>
<tr>
<td>3.7</td>
<td>Micronet Structure</td>
<td>51</td>
</tr>
<tr>
<td>3.8</td>
<td>JPL Star Structure</td>
<td>51</td>
</tr>
<tr>
<td>3.9</td>
<td>SIFT Structure</td>
<td>53</td>
</tr>
<tr>
<td>3.10</td>
<td>FTMP Redundant Clock</td>
<td>53</td>
</tr>
<tr>
<td>3.11</td>
<td>BFS Structure</td>
<td>60</td>
</tr>
<tr>
<td>3.12</td>
<td>Basic VSD Logic</td>
<td>60</td>
</tr>
<tr>
<td>3.13</td>
<td>Rollback and Retry</td>
<td>71</td>
</tr>
<tr>
<td>4.1</td>
<td>Train Computer Hazards</td>
<td>85</td>
</tr>
<tr>
<td>4.2</td>
<td>EMU Motor Circuit Schematic</td>
<td>90</td>
</tr>
<tr>
<td>4.3</td>
<td>Notching Relay Operation</td>
<td>92</td>
</tr>
<tr>
<td>4.4</td>
<td>Notching Truth Table</td>
<td>92</td>
</tr>
<tr>
<td>4.5</td>
<td>Simplified EP Brake Schematic</td>
<td>94</td>
</tr>
<tr>
<td>4.6</td>
<td>Brake Logic Truth Table</td>
<td>94</td>
</tr>
<tr>
<td>4.7</td>
<td>Basic Train Control System</td>
<td>102</td>
</tr>
<tr>
<td>4.8</td>
<td>System Degradation Chart</td>
<td>105</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.1</td>
<td>BRATO System Schematic</td>
<td>110</td>
</tr>
<tr>
<td>5.2</td>
<td>System Monitor for TMR Circuit</td>
<td>115</td>
</tr>
<tr>
<td>5.3</td>
<td>Output Drive Circuit</td>
<td>115</td>
</tr>
<tr>
<td>6.1</td>
<td>Time Redundancy for N Processors</td>
<td>121</td>
</tr>
<tr>
<td>6.2</td>
<td>Global Bus with Central Switch</td>
<td>124</td>
</tr>
<tr>
<td>6.3</td>
<td>Independent Bus Request System</td>
<td>124</td>
</tr>
<tr>
<td>6.4</td>
<td>Cyclone Bus Control Logic</td>
<td>126</td>
</tr>
<tr>
<td>6.5</td>
<td>Best-Case Effect of BUF on Throughput</td>
<td>129</td>
</tr>
<tr>
<td>6.6</td>
<td>Effect of Interference on Throughput</td>
<td>129</td>
</tr>
<tr>
<td>6.7</td>
<td>Early Cyclone Timing Cycle</td>
<td>141</td>
</tr>
<tr>
<td>6.8</td>
<td>Final Cyclone Timing Cycle</td>
<td>141</td>
</tr>
<tr>
<td>6.9</td>
<td>Duplex Cyclone Operation</td>
<td>145</td>
</tr>
<tr>
<td>6.10</td>
<td>Reliability of Duplex Cyclone</td>
<td>145</td>
</tr>
<tr>
<td>7.1</td>
<td>Cyclone I Prototype Block Diagram</td>
<td>150</td>
</tr>
<tr>
<td>7.2</td>
<td>Timing and Sync Block Diagram</td>
<td>152</td>
</tr>
<tr>
<td>7.3</td>
<td>Clock and Timing Logic</td>
<td>154</td>
</tr>
<tr>
<td>7.4</td>
<td>Sync and Tacho Logic</td>
<td>155</td>
</tr>
<tr>
<td>7.5</td>
<td>Synchronisation Timing</td>
<td>157</td>
</tr>
<tr>
<td>7.6</td>
<td>Successful Watchdog Restart</td>
<td>159</td>
</tr>
<tr>
<td>7.7</td>
<td>Bus Control Block Diagram</td>
<td>159</td>
</tr>
<tr>
<td>7.8</td>
<td>Bus Control Logic</td>
<td>161</td>
</tr>
<tr>
<td>7.9</td>
<td>Bus Control Timing</td>
<td>162</td>
</tr>
<tr>
<td>7.10</td>
<td>MPU Logic</td>
<td>164</td>
</tr>
<tr>
<td>7.11</td>
<td>Address Decode and Load Latch Logic</td>
<td>166</td>
</tr>
<tr>
<td>7.12</td>
<td>Memory Logic</td>
<td>168</td>
</tr>
<tr>
<td>7.13</td>
<td>Input/Output Logic</td>
<td>170</td>
</tr>
<tr>
<td>7.14</td>
<td>Tape and Common Memory Logic</td>
<td>172</td>
</tr>
<tr>
<td>7.15</td>
<td>LED Latches and Switch Buffers</td>
<td>173</td>
</tr>
<tr>
<td>7.16</td>
<td>Control Panel Interface Logic</td>
<td>175</td>
</tr>
<tr>
<td>8.1</td>
<td>Cyclone I System Memory Map</td>
<td>178</td>
</tr>
<tr>
<td>8.2</td>
<td>Peripheral Word Formats</td>
<td>180</td>
</tr>
<tr>
<td>8.3</td>
<td>Peripheral Word Formats</td>
<td>181</td>
</tr>
<tr>
<td>8.4</td>
<td>INS8900 Register Structure</td>
<td>183</td>
</tr>
<tr>
<td>8.5</td>
<td>Flagword Formats</td>
<td>186</td>
</tr>
<tr>
<td>8.6</td>
<td>MPU Duty Chart</td>
<td>186</td>
</tr>
<tr>
<td>8.7</td>
<td>Scratchpad Memory Map</td>
<td>191</td>
</tr>
</tbody>
</table>
9.1 Plots of Sector Length and Output 241
9.2 Power Interference Box Circuit Diagram 246

| PLATE 1 | The Cyclone I Prototype System 257 |
| PLATE 2 | The Cyclone I Prototype Module 257 |
| PLATE 3 | The Magnetic Tape Unit 258 |
| PLATE 4 | The Power Interference Box 258 |
LIST OF SYMBOLS AND ABBREVIATIONS

Most of the abbreviations are explained when first used, but are listed here for reference. Memory addresses are expressed in hexadecimal form, unless otherwise stated.

R(t) Reliability as a function of time

\( \lambda \) Failure Rate

MTBF Mean Time Between Failures

LSI Large Scale Integration

TMR Triple Modular Redundancy

R_s System Reliability

R_m Module Reliability

R_v Voter Reliability

NMR N-Modular Redundancy

I/O Input/Output

LRU Line Replaceable Unit

VSD Voter, Switch, Disagreement Detector

EMU Electric Multiple Unit

BRATO British Rail Automatic Train Operation

EP Electro-Pneumatic

BCP Brake Cylinder Pressure

BCH Bose-Chaudhuri-Hocquenghem Code

EOS End-of-Section

BUF Bus Utilisation Factor

T_s System Throughput (No Interference)

T_i System Throughput (With Interference)

X No. of Bus Access Cycles/Instruction

Y Total No. of Cycles/Instruction

U Bus Utilisation Factor

UART Universal Asynchronous Receiver Transmitter

MTT Multi-Task Tachometer
"When a fail-safe system fails, it fails by failing to fail safe"

"Complex Systems usually work in the failure mode"

*John Gall, Systemantics, 1977*

"Computers are unreliable, but humans are even more unreliable"

*from Murphy's Law*
CHAPTER 1

INTRODUCTION

1.1 The Background

The origins of automatic train control go back to the days when the Great Western Railway introduced the AWS driver warning system. Crude by current standards, it nevertheless provided for automatic brake application should the driver ignore a red signal. This represented a major step forward in the field of safety through automation on railways, but little real progress has been made since. Railway engineers have been striving to introduce the new technology of integrated circuits to locomotive performance control and signalling. However, the fail-safe requirement laid down by the operators has kept many system designs off the rails. The situation is made worse by the train-borne environment which, while being acceptable to humans, contains many potentially fatal electrical hazards to the unprotected computer. British Rail projects of varying complexity such as C-APT, TACT and BRATO have either come and gone without real-train operation or are bogged down for the conceivable future. When this project was started in October 1978, the BRATO project for automatic train driving was well under way, with tests being carried out on the ability of a 6800 microprocessor system to brake a train to an accurate stop.

The project specification called for an examination of alternative designs for highly-reliable and fault-tolerant computers with the aim of either proving the BRATO arrangement impossible to better, or developing a new machine in parallel with the BR design.
1.2 Summary of Work

Considerable research was carried out on the subject of fault-tolerant computing in general, taking in static as well as mobile systems. Emphasis has always been on the engineering side, with theoretical work limited to that applicable to recognisable and realistic hardware.

The project was sponsored by British Rail, and time was spent at the Railway Technical Centre, Derby, becoming acquainted with current ideas on automatic locomotive control, and with the BRATO project itself. The Author assisted with the testing of a simple uniprocessor 6800 system running a braking curve program. These trials generated much data enabling software to be fine-tuned prior to running on the more advanced BRATO hardware. This experience of the real problems encountered when attaching delicate microelectronics to large and heavy mechanical systems, early on in the project, modified many later ideas.

A total of several months was spent designing and building a train simulator for the purposes of laboratory testing both BRATO and CYCLONE. This simulator is physically divided into two units, a 19" rack containing three 6800 systems, and a desk console. The train pulse processor produces bi-phase tachometer signals in response to either simulated manual controls or an autodriver. The program also displays speed and distance on the console. The tachometer section is basically a uniprocessor version of the BRATO design and produces serial data for the third system, a receiver simulator unit. This unit produces telegram data at the required pre-programmed intervals and also drives a train position indicator. Manual signal switches allow the response of an autodriver algorithm to red signal aspects to be determined. A study of Chapters 4 and 5 will enable
the reader to understand more clearly the terms tachometer and receiver used here. Final debugging of the unit and programming with signal data was never carried out as the BRATO programme was abandoned.

A laboratory prototype fault-tolerant multi-processor has been built at Loughborough University, and given the name CYCLONE I. Extensive development of the basic module has been carried out, together with a great deal of software work. Laboratory peripherals such as a VDU and magnetic tape cartridge unit were added later to facilitate software development, and utility programs including a hexadecimal editor and mag. tape file handler written. System response to noise was examined using a power supply interference box especially built for the job.

1.3 Thesis Structure

Chapter 2 surveys the various techniques evolved over the years, to make computers more reliable. The alternative methods of fault-avoidance and fault tolerance are presented and discussed, together with the theoretical background to specific system configurations.

Chapter 3 considers the practical designs built since the 1960's and attempts to illustrate how thinking has changed with the advent of large scale integrated circuits. Many of the designs were for mobile applications, albeit exotic ones such as spacecraft control, and many of the problems are common to the train control field.

Chapter 4 discusses train control requirements and problems in some depth. The reader's attention is drawn to the safety aspects as well as to the benefits of automatic control including better time-keeping and fuel economy. The electrically severe environment present on board the average electric locomotive is described and techniques to improve computer survivability outlined.
Chapter 5 briefly describes the British Rail Automatic Train Operation project and outlines the work carried out by BR design engineers in meeting the tough safety and operational specifications laid down by the operators. The methods used may be compared with those suggested in the subsequent chapters on Cyclone development.

Chapter 6 covers the development of, and thinking behind, a multi-microprocessor system designed by the Author as an alternative to the BRATO scheme. The theory behind the bus arbitration and internal communication protocol is discussed together with the reasons for the particular choice of microprocessor.

Chapter 7 is an in-depth hardware description of the Cyclone I module. Full circuit diagrams are presented together with timing diagrams where relevant.

Chapter 8 covers all the system software developed to date and includes assembler listings and flow charts. The various improvements in speed and checking functions are described. Chapters 7 and 8 when taken together cover the basic design philosophy behind Cyclone, using as it does time redundancy to cut down on hardware redundancy.

Chapter 9 describes the module performance both in conditions of no electrical interference, and with power supply 'glitches' present. The design and construction of an interference generator is described.
2.1 Introduction

As a concept, computer reliability can have as many loose definitions as there are computer operators and users. It also depends on the level of viewpoint; a computer installation manager may see his system as reliable because, generally, throughput matches requirements. The operator may have a less than charitable view if in order to achieve this throughput, he has to correct mistakes and call for the maintenance engineer to replace components frequently. The maintenance man may regard the system as totally unreliable because he carries out preventive work as well as locating potential faults and eliminating others which as yet have not caused errors 'downstream'. This traditional human approach to reliability has been used since the first system became operational and still figures strongly today, although new techniques of hardware and software design are making computers inherently more reliable. In order to design these new systems, researchers have attempted to qualify the terms reliability, availability, etc., and then to quantify them, first in general mathematical expressions and then in more specific numerical form to allow direct comparison of different system ideas. The first section of this chapter will give some of the generally accepted verbal definitions. A brief discussion on the relative merits of two major design approaches follows, and finally there is an examination of the theoretical bases behind redundant circuit design.
2.2 Reliability Definitions

From the extensive list of definitions due to Naresky\(^{(1)}\), reliability is defined as a statistical probability of success:-

- Reliability is the probability that a given system will perform a required function under stated conditions for a stated period of time.

Hence reliability is a function of time and varies between 1 (totally reliable), and 0 (totally unreliable). The object of the designer is to maintain this figure for the system, as close as possible to unity for as long as possible, or at least until the mission time or life time requirement is achieved. For the occasion when something does go wrong with the system, basic definitions of fault, failure and error, are needed:-

- A fault is the direct cause of the system not performing its designated task correctly. A fault can exist in hardware, timing or in the software (software mistakes are often referred to as program 'bugs'). It can also be permanent (hard) or transient in nature.

- A failure is the deviation from the system specification due to the existence of a fault. This in effect means that the computer gets itself into an incorrect state after sequencing through a number of correct states.

- An error is the manifestation of the machine failure in the form of a mistake in a particular piece of program calculation.

When designing the system, likely causes of faults must be considered, and these are termed "threats"\(^{(2)}\).
A threat is a stress producing a fault and can be normal environmental, abnormal environmental, or design error.

In order to arrive at a mathematical function for reliability two main assumptions are made. These are that device failures are random in occurrence and are thus statistically independent, and the failure rate expressed as so many failures per hour, is a constant over the equipment lifetime. Both these assumptions are shaky but providing certain conditions are met, they have been found to be reasonably valid for system analysis purposes. Statistical independence assumes that the failure of one component does not impose increased stress on its neighbours thus increasing their likely failure rate. When computers were constructed from discrete transistors, resistors, etc., then failure short circuit of a capacitor, say, could cause overload of a transistor and lead to cascaded failures. Integrated circuit logic elements such as TTL appear to be less susceptible to cascading damage and should conform more closely to the reliability model. The constant failure rate requirement can be met if the initial "burn-in" and "wear-out" phases of system life are left out (Fig. 2.1) and only the flat part of the "bath-tub" curve is used. The burn-in phase should eliminate all the inherently faulty components, hardware design errors and program 'bugs'. Hence the theoretical reliability of a non-redundant system module with respect to time is given by:

\[ R(t) = e^{-\lambda t} \]

where \( \lambda \) is a constant failure rate.

This yields the exponential curve shown in Fig. 2.2. The module is in theory totally reliable, \( R(t) = 1 \) when time = 0 which is when the system enters the constant failure rate area of its lifetime (Fig. 2.1).
FIG. 2.1 COMPONENT FAILURE RATE WITH TIME

FIG. 2.2 RELIABILITY OF A SIMPLEX UNIT
This is all very well in theory, but some method of determining when the system failure rate has flattened out must be found. In practice this is achieved at high cost by 'soak' testing the system until the burn-in phase is left. Generally manufacturers of military and some life-critical equipment are the only people who do this. With commercial and domestic systems, new products are likely to be tried out on the customer and reliability calculation adjustments based on complaints received. The Mean Time Between Failures (MTBF) is often used as a reliability parameter and is simply the reciprocal of the failure rate $\lambda$. In order to arrive at a figure for $\lambda$ or MTBF for a module, the individual component failure rates are added together using the assumption of statistical failure independence. From the graph of Fig. 2.2 it can be seen that for the particular case $t = \text{MTBF}$, then $R(t) = 0.37$. In other words as the system approaches an age corresponding to its MTBF, there is only a 37% probability that continued failure-free operation will occur. The discussion will now centre on approaches to improving basic system reliability.

2.3 Approaches to Reliable Design

2.3.1 Fault avoidance

Fault avoidance or fault intolerance aims to increase the MTBF of the system by improving the individual module and component MTBFs. This is done to the point where the required lifetime or mission time of the equipment is considerably less than the system MTBF, yielding probabilities of successful operation of say 90-99%. Generally the mission time is used as the basis for calculation particularly in the
military aircraft field, where the MTBF only has to be hours. Fault avoidance is achieved in several ways:

- Strict component quality control
- Heavy component de-rating; only using, say, 10% of power capabilities
- Reduced environmental stress using cooling systems, etc.
- Absolute minimum of components used to do the job, i.e. design efficiency with no redundancy, (which also impedes testing) \( ^{(3)} \).

These techniques have been used extensively in the field of mobile computing, particularly in military applications because of the need to keep weight down. New techniques using LSI chips have allowed the alternative approach - Fault tolerance - to establish itself as a better direction to take when designing reliable systems.

2.3.2 Fault tolerance

Fault tolerance assumes that faults are likely to occur no matter how many steps are taken to avoid them. The definition of a fault-tolerance computing system is given by Avizienis\(^{(4)}\):

- A fault-tolerant computing system has the built-in capability (without external assistance) to preserve the continued correct execution of its programs and input/output functions in the presence of a certain set of operational faults.

This deceptively simple definition is in fact very difficult to translate into a real system. Assuming that we have a permanent fault in the system then three requirements must be met to satisfy the desire for 'continued correct execution'. These are:
• Error Detection. The system must be able to detect its own mistakes.

• Fault Diagnosis. Having detected an error while running the application program, the system must be able to isolate the fault to a group of components or modules which can be by-passed or replaced under computer control.

• Fault Recovery. Once the fault has been located, the system must take action to eliminate or minimise its effect.

As pointed out by Bennetts\(^5\), fault recovery does not imply fault repair and the analogy of using the spare tyre on a car, repairing the damaged item later is quoted. Ideally, the above three processes should take place as quickly as possible, so that interruption to data throughput is kept minor. Protective redundancy is introduced in the form of extra hardware or software or both, in an attempt to achieve the design goal of almost instant recovery after failure. In practical terms it is almost impossible to cater for every single type of component failure likely to occur. Some faults are bound to cause catastrophic system loss, and all that can be done is to reduce their probability of occurrence to an acceptably low figure.

Some types of faults "lurk" in little-used parts of system operation and may escape the "cover" of error protection.

• Coverage is the conditional probability that a system will recover, given that a fault has occurred.

The concept of coverage allows the effectiveness of particular reliability schemes to be assessed in terms of their ability to detect and deal with all the possible failure modes. Bouricius and later, Arnold\(^6\) have extensively analysed coverage and found that it must be almost total if
predicted reliability is to be achieved. Arnold concludes that certain classes of undetected single fault or detected but non-recoverable fault can drastically reduce system reliability, even if each is individually innocuous.

2.4 Basic Fault-Tolerant Computer Design Paths

Fault tolerance and fault avoidance are not mutually exclusive and both techniques can be combined when working up a particular design. The introduction of redundant components and the inclusion of spare modules does not automatically improve system reliability. Indeed, as the analysis of TMR will show, replication of basically low reliability components will make the redundant system less likely to complete its mission that the simplex one. The use of top-quality components and de-rated design are necessary for the maximum benefit of fault-tolerant computing to be achieved. The extra redundancy can take many forms from simple to highly complex as shall now be shown. The basic "building block" concepts covered here illustrate both static and dynamic redundancy techniques:-

- Static redundancy or fault masking whereby a module features internal, wired-in redundancy to mask faulty components. One such method is termed "quadding", discussed in Chapter 3. Alternatively modules are duplicated or triplicated and an output derived from a majority voter.

- Dynamic redundancy where standby spare units are switched in and out as required by fault detection logic and/or software.
2.4.1 Duplex operation

Duplication of a module with a single voter is shown in Fig. 2.3. In its basic form this arrangement has no capacity to locate faults and for safety reasons must shutdown on disagreement. If module reliability and unreliability are represented by \( R \) and \( Q \) respectively when \( Q = 1 - R \), then with a perfect voter, system operation probabilities are given by:

\[
(R+Q)^2 = R^2 + 2RQ + Q^2 \quad 0 \leq R \leq 1
\]

The system is allowed to function only if both units work and so system reliability is \( R^2 \), indicating that duplication reduces overall probability of mission success, if the modules are imperfect, but with a perfect voter. In practice, this system is being used by BR, as it does meet mission time requirements, it is safer than simplex, and their particular arrangement can recover from transient disagreement.

2.4.2 Basic Triple Modular Redundancy (TMR)

A simple TMR system (Fig. 2.4) extends the duplex arrangement by adding a third identical module and by changing the comparator into a majority voter. Again, assuming a perfect voter, module reliability is \( R \), unreliability \( Q \) then the system probabilities are derived from the binomial expansion of \((R+Q)^3\).

\[
(R+Q)^3 = R^3 + 3R^2Q + 3RQ^2 + Q^3
\]

Successful operation of TMR requires either all three modules to work, or two out of three. Hence the system reliability \( R_s^3 \) contains the first two terms of the expansion only, \((7)\):-
FIG. 2.3 DUPLEX SYSTEM

FIG. 2.4 TRIPLE MODULAR REDUNDANCY

FIG. 2.5 TRIPPLICATED TMR
\[ R_s = R_m^2 + 3R_m^2Q = R_m^2 + 3R_m^2(1-R_m) \]
\[ R_m = \text{module reliability} \]
\[ = 3R_m^2 - 2R_m^3 \]...

The main point of interest here is that if the module reliability \( R_m \) falls below 0.5, then the system is less reliable than a simplex one. Two important conclusions can be drawn:

- TMR only improves overall reliability if reliable components are used in the first place. It will not compensate for poor quality.
- TMR by itself will improve short-term mission reliability only.

### 2.4.3 Triplicated TMR

The basic TMR system depends heavily on one non-redundant voter. If we now bring in voter unreliability, then the voter itself needs to be triplicated (Fig. 2.5). The initial proposals of Von Neumann were expanded by Lyons and Vanderkulk (8) with much analysis of the effect of imperfect voters on reliability. Lyons gave examples of various possible configurations and found that each had its own merits depending on the specific mission time requirements. Wakerly (9) has applied the general terms used so far to the practical aspects of microcomputer design. He divides the computer into two parts, processor and memory, with TMR applied in three main ways:

- Three identical CPU and memory systems running in synchronism, the only voters being on peripheral device inputs to check that all systems agree with the output requests (Fig. 2.6a)
- Three identical CPU and memory systems with voting between CPU and memory inputs (Fig. 2.6b)
Three identical CPU and memory systems with voting between memory output and CPU (Fig. 2.6c)

The first option proved more reliable than a simplex system but if a transient disturbance knocks out processor synchronism, there is no method to restore order. Analysis shows that the second approach, CPU-voter-memory is less reliable than the first taking into account voter reliability. However, it does have the capacity to detect and mask CPU errors from corrupting the memory, and can order a reset to effect resynchronisation in the event of transient failure. The third approach is the most reliable of all in hard failure terms, because by suitably arranging the voters around a semiconductor RAM chip array, multiple RAM chip failures can be tolerated, of a type which could cause system failure in the other two configurations. However, the CPU-memory-voter arrangement will need extra circuitry to deal with transients. The reset input on the microprocessor chip is used by Wakerly (9) as the main method of re-establishing synchronism. Reset and Interrupt inputs are seen as the only guaranteed method of restoring order in a transient-affected microprocessor system. The application of pre-defined synchronising sequences to system inputs as described by Wakerly (10) is really only suitable for use in less complex sequential logic. To compare basic and triplicated TMR, the voter reliability $R_v$ is added to Equation (1). For basic TMR the single voter must work at all times and so is a multiplying factor on $R_v$:

$$\text{Reliability of basic TMR with voter reliability } R_v \quad = R_v (3R_m^2 - 2R_m^3)$$
FIG. 2.6  SYSTEM - VOTER CONFIGURATIONS
For the triplicated TMR if each module has an attached voter which must work then a module-voter reliability is $R_m R_v$. Substituting in Eqn.(1):

$$\text{Reliability of triplicated TMR with voter reliability } R_v$$

$$= 3R_m^2 R_v^2 - 2R_m^3 R_v^3$$

$$= \frac{2}{m} R_m^2 (3 - 2R_m R_v)$$ \text{(Voter tied to module)}

However, there is no reason why the failure of a voter should knock out its attached module. Any two out of three modules may work and any two of three voters may work, so the pessimistic equation above becomes:

$$\text{Reliability of triplicated TMR with voter reliability } R_v$$

$$= (3R_m^2 - 2R_m^3)(3R_v^2 - 2R_v^3)$$ \text{(Untied voters)}

The reliability functions for the various simplex and TMR arrangements so far discussed are plotted in Fig. 2.7. The ratio of voter failure rate to module failure rate $k$ is taken as 0.1 in this example which assumes very reliable voters. Empirical evidence indicates that realistic figures lie in the range 0.1 to 0.5 depending on voter complexity. For the various forms of TMR, the voter can be very simple and thus the lower figure can be taken, but the complex voter-switches necessary for standby work may require use of the higher number. The graph plots of Fig. 2.7 indicate an increasing improvement in short term reliability, using TMR, but all the redundant methods are inferior to standard simplex once system reliability has fallen to about 0.5. An examination of the long term requirements of a particular system may indicate that static redundancy TMR methods
FIG. 2.7 PLOTS OF SIMPLEX AND TMR RELIABILITY

FIG. 2.8 PLOTS OF VOTER-SWITCH RELIABILITY
do not give a good enough probability of success. In an attempt to 'flatten-out' the reliability curve, some form of standby or dynamic redundancy is needed.

2.4.4 Adaptive voting

In a typical arrangement, five identical modules drive a single voter which works on a majority 3-out-of-5 basis. As modules fail, the system degrades to a TMR 2-out-of-3, and only ceases to function when the final two modules disagree. This yields a large improvement in system reliability over the static methods (Fig. 2.8). The plot shows that the adaptive voter still has a reliability figure of 0.8 at the corresponding time when the simplex circuit has dropped to a 0.5 probability. The reliability equation for the adaptive 3-out-of-5 voter system, with a perfect simplex voter circuit is:—

\[ R_s = 1 - 5(1 - R_m)^4 + 4(1 - R_m)^5 \]

or generally for \( m \) redundant modules:

\[ R_s = 1 - m(1 - R_m)^{m-1} + (m - 1)(1 - R_m)^m \]

2.4.5 TMR/single switching

Three modules are used in this configuration, operating in standard TMR until one fails. Not only is the failed module discarded however, but also one of the good ones and operation reverts to simplex. The approach is justified because the resultant failure rate using one module is better than for two. Fig. 2.8 shows a significant improvement in reliability over simplex with a reliability of 0.69 to the latters 0.5 at the same time. This theoretical improvement over simplex and basic
TMR unfortunately suffers from the drawback of an unchecked module running by itself. Extra hardware needed to deal with this serious safety risk may wipe out the advantages.

The reliability equation for the TMR/single system is:

\[ R_s = \frac{3}{2^m} \cdot (1 - R^3) \]

The next arrangement is the logical extension of TMR/single operation.

**2.4.6 TMR/single/single switching**

Instead of discarding the good module in the previous configuration it is used now as a standby, switched in when the second module fails. The result is a considerable flattening out of the reliability curve brought about because of the minimum amount of hardware used. Now there is a figure of \( R_s = 0.875 \) to the simplex 0.5, but once again there is the risk of unchecked processor operation.

The reliability for TMR/single/single operation is:

\[ R_s = 1 - (1 - R^m)^3 \]

The safety aspect of train control computer design probably eliminates this method as well. For safety reasons at least the first failure, and ideally further failures, should not destroy the redundant checking.

**2.4.7 TMR/hybrid**

The basic TMR system is now provided with standby spare modules to switch in as active units fail. When all spares are used, the system runs until two remaining units disagree.
The reliability equation for TMR/hybrid systems is:

\[ R_s = 1 - (1 - R_m)^s + 2(1 + R_m(s + 2)) \]

where \( s \) is the number of spare modules provided.

The plot in Fig. 2.8 is for two spares and while less reliable than the previous method, up to three failures are allowed before the system shuts down due to a lack of checking on the last module. Mathur and Avizienis (11) have subjected this classic reliability improvement to much analysis, expanding the TMR core to a more general NMR, but concluding in the end that the best approach is to keep \( N = 3 \), i.e. TMR, and add as many spares as required to suit the mission time. The law of diminishing returns does apply here, and reliability improvement is minimal beyond six or seven spares. In addition, the necessary switch complexity, whose reliability has not been taken into account thus far, will set a ceiling on further improvement. However, this system does satisfy safety requirements for first and subsequent failures, and is suitable for consideration in the train control application.

2.4.8 TMR/hybrid/single

Bearing in mind that the basic TMR/hybrid system tolerates one or more module failures without loss of checking, there is a strong argument for allowing single processor operation as a last phase. If it is assumed that several consecutive failures reducing operation to a basic TMR level have been announced to human operators, then the increased vigilance of the latter would allow a further degradation to simplex running, possibly manually enabled. In these circumstances we can improve theoretical reliability still further (Fig. 2.8). At the normal
TMR crossover point of 0.5, this redundancy system still maintains a probability of 0.89 in this case using two spares.

The reliability equation for the TMR/hybrid/single voting system with two standby spares is:

\[ R_s = -\frac{3.5}{2^m} + 5 R_m^4 - 5 R_m^3 + \frac{5 R}{2^m} \]

2.4.9 TMR/hybrid/single/single

Finally, the life of the last system arrangement can be extended by switching in the working module taken out when one out of three remaining units failed as for TMR/single/single. As before, this technique does require special-purpose software and hardware to detect failure of a non-checked single processor. Reliability is once again improved in theory (Fig. 2.8) with a figure of 0.97 for the normal TMR crossover point of 0.5, with two spares.

The reliability equation for the TMR/hybrid/single/single system with \( s \) standby spare modules is:

\[ R_s = 1 - (1 - R_m)^{s+3} \]

For safety-critical applications, it is felt that the apparent increase in reliability achieved is not worth the risk of prolonged single processor operation. Dropping down from full TMR to a single processor under strict manual supervision may be tolerable, but replacement of that processor by a further single unit may undermine the whole purpose behind checked computer operation. All the system types discussed so far have powered standby units with the same reliability as the active units.
2.4.10 Hot and cold standby

A further step to improve reliability involves keeping the standby modules unpowered until they are switched in. It is reasonable to assume that the inert unit should suffer at worst a marginal loss of reliability with time, at best, none at all. Experimental work evaluated by Nerber\(^{12}\) indicates the difficulty in assessing the failure rate of unpowered equipment. Experiments suggest that it is 10-30% that of powered equipment. Dennis\(^{13}\) in his comprehensive summary of voter-switch types has examined the effect on the overall reliability of all the standby systems used so far. He concludes that for short missions \((R_m = 1 \text{ to } 0.6)\), a variation in the standby failure rate from zero to that of a powered unit has no dramatic effect on the system reliability.

For long term missions however \((R_m < 0.15)\), such variation does have a marked effect. Hence the conclusion is drawn that for short-term missions, the standby modules might as well be kept powered up for three main reasons:

- No extra switch complexity is required to control the module power lines.
- Current surges in system power lines at switch on are avoided.
- If the modules are local microcomputers, then the standby units can be running system programs in parallel with the active units. This then reduces the time necessary to effect an orderly takeover of program function on unit failure. Cold standby units require internal registers, memory locations, etc to be loaded with current data before being switched in.

Long term system survival is not considered to be of major importance in railway operation, because of the availability of manual maintenance. For this reason, the use of cold standby units is probably unjustified.
2.4.11 Coverage and imperfect voter-switches

Dennis (13) and Arnold (6) show that even a small amount of 'uncoverage' will alter very significantly these reliability calculations. All the voter-switch systems discussed are very susceptible, but if coverage is the same for each then the more sophisticated types retain their overall improvement in reliability over the simpler types. The concept of coverage includes not only the ability of the system to detect failures, but also its recovery capabilities, in this case the efficiency of the voter switch in bringing in alternative units. Consequently, the reliability of the voter switch may be adversely affected by circuit complexity.

The different voter-switch configurations discussed are not themselves redundantly arranged so their reliability $R_v$ is just a multiplying factor on each of the reliability equations. This approach is justified if voter switch reliability is high relative to each module. This should be the case as recent developments in LSI circuits suggest that a module is likely to be a complete microcomputer or a large memory array, for example. The voter switch may well consist of a few flip-flops, gates and a read-only memory. The latter ROM can be used to replace the hundreds of gates necessary to handle all the various combinations of active and standby modules needed to maximise the coverage. One such ROM performing 'logic-through-memory' has an unreliability of quite small proportions when considered alongside the total unreliability of all the memory and processor chips in the modules it is controlling.

Voter-switch-disagreement detectors are examined extensively in work by Mathur (11), Siewiorek (14,15) and Ogus (16).
Voter-switch complexity and spare assignment strategies

Two main types of voter can be used to establish the "majority view" of a set of M modules composed of N "core" voting units and S spare modules. Generally the number of modules being voted on, N, has a fault tolerance limit of t failed modules such that:

\[ N = 2t + 1 \]

This means that t + 1 or more failures lead to a system failure.

- The Majority Voter has 2t + 1 inputs from the core and yields the output of the majority. When a core module fails, one of the S spare modules is switched in to take its place. Hence the majority voter can tolerate t + S failures (Fig. 2.9a).

- The Threshold Voter has 2t + 1 + S inputs from core and spare modules, and has a threshold of t + 1. The outputs of all modules are gated so that they can be isolated from the voter on failure and standbys switched in (Fig. 2.9b).

The threshold method avoids having to replace failed units with standbys, and spare modules are "informed" that their outputs will be required and enabled into the voter. The main drawback is that threshold logic requires a large number of AND and OR gates, especially if t and S are large. This situation may be relieved if threshold logic is manufactured in integrated circuit form. Up to now it has been assumed that voter switching allows any spare module to replace any failed module in the core. Siewiorek has suggested that a very significant reduction in switch complexity can be made by 'assigning' spares to particular sets of core modules \(^{(15)}\). He states that if each spare is assigned to at least t + 2 modules then the resultant reliability will be the same.
FIG. 2.9 MAJORITY AND THRESHOLD VOTERS
as if each spare could be assigned anywhere. This cuts down the number of switch permutations and thus should reduce the logic hardware. In addition, some modules need have no assigned spares at all without reliability sacrifice. Some specific cases such as a 3-out-of-5 core with 2 spares will work with a t + 1 assignment. This is illustrated in Fig. 2.10. Siewiorek concludes that an NMR core may in some cases be more reliable than a TMR core contrary to Mathur and Avizienis' findings. However, this depends heavily on reducing the switch complexity, and hence increasing voter-switch reliability. In addition, the TMR/hybrid degrades to a point where only one good unit is left whereas say, a 3-out-of-5 leaves two serviceable modules, unless extra switching is incorporated to allow 2-out-of-3 running.

2.5 Transient Faults and Soft Errors

The hardware designs discussed in the previous section are aimed at detecting and switching out modules which have developed permanent faults. Special measures must be taken to deal with non-permanent faults to avoid the unnecessary waste of perfectly serviceable units.

- A transient fault occurs when a logic element assumes an incorrect state for a variable length of time before returning to correct operation.

- A soft error is caused by a transient fault and is usually corrected by repeating the operation during which the transient occurred.

In the train-borne environment these errors are likely to be frequent and methods for dealing with them are discussed in the next chapter.

The design of the computer logic should be such that transient conditions
FIG. 2.10 SPARE ASSIGNMENT STRATEGIES
due to marginal timing and overloading outputs do not arise. Effects are then limited to those caused by external influences of the type discussed in Chapter 4.

2.6 Conclusion

This chapter has shown how fault-tolerant computing has taken over from the pure fault-avoidance approach as the main means of increasing reliability. The use of standby sparing to increase mission time capabilities has been demonstrated, but careful choice is required to meet safety requirements as well. The following chapters will discuss further techniques and describe some of the practical realisations of high-reliability computers.
REFERENCES FOR CHAPTER 2


3.1 Introduction

In the last chapter we have introduced the concept of redundancy into computer design, and so now we must look at some of the practical aspects. The term multiprocessing does not necessarily imply improved reliability over uniprocessor operation, although a distributed architecture can mean that failure of one part does not render the entire system useless. For example if the number-cruncher processor breaks down, the input/output processor may still allow file manipulation and transfer between memory and terminals. We have a complete loss of main function, but the system has "degraded", rather than collapsed. This type of limited operation may well have been considered at the design phase, indication that the designer has entered the world of fault-tolerant computing. The logical extension of this takes us through the areas of automatic fault detection and system reconfiguration to self-repair, the possibilities only being limited by size and cost, not it seems by the ever-advancing component technology. In this chapter we will explore the various requirements of a multiprocessing system in the kind of environment in which it is to be used. Next we will examine the hardware structures used in various designs, followed by a study of program or software features used to achieve the aim of a high-reliability computer. Following these considerations, the next chapter will go into the special requirements for automatic vehicle control.
3.2 Early Developments

The first digital computers\(^{(1)}\) were unprocessors, only capable of executing one program at a time, with a single central processor (CPU) carrying out all the tasks of number-cruncher and input/output controller. This situation continued to a large extent through most of the 1950's with most advances taking place in CPU speeds and storage system capacities. By the 1960's, however, the picture had changed, and with physically smaller units available many people were connecting computers together in order to spread the load and increase throughput. Smith\(^{(2)}\) coupled an IBM 7090 computer, which did the main processing, to an IBM 7040 which acted as a supervisor and Input/Output controller. At about the same time (1962), Baldwin et al\(^{(3)}\) joined similar machines together sharing common disc files. The simple expedient of relieving the CPU of the "housekeeping" exercises of printer, terminal, etc., control resulted in a great increase in program throughput. It was realised as early as 1956 by Von Neumann\(^{(4)}\) that by splitting up the processing and introducing redundancy, substantial increases in reliability could be achieved. However, the practical pursuit of this reliability was difficult for many years because of the component technology available. Research into multiple-computer systems subsequently followed two main paths, one stimulated by the NASA space programme and the other by commercial pressures for reliable data processing systems. Development during the sixties of the latter is summarised by Flores\(^{(5)}\). Most multiprocessing design since then has attempted to a greater or lesser extent, to introduce reliability improvement and we can now consider these designs grouped into four main areas:-
• Accuracy critical applications, e.g. real time control of air or spacecraft flight systems.

• Availability critical applications, e.g. telephone switching networks where excessive down-time costs must be avoided.

• Limited or no maintenance applications, e.g. a deep-space probe computer.

• Super computer. Component density and processing speed may lead to a very small MTBF.

There is of course some overlap between these categories and this should become clear as we examine each in turn.

3.3 Application Requirements

The desired application for the computer will not only determine whether multiprocessing needs to be used, but also strongly influence the choice of "reliable" design\(^{(6)}\).

3.3.1 Accuracy critical applications

A great deal of the research in this field is carried out by the aerospace industry, with their results and designs possibly forming the basis of work in less glamorous fields such as automatic train control. During the flight phase of a space rocket launch, the operation of the control surface systems, engine systems, etc., is performed by on-board computers because the astronauts are in no position to "drive" the rocket manually. It follows then that these computer systems must be extremely reliable and absolutely no false signals must be sent to the various actuators. A computer outage of even millisecond duration might
spell disaster. Similarly, a new breed of aircraft is being developed which is dynamically unstable and relies on continuous on-board computer operation to maintain steady flight. Once again faulty processor outputs cannot be tolerated for even very small time intervals. Down at train control level the degree of restraint on faulty outputs is largely the same because the price paid for failure in each case is probably loss of human life. Ironically it is at this level of transportation where the greatest number of people are potentially at risk, that the least money for research seems to be available. The reasons for train automation will not be discussed here as they are well presented by Milroy (7), so moving on from the computation-critical application we enter an area where brief stoppages may be acceptable.

3.3.2 Availability critical applications

In this area, generally speaking we have computers whose failure costs money rather than human lives. Telephone switching networks are a prime example with the Bell Telephone Electronic Switching System (ESS) research going as far back as 1953, the early objectives being outlined by Downing et al (8) and later developments by Toy (9). These objectives stated that no more than 0.01 percent of calls should be processed incorrectly, and few calls should be disrupted in the upheaval of recovery. Similar aims have led to the development of the AXE telephone switching system (10). In these cases it is assumed that maintenance is possible, as frequently as required, usually carried out by on-site personnel. Unlike the accuracy critical examples we have discussed, this time faulty outputs are tolerable, usually only resulting in occasional and brief inconvenience to the customer. Hence rapid fault
detection is desirable with one stage of automatic repair allowing continued operation until such time as the repairman responds to a failure indicator, and replaces the offending board. The design should be reliable enough so that the automatically switched-in spare does not fail before the normal line unit is fixed or manually replaced. A combination of accuracy and availability techniques are used in most fault-tolerant multiprocessor designs to achieve a result which will in theory produce error-free outputs for long periods of time. This leads on to the third type of application — the very long term operation — no maintenance systems.

3.3.3 Long term applications

During the late sixties, the move towards computer systems suitable for maintenance free lifetimes of years rather than hours started. The Saturn V launch computer (11) and the Orbital Astronomical Observatory (OAO) satellite computer were not considered to be suitable for the task of correct operation during the now-defunct operation "Grand Tour". The former was inadequate in that its design life was only 250 hours, and the latter considered obsolete because component redundancy assumed that the failure of adjacent components was statistically independent. Both these designs used the fault-masking approach to reliability improvement described in the last chapter, but subsequent thought given to long term mission requirements indicated that self-repair by fault detection and spare module switching was a better approach. In order to meet these tough lifetime requirements, several independent research programs started, resulting in prototype systems being constructed and tested. The first of these was the Jet Propulsion Laboratory's STAR (Self Test and Repair) Computer (12). This prototype machine incorporated many new innovations.
including a dedicated test and repair unit and was used as a test-bed for this and other design concepts, the results being published by Avizienis and others\(^{(13,14)}\). Unfortunately STAR was never tried in space as Grand Tour was abandoned. However, research continued and new designs emerged such as the Modular Spacecraft Computer (MSC)\(^{(15)}\), which is aimed at mission times of five years or more. With integrated circuit components containing denser and denser structures, computers designed around low maintenance situations may be found in areas previously covered by high-maintenance availability types because reduced costs will justify it.

3.3.4 **Super computers**

In this field we are thinking particularly of recent generations of ultra-fast parallel "number-cruncher" computers such as the Cray I\(^{(16)}\). Rennels in his review of fault-tolerant multiprocessors\(^{(6)}\) makes the point that with these very high clock rates and very dense circuitry, errors due to transients are likely to be very frequent. Indeed the MTBF for the Cray I is quoted as 4 hours. Of course this must be regarded in the light of its throughput—up to 250 million floating point operations per second. These high speed machines may well be used to process real-time picture information and the odd glitch may not ever be seen on the display screen, let alone cause serious problems so the low MTBF can be tolerated. The problem of transients in these computers will probably get worse as integrated circuits become more susceptible to high energy sub-atomic particles either occurring naturally within the packaging compounds or coming in from outside. As an example, an ITT Reliability report on 16K dynamic RAMs suggests that a system with 4000 of these devices is likely to suffer 'soft' failures at
least every 8 weeks. This is because a memory cell may now only "contain" a few hundred thousand charge units. A 5 MeV alpha particle can generate $1.4 \times 10^6$ electron hole pairs over a distance of 25 microns, sufficient to alter a cell state. The cell is not permanently damaged, but the computer should register a transient fault. Data memory corruption may as previously suggested, be reasonably acceptable, but of course program memory loss can be extremely serious. Hence fault tolerant methods of one form or another may well be needed in order to keep up a tolerable MTBF when running these "super-computers".

3.4 Hardware Structures to Achieve Reliability Performance

Having considered the areas requiring attention, we now move on to considering the various methods developed over the years for improvement of system reliability. During the late fifties and sixties, the technique known as fault masking was seen as a good method of attaining the required number of fault-free hours of space and aircraft computer operation. This has lost ground to standby redundancy in later years but will be discussed in order to show its main deficiencies.

3.4.1 Masking redundancy

Masking redundancy involves the replication of components and was first developed for relay logic, and later evolved into the "Quadding" method used with transistors. At component level, this latter approach means replacing every transistor in the simplex design with four arranged in series–parallel combination.

The main problem inherent with the design of such
circuits is establishing the relative probabilities of open and short circuit failure in each component. The passive components, resistors and capacitors, are generally assumed to have a negligible probability of failing short circuit, thus 'critical' resistors are arranged in quad parallel (Fig. 3.1). From this example we can see how difficult the design becomes, because the circuit must be able to work over a wide range of component values and currents as each resistor fails open circuit. The next problem concerns the establishment of overall system reliability at any point in time. Because of the fault masking effect, we may have failed components in the unit even before it leaves the factory as testing is all but impossible. If the unit is subjected to usage before entering normal operational service, then some parts of the circuit may well be reduced to simplex operation, with little hope of the target lifetime being achieved. Having said all this, the quadding technique does have some powerful advantages:—

- Faults are effectively detected and corrected immediately with no loss of computing time and no extra fault detecting hardware.

- Although tedious, the conversion of a simplex design to a quadded redundancy type is relatively straightforward.

Masking redundancy is not limited to the component level of course. We can have multiple copies of circuit modules or even complete systems. In these instances though, we have to introduce the voter logic as outlined in Chapter 2 (Fig. 3.2). In addition fault masking includes the realm of error detecting and correcting codes, the most famous of which is due to Hamming \(^{(19)}\). The merits of these are discussed in the section on software.
FIG. 3.1 EXAMPLE OF QUADDING

SIMPLEX DESIGN

FIG. 3.2 DUPLEX REDUNDANCY

QUADDED CIRCUIT

FIG. 3.1 EXAMPLE OF QUADDING

FIG. 3.2 DUPLEX REDUNDANCY
3.4.2 Duplicated modules and voting

The issue of system partitioning is a very contentious one, with early experiments\(^{(20)}\) suggesting that computer reliability improves the more you break it up into replaceable modules, questioned by later work on the modular spacecraft computer\(^{(15)}\). It must be remembered that the more modules there are, the greater the overhead in voters and inter-connections. If the former are little more than simple \textit{NAND} gates, then their unreliability may have little effect when considered in relation to each module's figure. However once the voters become more complex, perhaps with the ability to switch in standby spares, described later, then their contribution to the overall unreliability becomes significant. A lot of the modularisation argument is being overtaken by advances in component technology, with complete processors available as a single component. At whatever level the partitioning takes place, the duplication and voting technique remains essentially the same. In Fig. 3.2 the outputs of two modules which are identical and process the same input data, are compared by the voting logic. The voter only produces an output to the next section if the two modules agree. This gives us a "safe" system whereby the failure of one module cannot cause the system to produce incorrect outputs. However, if our voter is only a simple gate then we have no guarantee of safe or reliable operation. We are presented with three main alternatives:

- Improve the voter so that it can detect incorrect as opposed to just dissimilar inputs. This "AND" gate type of operation would shut the entire system down in the event of disagreement and would be the safety-conscious approach. However, there will be a loss of availability as transients may cause system shutdown.
• Adopt the fault-masking approach and use an "OR" gate type voter. This type of logic will need to detect which of two dissimilar inputs is correct and transfer the valid one to the output. This improves the transient response of the system, but increases vulnerability to hard faults fooling the voter.

• Incorporate elements of both the above in a design to yield a system unlikely to fall over at the first transient, but which will shutdown at the first hardware failure.

Now the problem of where to partition the system is clearer. If each module consists of an entire processor system, i.e. MPU, memory and I/O, then very little voting logic is required and its complexity relative to the modules is very low and thus its contribution to unreliability probably negligible. However, one fault causes shutdown even if one processor system is running normally, because the disagreement detector cannot tell which is the faulty unit. Such a configuration is proposed for the BR Advanced Passenger Train speed advisory system C-APT\(^{(21)}\). The basic system is illustrated in Fig. 3.3. The system monitor is the voting logic and will shut off power to the output interfaces in the event of protracted processor disagreement. The duplicated processor system also forms the heart of JPL's Fault-Tolerant Building Block Computer (FTBBC)\(^{(22)}\). However, in this case the duplicated section itself forms a module which is replicated as required.

3.4.3 Triple modular redundancy

The accuracy critical applications will require better performance than that provided by the duplicated system so far suggested and so will the long-term mission tasks. In both situations we cannot afford to have
FIG. 3.3 BASIC C-APT DUPLEX SYSTEM

FIG. 3.4 C.VMP CONFIGURATION
system shutdown at the first fault, and in order to be sure of locating the faulty unit, the voting logic will be inordinately complex with the result that its contribution to system unreliability will no longer be negligible. So at this point we introduce a third module and utilise a two-out-of-three system. A single failure, transient or permanent, will be detected by disagreement logic linking all three outputs. In addition the arrangement allows the faulty unit to be isolated, temporarily or permanently. TMR can be used in the form of masking redundancy whereby the voter logic decides on a majority basis which output is correct and then passes this information onto the next module, taking no action against the faulty unit. However, the current trend is to use TMR as the basis of a standby sparing system. The component level has contained integrated circuits for many years and testing is largely limited to logic checks rather than confirming the operation of individual transistors and capacitors. TMR used in the masking redundancy technique means that the redundant units can usually be tested more easily than their quadded component counterparts, but in spite of this, the standby sparing system techniques have gained favour over masking. Having decided that TMR is a good method of improving reliability both intuitively and theoretically from Chapter 2, some of the pitfalls must be examined before rushing into a final design. Without sufficient thought being given to the type of fault to be dealt with, the desired reliability goal may prove illusive.

3.4.4 System partitioning

Looking in detail at the hardware of a TMR processor system we can isolate areas of the circuitry which may not be redundantly replicated and thus find potential weaknesses. The first item to be considered
when designing, is what each module to be triplicated will contain. Before the microprocessor chip became really established, many designs using TTL devices for example, split what would be termed the central processor unit up into modules for triplication. Erwin\(^{(23)}\) discusses a spacecraft computer proposal whereby the CPU is split into Arithmetic and Control units and each replicated. Generally the trend now with MSI and LSI components available, is towards replication of the complete CPU with similar treatment for the I/O and Memory modules, or indeed replication of the complete CPU, Memory and I/O as one module. An example of the former is the C.vmp\(^{(24)}\) built at Carnegie-Mellon University. The C.m* built at the same place is an example of the latter approach. The C.vmp (Fig. 3.4) uses three identical processors connected through a common voting block to three memory and disk modules, with voting on data moving in either direction. The system is in fact fault-masking in that incorrect data on one bus can be corrected providing the other two busses agree. This self-checking computer system module can then communicate with other similar units to form part of a larger standby sparing system. Cm* on the other hand, consists of a network of "clusters" (Fig. 3.5), each containing up to fourteen identical processor-I/O-memory modules. Each of these clusters is linked into the network by another processor known as the Kmap which acts as a single error checker and configuration controller, allowing faulty clusters to be bypassed. The FTMP described by Hopkins\(^{(25,26)}\), was developed from earlier considerations based on processor duplication\(^{(27)}\) and uses serial communication along redundant busses linking as many processor modules as required to their memory modules (Fig. 3.6). The processors are grouped together in threes forming "triads", as are the memories and I/O units. The members of a processor triad run as a TMR
FIG. 3.5 CM* STRUCTURE

FIG. 3.6 FTMP STRUCTURE
system in tight synchronism. All these systems are capable of detecting faulty units and rearranging their operating modes so as to ignore these failed sections. This type of redundancy - self reconfiguration with TMR as the basis for fault detection - is usually termed hybrid redundancy. Bouricius et al (28) indicated that TMR alone is probably the best choice for short mission times, but an ability to reconfigure in the event of failure is necessary for long-term applications. Of course, the basic TMR system does have an ability to withstand failure with no degradation in system throughput. The voting logic can be designed so that in the event of the total failure of, say, a processor module, the other two may carry on until such time as they disagree whereupon both must shutdown. It can be seen that TMR is very useful for detecting and either masking or triggering correction of permanent faults. However, what about transients? If the transient is just a brief and obscure aberration in one of the triplicated units, then TMR will usually cope with it, but what if it is due to some common mode effect, causing simultaneous errors in all three? The power supply lines are one such source of transients, and common clock circuits another. Shared data and/or address bus lines while being a weakness as far as permanent failure is concerned, help prevent common mode transient effects getting in this way, as only one processor will have control at any one instant. The areas in a system where single failures can cause total systems shutdown are termed "hard-core", and must be eliminated or circumvented. Methods to deal with those particular problems will be dealt with in a later section, but meanwhile let us examine some of the reconfiguration techniques.
3.4.5 Fault-detection and system reconfiguration

The two topics of fault detection and reconfiguration are grouped together, because the former must precede operation of the latter and the hardware to perform each function is often combined.

Firstly then, let us consider ways of detecting an error in processing which may be attributable either to a transient or permanent failure. There are three main hardware methods available:-

- Hardware parity generators/checkers on all inter- and intraprocessing communication busses. This method is suitable for use on non-redundant simplex systems but, can be used on redundant types.
- Data comparators and voters on bus links in duplicated or TMR systems.
- k-out-of-n checkers on processor busses. This method normally requires software generation of the data pattern, but once this is done, detection hardware is minimal.

Hardware parity detection was used extensively on the Saturn V launch computer (11). Data comparison and control signal voting are used extensively, although most systems use the technique as part of a software controlled checking system. The BR C-APT duplicated processor system uses a control signal monitor circuit which utilises special purpose check outputs from each computer to establish system integrity. The k-out-of-n code checking method was used extensively on the JPL STAR computer. All these hardware error detectors are each normally wired in one place and will respond immediately to incorrect signals or data. An interesting variation consists of a "roving comparison" unit as used in the Westinghouse Micronet System (29). This system consists of processor "strings" arranged in parallel (Fig. 3.7) to yield a pipeline type data processor.

Each string contains spare processors with one of the latter being used as
a checker paired with one of the active units and manipulating the same
data batch. Outputs are compared and the check processor then moves on
to the next active unit along the string to check it’s operation.
If a disagreement occurs then the active unit is assumed to be
faulty and replaced, but if the checker disagrees with several processors
then the former is replaced. This type of error control is only
suitable for this type of large volume batch processing where faulty
outputs can be tolerated for a short time. Comparison is in software
but the machine is mentioned here because of its unusual hardware
configuration. Much research has gone into the subject of reconfiguration
and trends in design are becoming clearer. The JPL STAR computer
featured for the first time, a Test-and-Repair Processor (TARP). This
unit monitored all the system operations, detected faults and switched
spares in and out as necessary. Such a processor was potentially a weak
link in the system and was operated in full TMR with standby spares
(Fig. 3.8). Most early designs involving reconfiguration with standby
spares seemed to involve these special "fault processors". Erwin\(^{(23)}\)
describes a standby-spared design controlled from a Configuration Assign-
ment Unit (CAU) run in TMR. The Modular Spacecraft Computer described by
Conn et al\(^{(15)}\) uses triplicated Configuration Control Processors (CCP)
to trigger and synchronise data movements rather like the normal timing
unit in a conventional design. This task is in addition to their
monitoring and error detection role. Later designs seem to favour
distribution of the reconfiguration power away from one central unit no
matter how well protected with TMR it may be. As mentioned before, the
Cm* architecture is arranged as a network of multiprocessor clusters linked
to common bus systems via intelligent bus coupling units known as Kmaps.
These Kmaps constitute the distributed reconfiguration system using a
FIG. 3.7 MICRONET STRUCTURE

FIG. 3.8 JPL STAR STRUCTURE
sophisticated virtual memory mapping arrangement. The Kmap processors have the capacity to detect and correct single bit errors, with a periodic diagnostic program running to "cut-out" failed clusters. FTMP also has a distributed reconfiguration intelligence, but the bus couplers between individual processors, stores and the busses are not as intelligent as the Kmaps and each uses duplicated bus guardian units to control bus access. A processor module, memory module, I/O module and control logic make up a single line replaceable unit (LRU). A complete system consists of ten LRUs arranged as three triads and a spare. Hence although each LRU contains a configuration control unit for the modules within it, the overall multiprocessor system appears as a set of identical boxes with no separate and unique control unit. The SIFT computer for air-borne applications described by Wensley et al\(^{(30)}\), relies heavily on software for error detection and system restructuring. The basic hardware system illustrated in Fig. 3.9 consists of a number of main processors connected to memory units and I/O processors via hardware controllers on a redundant bus system. Apart from the bus redundancy there is very little hardware dedicated to fault detection and correction, software comparison techniques being used extensively as and when required. Each bus has its own hardware controller consisting of a simple microprogrammed unit of a complexity estimated at 10% of a main processor. The processor and memories also have their own interface controller with the result that any processor can 'talk' to any memory via a system of flag requests between the various interface units and bus controllers concerned. An interesting difference between the SIFT and the FTMP lies in their clocking arrangements, the former using asynchronous operation, the latter running its processors in tight synchronism. The relative merits of these approaches will be discussed later. Ornstein\(^{(31)}\) and Katsuki\(^{(32)}\)
FIG. 3.9 SIFT STRUCTURE

FIG. 3.10 FTMP REDUNDANT CLOCK
describe the Pluribus multiprocessor system which uses commercial
Lockheed SUE processors linked via redundant bus couplers to memory and
I/O units. Like SIFT, fault detection and reconfiguration is carried
out by program, but unlike SIFT, Pluribus was designed for static
use as a communications mode. However, the physical partitioning of
hardware described by Katsuki may be useful for other applications.
Both SIFT and Pluribus feature distributed reconfiguration 'intelligence'
to reduce the likelihood of a single failure knocking out the whole system.

Much of this intelligence is due to software within each processor
keeping track of failed units and making use of redundant hardware busses
to by-pass them. An interesting form of internal processor recon-
figuration is described by Miller\textsuperscript{(33)}. The Variable Instruction Computer
(VIC) employs duplication within the processing unit. The normal macro-
instruction to be obeyed by the processor is carried out using several
micro-instructions as in a conventional computer. However, the hardware
is designed so that many different combinations of micro-instructions may
be used to yield the same macro-instruction. The redundant duplication of
key registers and controls within the processor leads to a 'multi-threading'
design able to by-pass failed gates and flip-flops. Error detection is
achieved by hardware operated parity generation and checking. VIC has
subsequently been updated into a multiprocessor system described by
Dietrich\textsuperscript{(34)}. All the hardware designs discussed so far have attempted
to eliminate what are termed 'hardcore' areas within the computer
system as a whole. These are areas of circuitry whose failure due to a
single fault will lead to total breakdown. The partitioning or breaking-up
of a system into obvious modules of processor, memory, etc., will not
be enough, and further investigation of vulnerable areas is needed.
These vulnerable areas may need triplication or a distribution of their function amongst the other modules. Hardcore problems needing consideration are as follows:—

- **Timing.** An obvious trap to fall into is to replicate all the processors in the system, but to control their timing from a single clock source.

- **Busses.** If all the processors in the system share the same bus, then its failure will lead to total collapse of operation.

- **Module links.** Many proposed systems rely on reconfiguration to avoid failed units. For this to work, every module needs to be linked to every other with a heavy requirement for the hardware switching reliability.

Approaches to these problems will now be covered in more detail.

3.4.6 **System timing**

Having designed the processor clock and timing logic as a hardcore problem area, its elimination as a weakness must be considered. A quick study reveals that the whole question of inter-processor synchronisation will also need to be examined. There are two main options available:—

- Run all processors in tight synchronism and devise a triplicated redundant clock system.

- Drive each processor from its own clock and run the whole system asynchronously.

The first method is that adopted in the FTMP computer. Each LRU of the system has its own clock generator, with four being considered active.
Four clock lines connect to all units, each of the latter containing a clock receiver able to use any three of the available signals to produce a valid clock (Fig. 3.10). Use is made of phase-locked loops to ensure phase lock despite oscillator failure. Davies and Wakerly describe a similar fault-tolerant clocking system in their paper on synchronisation as does de Souza. All these clocking systems use oscillators controlled by majority voting from the others. The result is usually three or more phase-locked clock signals which are fed to the inputs of a majority voter within each processor module. While these solutions to the timing hardcore problem are effective in reducing vulnerability to hard faults, the tight synchronism of all processors may allow transient common-mode faults to go undetected, as a brief 'glitch' on power rails may yield identical incorrect results in all three processors of a TMR system. The FTMP is tightly synchronised and the designers seem to be relying heavily on physical shielding to prevent external transients from having the described effect. The PTBSC uses duplicated processors with tight synchronism in it's core unit, but communication with other building block units is not synchronous and thus it may be able to detect common-mode transients if configured to do so. The SIFT computer uses a 'loose' synchronism which represents a compromise between the tight operation necessary for output comparison and checking, and completely asynchronous running. Typically, processors are only synchronised to within 50 μS and hence some measure of common-mode transient immunity is achieved. The totally out-of-synchronism technique tends to make inter-processor result comparison difficult and two or three supposedly identical programs run in different processors may diverge, leading to shutdown even though no real fault exists. Hence
loose synchronism appears to offer the best choice, although it usually results in more complex hardware and/or software. Consider these approaches:

- Processors are synchronised to within a single instruction interval, and all or most processor-memory transactions are checked for identity either in software or by skew-tolerant hardware.
- Processors run in a very loosely coupled mode but notifying each other perhaps via discrete signal or common memory, when each has either a change of input data or it wishes to change a system output. Agreement would be necessary before a change of input is accepted or modified output allowed.

Both these techniques have been tried at BR Derby and seem to be successful\(^{(21)}\). The Space Shuttle Computer\(^{(37)}\) uses software extensively with the second approach above, such that processors are brought into synchronism as soon as an input or output request is initiated. Sub-systems such as the flight-control effectors then operate after an at least three-out-of-four vote. Before leaving the topic of timing, thought should be given to recovery from transient effects. The loose coupling of processors and voting should mean that no recovery is required as the fault will be 'ignored'. However, transients of sufficient duration to confuse the system may occur and will result in incorrect operation. The subject of recovery will be dealt with in a later section.

### 3.4.7 System busses

The next hardcore area to be considered concerns the form of the communication network within the system. It is very tempting, and very easy to use a single data 'highway' linking all processors and memories, etc.
But of course it only takes one line of the bus to fail, or one bus buffer to fail driving high or low, and the whole system is almost certainly doomed to crash. The answer is of course to replicate the bus, but to what extent? Several methods are available:

- Simple replication involving spare copies of the main busses, either operating in masking mode or as standby spares.
- Replication of busses, but all available for simultaneous use by different processors.
- A two level bus structure with 'local' busses shared by say three processors and their memories, linked to a system bus network.

The first option does not really get off the ground because of all the wasted bus capacity. The standby method is probably a non-starter because all the extra switch complexity would swamp the increase in overall reliability. The second method is attractive in that the extra busses are used either as part of the error checking arrangement or to increase throughput. Sklaroff\(^{(37)}\) describes a system where there are four redundant serial busses all connected to each of four identical processors. Redundant sub-systems are each coupled to a different bus line. In this way a bus can be cut off or ignored by the processors if a faulty system cannot be prevented from 'babbling' unwanted outputs. At the same time a faulty processor can be taken out without loss of system function.

The third option is being widely used and is likely to continue growing in importance as microprocessors improve in performance and speed. The Cm* computer is arranged so that a local cluster is connected to three inter-cluster system busses\(^{(24)}\). Not only can information be passed between indirectly connected clusters (Fig. 3.5) but in the event of bus or cluster failure, re-routing is possible albeit with some
performance degradation. The FTMP bus structure uses a local bus within each processor-memory-I/O LRU, but voting triads communicate internally and with other triads on redundant serial lines in a similar way to the space shuttle computer. An interesting network is described by Bernhardt and Schmitter(38). Local microprocessor systems each based on an 8086 main processor and 8085 I/O processor, are linked such that each local module communicates with its next and next-but-one neighbours. In this way each module is 'monitored' by two others to detect failure and implement reconfiguration if necessary (Fig. 3.11). The main trouble with the basic system described is that the possibility exists of a local processor developing a fault and sending bad signals to any control effectors before two other modules realise and intervene. Indeed in many designs, this error output latency may be a serious problem.

Having introduced a redundant or multiple bus system, we have also brought in more potentially unreliable components - bus connectors and switches.

3.4.8 Voters and switches

The early definition of TMR(4) involved the use of redundant system modules, but has been modified by introducing the principle of standby sparing. As soon as we enter this field of what is termed "hybrid" redundancy, we are incorporating the changeover switch into the system logic. The voters remain of course, but also needed will be disagreement detectors. Although the voter and disagreement detector might appear to be the same thing, in fact the former will present the "majority view", assumed to be the correct one, while the latter will locate which module(s) are at variance with the rest. Mathur and Avizienis(39) discuss the reliability improvement to be had by using standby spares. Their conclusion derived from the general case of N-Modular Redundancy(NMR)
FIG. 3.11 BFS STRUCTURE

FIG. 3.12 BASIC VSD LOGIC
is that an active triple modular core (i.e. $N = 3$) is best with as many standby spares as necessary to achieve the desired mission reliability. This is true if the standby spare reliability is much greater than the active unit reliability. This seems likely to be the case, particularly if the spares are unpowered - cold standby as opposed to hot standby. Mathur and Avizienis show the voter (or as they term it restoring organ), disagreement detector and switching logic combined as one unit known as the VSD (RSD) (Fig. 3.12). The practical realisation of the VSD tends to rival the processing modules in complexity, and so the whole concept is in danger of falling apart if the reliability figure for the VSD cannot be made significantly better than that of the modules it controls. To this end, Siewiorek and McCluskey\(^{(40)}\) developed the iterative cell switch to reduce the component count of the VSD. They have also described various switching strategies to simplify the switch further\(^{(41)}\). Switch complexity can only be reduced so far, and if further improvements in its reliability are to be achieved, then it too may need to be made fault tolerant. Ogus\(^{(42)}\) takes the basic VSD and adds redundant components to improve long term reliability and to make it 'fail-safe'. The latter means that a fault in the switch cannot cause a known failed unit to be switched back in, although good units may be switched out. Ogus also determined that a theoretical maximum number of spares exists, after which reliability is degraded. The iterative cell switch suffers from a large component count if simple SSI gates are used to implement it, but this may change if LSI models are produced. It does have the advantage of generality, in that it applies to NMR systems with any number of spares and that the switch logic, possibly using EPROM can be modified to suit different strategies. For instance a retry capability was suggested\(^{(40)}\) whereby failed modules could be
tried again periodically. Transient failures could be overcome in this way. This type of switch mechanism lends itself to switching complete computer modules around rather than just sub-systems such as memories. The main snag is that cold-standby units will need to be "briefed", i.e. registers and memories pre-loaded before being allowed access to the voter, whereas if applied to say read-only memory sub-systems, no such briefing is required. Goldberg et al describe a fault tolerant memory system based on LSI memory chips which is capable of self-reconfiguration although the design suggested would use external error detection and change signals. The main emphasis of the approach is towards incorporating the extra logic within future chip designs and may not have significant advantage over block replacement otherwise. Luque et al describe a fault tolerant memory system which although duplex can detect any errors in the main memory unit. It requires a second memory module containing the upper and lower halves of each word combined via an encoder, typically an exclusive-OR gate. For instance if a data byte is 8 bits wide, then the check memory unit is only 4 bits wide. The check hardware is limited to a few exclusive OR gates and comparators. These two memory systems illustrate a different approach to reliability design from that discussed up to now, whereby a particular module is singled out for special treatment. This technique may well have the advantage of reducing the hardware overhead, but degrades the modularity of system design making reliability calculation more difficult. Whichever design path is taken, software will form at least part of the reliability strategy if corrective action and recovery are needed.
3.5 Software Techniques to Achieve Reliability Performance

The question of where to draw the line between hardware and software dedicated to improving reliability is a difficult one and Chandy et al (46) make the broad assumption that extra hardware involves capital cost whereas extra software incurs a processing time penalty. Hardware costs have dropped dramatically since then, and given the current technology, the cost of developing complex software may well far outweigh the hardware expenditure. Nevertheless their theoretical approach using Hardware and Software Reliability Efficiency Indices (HRE, SRE) together with a Real-Time Criticality Index (RTC) may at least form a basis for discussion. Although software development may cost a lot of money, the extra processing time needed to run it could well render it unusable even if the money were available. Many real-time control systems such as missile and spacecraft flight controls impose severe restrictions on the amount of CPU time available for non-data processing tasks. Hence the RTC for such projects would be very large. Set against this desire or need to reduce the lateness penalty incurred by the error checking and recovery software, is the mission failure penalty. Fortunately improved hardware speed is allowing an increasing amount of operational checks to be carried out by program and some techniques will now be discussed.

3.5.1 Code checking while running programs

Both data and instructions held in store can be checked by software for correctness if they are coded in a particular way. The easiest approach has been simply to append a parity bit to each store byte or word. The hardware overhead mainly involves the extra memory capacity and simple parity operating and checking software is relatively compact.
Unless the processor is custom made, then parity checks on instructions may need to be done by external hardware, although generation will probably be a once-only exercise when the program is inserted into (say) PROM. Indeed, software checks on all data words may be undesirable from a speed point of view, although some microprocessors, notably the 8080 feature test parity and jump instructions, which may speed things up if data access requirements are not excessive. Odd/Even parity checks will detect single or odd numbers of bits in error but will fail completely to notice even numbers of incorrect bits in the byte or word. The parity check does not of course give any indication of which bit is in error, and to do this more redundant bits need to be added. For example the Single Error Detecting code due to Hamming \(^{(19)}\) needs three redundant error check bits to correct single errors in a data word of length seven bits, i.e. the code protects four information bits. These sort of codes need to be 'stripped-out' of the data before application to an arithmetic logic unit. Codes which are preserved after arithmetic operation are called arithmetic error codes \(^{(47)}\). These were extensively used on the JPL STAR computer \(^{(12)}\), combined with special-purpose hardware to allow concurrent data flow and checking with no time penalty. The STAR used an inverse residue modulo 15 code on the addresses and operand words, with a 2-out-of-4 code to protect the operation code. The definition for the residue code is:

\[
r = (15 - n \mod 15)
\]

where \(n\) is the data and \(r\) is the check code. As \(r\) cannot be greater than 14, there is a need for only four check bits conveniently put on the end of the main data word. Decoding can be done in software although the STAR use an end around carry adder unit to give a residue of 1111
in binary for a correct word. The operation code is twelve bits long but when 2-out-of-4 code is applied to each of its four-bit sections, only \(6 \times 6 \times 6 = 216\) possible codes remain. This is because only two out of the four bits may be a logic 1, thus limiting the number of possible codes to six. This k-out-of-n code and the Hamming codes are termed non-separable because the basis of detection is tied up with the particular pattern of ones and zeros. Single parity, Berger codes and the residue code just described are separable so that the redundant check bit(s) are simply appended. The Berger code involves counting the number of 'ones' in the data, expressing this number in binary and then taking the one's complement. This number now becomes the check attached to the data. Checking is carried out by complementing the check and comparing with the data. For example if we have seven bits of data then we will need three check bits, producing a code word of ten bits.

e.g. Data = 0110101 Check = 011

Berger has shown that this type of code is more effective in detecting bi-directional or symmetric double errors than a k-out-of-n code. A bi-directional error is one in which incorrect changes from 0 to 1 and 1 to 0 are equally likely. A unidirectional error checking system assumes that failure only occurs to one state. Clearly simple parity will not detect a double unidirectional error in a data byte, or errors where a 0 changes to a 1 and is compensated for by 1 changing to a 0. The Berger code improves on this but at the expense of three redundant check bits requiring extra memory hardware. Simple parity has nevertheless seen extensive usage in computing because it is separable, easy to decode with a few gates or in software, and only requires one extra memory bit per data word. The non-separable codes therefore have lagged behind, inspite of decoding and handling overheads. Even the Berger
code has not seen much use because of the summing hardware or software required. More recent research has centred on developing separable error-correcting codes compatible with the current parity codes\(^{(48)}\).

Pradhan and Stiffler describe a code capable of correcting all single bit errors and detecting all multiple errors of the undirectional type. In order to protect two data bits, \(u_1\), \(u_2\), however, four redundant check bits \(p_1\), \(p_2\), \(p_3\) and \(p_4\) are needed such that:

\[
\begin{align*}
\ p_1 &= u_1 \oplus 1 \\
\ p_2 &= u_2 \oplus 1 \\
\ p_3 &= u_1 \oplus u_2 \\
\ p_4 &= u_1 \oplus u_2 \oplus 1
\end{align*}
\]

On reception of the encoded data word, a four bit error syndrome is generated:

\[
\begin{align*}
\ s_1 &= p_1 \oplus u_1 \oplus 1 \\
\ s_2 &= p_2 \oplus u_2 \oplus 1 \\
\ s_3 &= p_3 \oplus u_1 \oplus u_2 \\
\ s_4 &= p_4 \oplus u_1 \oplus u_2 \oplus 1
\end{align*}
\]

Examination of this four bit error syndrome will reveal whether the data is correct (\(p_1 = p_2 = p_3 = p_4 = 0\)) or certain combinations will indicate the position of a single error, while other combinations will at least indicate the presence of multiple unidirectional errors.

Pradhan and Stiffler conclude that this type of code is well suited to transient error detection in fault tolerant computers, basing this on two main assumptions:

* Single random errors due to environmental conditions are likely to be bidirectional.
• Burst errors are likely to be unidirectional.

Given the large number of redundant bits, this method may be best suited to serial communication channels in order to keep extra hardware to a minimum. The FTBCC\(^{(22)}\) uses a combination of Hamming codes and byte parity to ensure correct transfer of data between the memory and duplicated CPUs within the self checking core module. The Hamming code is used in conjunction with redundant memory bits (16 bits data + 6 bits check + 1 bit spare, per word), to isolate a faulty bit and replace it with the spare. This illustrates a technique whereby if one bit in a particular word fails, you assume that all the corresponding bits in all the other address locations have failed and so replace the whole bit 'column'. With semiconductor memories organised \(Nk \times 1\) bit, switching between two chips is probably all that is required. The argument is that this switching is faster than the continual correcting of the faulty bit.

Under the heading of software techniques are grouped a series of programming methods and 'tricks' which can greatly improve the system's chances of recovery to a valid state after transient or semi-permanent corruption. These ideas are described by Yarkoni and Wharton\(^{(50)}\) and are summarised here because they were originally developed for automotive applications, and it is felt that they are of considerable interest to a train control designer.

• If possible the entire address capacity of the computer should be filled. All unused store should be filled with No Operation codes (NOP) except for jump instructions at intervals to send the processor to a recovery routine. This should be possible with the 8 bit microprocessors likely to be used, each usually having 16 address lines and hence a 64K byte address range. This action should result in at least eventual recovery if the program counter is 'jumped'
by a transient yielding an otherwise unused address.

- Multiple-byte instructions used with most 8-bit microprocessors can mean that even if a corrupted program counter address lands in program space, it may pick up an address byte as an op-code. This is very difficult to guard against, except to try and ensure that critical op-codes do not appear as the address bytes in absolute address instructions. Critical op-codes include Disable Interrupts.

- Add redundant routines to check the validity of return addresses on the stack before returning from subroutines or interrupts. If the program lives in the bottom of the store, then large return addresses should be suspect.

Yarkoni and Wharton have shown experimentally, that use of these techniques yields a useful software "safety net" for some classes of transient fault.

3.5.2 Self-test while not running programs

Error checking while running the application program(s) may not be enough to ensure that there are no 'lurking' hard faults. Both SIFT\(^{(30)}\) and FTMP\(^{(25)}\) run diagnostics at intervals to detect faults in little used sections of processor or memory. For example a particular instruction may only rarely be used, in itself an acceptable situation because we can rely on the normal error correcting system in the event of failure when used. But should a second fault occur while the first remains undetected then it is conceivable that the two combined might lead to system failure. Self-test programs may be run at three different times:
• At power-up reset in order to make sure that the system is fully functional before beginning any application software. If this is not done, then any automatic spare switching may mask faults and the mission may start with reliability seriously reduced.

• On major system failure where the normal recovery software and hardware has been unable to restore system function.

• At intervals while running application programs. A jump to self-test may well be incorporated in loops where it is known that the processor will be spending a lot of time waiting for an external signal.

The major assumption when writing a self-test program is that no resources required to run the test are faulty, which often severely restricts the areas in which faults are permissible. Steps must be taken to ensure using only the minimum amount of CPU and memory capacity to conduct the test. The processor must test itself first using instructions that use as little of its internal hardware as possible. Ballard\(^{(51)}\) describes the construction of a 'weight table' listing instruction type together with the registers it uses, whether or not it uses the ALU, operation of carry and other flags, etc. Instructions with the lowest weighting of resources are used by the check program to start with to test the other more complex operations. Once the processor is found to be fault free we can move on to the RAM. Algorithms for RAM testing abound and will not be covered here, but the reader is referred to Nair et al\(^{(52)}\) where a functional test is described and references to other types listed. It is assumed of course that the ROM which holds the test program is itself not faulty. Address and data line failures within the chip are likely to be catastrophic, but bit
failures not so if they do not occur in the check program area.

ROM integrity can be checked by calculating and inserting a check word at the end of the stored data when it is initially programmed. Every column of data bits within the ROM is arranged to have odd parity by means of the check word bit. Wood\(^{53}\) suggests that this technique will detect all single bit faults and a large proportion of address and data line failures. Having detected a hard fault the processor must attempt reconfiguration or shut down. It is quite likely however, that the failed processor-memory system will not shut down and will go wild possibly with dangerous consequences. A major advantage of the multiprocessor system is that the other working processors may send status messages to each other on successful completion of their own tests and two functioning units can vote to cut-out an inoperable unit. For safety reasons, a fuse-blowing arrangement may be preferred in train systems in order to eliminate any chance of a failed unit getting back on line.

3.5.3 Rollback and retry

On detection of an error, an instruction or operation may be repeated one or more times until success is achieved or hard failure is confirmed (Fig. 3.13). This principle of rollback and retry has been used extensively for many years mainly for correcting mis-reads of magnetic tape or disc storage, but has been extended to cover CPU malfunctions\(^{54}\). If a large number of computer malfunctions are in fact transient, something which experience has shown to be the case, then instruction retry may save a lot of time wasted on fruitless diagnostic testing. The decision whether or not to use rollback and retry and if so in what form, is affected by these inter-related factors:
FIG. 3.13 ROLLBACK AND RETRY
• How soon after the malfunction the computer notices the error and thus how many memory and register contents have been affected;

• What the rate of transient malfunction is likely to be;

• How often the computer makes a separate copy of correct register status, etc., and forms a rollback destination;

• Whether the system application requires high-speed continuous processing, high availability with down times of seconds duration tolerable, or something in between.

Chandy and Ramamoorthy\(^{(55,56)}\) outline the problem of where and how to insert checkpoints within a given program. An analytical method is proposed\(^{(55)}\) to determine whether or not a particular program requires rollback points and if so where they should be inserted, using parameters such as the Maximum Recovery Time, and the Maximum Load and Save Times of the recovery routines. The Authors conclude that the worst-case design should be aimed at initially because reliable data on Expected times is difficult to determine in advance. This type of checkpoint insertion is dynamic, that is to say that decisions on rollback points are made according to individual program conditions. Static rollback simply involves saving program status at regular fixed intervals. The factors affecting the choice of interval are the same as for dynamic: status-saving too frequent wastes processor time if errors are infrequent; intervals between save points too long results in excessive recovery time if errors are frequent. However the static approach involves a lot less consideration than the dynamic and may be economically more viable. In the proposed Cyclone computer\(^{(57)}\), the saving of program status at regular intervals is accomplished using
hardware interrupts, with the interval between status-saves switch selectable. The term "saving processor status" has been used and can be taken to mean anything from just dumping MPU registers into memory to making another copy of the complete scratchpad memory contents. The latter approach would probably result in too high an overhead in processing time if done frequently, but if blocks of data remain unchanged for long intervals then copies used when the main storage is corrupted may mean the difference between a successful and an aborted mission. An example of such operation in automatic train driving could be the storing of multiple copies of data obtained from trackside transmitters which in processor terms means that it does not (or should not) change for long intervals i.e. minutes. An extension of retry is proposed by Shedletsky (58) called Alternate-Data Retry (ADR). This technique enables stuck-at-0 or 1 hard faults on say system busses to be by-passed when the data is sent again after the first was rejected by the receiver. For example if unit 1 sends data with odd parity 0001 to unit 2, but unit 2 actually receives 1001 due to a line being stuck-at-1, the even parity signals an error and unit 2 requests the data again. This time however, the one's complement is sent, 1110 which the unit 2 correctly receives in spite of the fault. Such a technique may well be useful for maintaining interprocessor communication along parallel busses.

3.6 Special Hardware Recovery Techniques

If faults are not quickly detected and errors propagate (56) to the point where the system or part of it collapses into chaos, then extra hardware measures need to be taken to effect recovery.
3.6.1 Watchdog timers

In principle the watchdog timer measures the time taken for the computer to carry out a certain task and if this exceeds tolerance limits then a reset or interrupt signal is generated. The assumption is that an overlong execution time implies that the processor is no longer carrying out its assigned job. Various approaches are possible:-

- The processor presets the timer itself to a value it thinks appropriate before executing a specific task \(53\).
- A hardware timer times out if no activity on certain control lines occurs over a fixed interval \(32\).

The first method requires some complex software to decide on the required interval for say, a subroutine, but does have the advantage that fault detection may be rapid before too much damage is done. The second method has been used for years as a "last resort" mechanism to force system shutdown or reset and checkout. It is easy to implement with a re-triggerable monostable, but to cater for all time intervals allowed in normal running, the interval may have to be long during which much store corruption may have taken place.

3.6.2 Illegal address detection

It is highly unlikely that all the address space available to the microprocessor (typically 64K bytes) will be used. However, the unused chip select lines from the address decoders can be linked to an error condition detector which will generate a reset or interrupt if any of these lines activates. This can only happen if the processor has 'gone wild' or if the decoders fail, either of which will need
attention. In both this case and the watchdog timer, the interrupt will force the processor(s) if capable, to run self-test routines as previously described. Successful test completion can lead to system or module restart, on the assumption that the cause of the problem was a transient. The problem is to decide whether RAM data is still valid. Range tests on data may be useful and if unused RAM was filled with a check pattern on initial power up this could now be checked to see if the transient corrupted it. If say two-thirds of scratchpad RAM is unused then the integrity of the used one third may be assumed with some confidence if the former is uncorrupted. 

3.6.3 AC output coupling

Critical output lines liable to suffer from stuck-at-1 or 0 faults may be isolated by capacitors from the peripheral control device. A continuous oscillation of the output line being required in order for the signal to be passed on.

3.7 Conclusion

This chapter has introduced many of the hardware and software techniques developed over the last twenty years or so to improve the reliability of computers by making them fault-tolerant. The examples of machines actually built to test ideas have shown many of them to be valid. Not all are applicable to the particular subject of train control and the constraints on the design of such systems will be discussed in the next chapter,
REFERENCES FOR CHAPTER 3


3. BALDWIN, F. R., ET AL: "A multiprocessor approach to a large computer system", IBM System Journal 1, Sept. 1962, pp.64-76.


17. ITT RELIABILITY REPORT: Quality and Reliability in Memories, Issue 2, No. 6251/2318E.


4.1 Comparison of Mobile and Static Computer Installations

Ever since the first electronic stored-program machines were built (1), computers have tended to be characterised as bulky and heavy, but at the same time delicate and easily damaged. Of course, such devices had to be housed in special rooms in carefully controlled environments if some measure of reliable and continuous operation was to be achieved. This mobile operation remained largely a dream until the semiconductor transistor replaced the thermionic valve as the prime electronic switching element. However, the major advances of on-board mobile computer operation have come with the advent of the integrated circuit. Thanks largely to the need for small computers to fit into space vehicles, we now have a technology applicable to other transportation systems.

4.1.1 Physical size and weight restrictions

Space and weight are not usually among the primary considerations when designing a fixed computer installation, but can impose severe if not intolerable restrictions especially if the machine needs to be mobile to the extent of being put into earth orbit. Most early spacecraft equipment was of the simplex type with an absolute minimum of back-up or safety devices. Reliability was achieved by extensive testing and burning-in to weed out faulty components and indifferent design. This technique is inherently expensive and has not been considered generally viable when it comes to ordinary commercial
applications such as the automatic train control discussed here. In addition the number of lives put at risk by faulty operation is much greater in train control than say satellite operation. With tens of thousands of transistors on a single large scale integrated (LSI) chip, at last we can make small computer systems of great complexity with a considerable amount of capacity available for back-up and safety systems. In the train control application we have space available in the train driving cab and perhaps also in compartments in each train carriage. This distributed computing power is currently being developed for the advanced passenger train where there is a requirement for a coach tilting control system. Such is the reduction in size of the basic processor system, that the main space and weight problems may well only involve the input/output interfaces and power supplies. Unfortunately the former may still be electromechanical relays especially when switching very large d.c. currents to traction motors and the like. Mechanical switching is likely to be replaced by thyristor control and so size and weight reductions can be achieved in this area. Power supply requirements have been reduced drastically with the use of MOS integrated circuit technology and with the introduction of switching supplies, large currents can be drawn from much smaller units than the series regulator types used before.

4.1.2 Electrical noise problems

In a fixed computer installation we can endeavour to remove and suppress all sources of electrical interference. Electrical machines can be housed well away from the computer and if necessary the computer room can be built with extensive electrostatic and electromagnetic shielding. In the design of vehicle borne systems we are often faced
with old and electrically very noisy motors and contactors, which will be sitting only a few feet from the computers in many instances. Although contactless thyristor switching removes some of the arcing problems, motor brushes will continue to be a problem. (Fig. 4.1). The results of experiments carried out at British Rail, Derby[5], have indicated that noise pick-up is so bad that an oscilloscope with its input grounded still shows a considerable trace. Thus it can be seen that the problems of fitting a new and sophisticated electronic control system to an existing railway train may make design work more difficult than when the whole project is designed and built from scratch. Interference enters the computer control system on two main routes:-

- Interference entering on the power supply rails
- Noise pick-up on control lines to relays, etc., and on sense lines, for example from wheel-mounted slotted-disc tachometers

Although the first source of corruption may seem simple enough to cure with adequate power supply units and filtering, experience, sometimes bitter, has shown that noise "spikes" still get through. The class 304 electric multiple units (EMU) of British Rail use an overhead wire for power pick-up at 25 kV a.c. After transformation and rectification, this supply is used to charge 110V batteries. The scale of the filtering problem can be seen when noise peaks of 7 kV appear on this nominally 110V d.c. line. In spite of filters being used, the spike energy getting through has been enough to blow integrated circuits clean off the board, let alone corrupting the odd signal. Clearly in this environment 'fault-tolerant' hardware and software is not enough, and such destructive influences must be removed if anything
FIG. 4.1 TRAIN COMPUTER HAZARDS
like reliable operation is to be achieved. Filtering sufficient to remove or at least drastically reduce the probability of device destruction is deemed to be possible, and so the work carried out on this project assumes that power supply corruption is of the following form:

- Power loss varying from "glitches" of microseconds or less to total failure
- Voltage increases of maximum millisecond duration, but insufficient amplitude to destroy the device. Any larger spikes must be very infrequent with a mean time between occurrences in the order of hours or more.

The system will incorporate safety devices to ensure, say brake application, in the event of destruction of the computer system. However, short duration overvoltage conditions may find weaknesses in say a particular MPU chip which may ultimately fail, leaving the rest of the system intact to recover or at least degrade "gracefully". Corrupting influences on output and sense input lines are somewhat easier to deal with. For example if we take the wheel-mounted tachometers and mount them on different axles with the wiring for each spaced well apart, we are well on the way to ensuring that corruption of one does not necessarily affect the others. The signals involved are low current and voltage types, and so isolation from high voltages can be achieved using opto-coupling methods. In addition with three or more tachometers, we have redundant information yielding probable immunity from wheel-slip error and transducer failure. Routing the cables away from each other should reduce the likelihood of common-mode induced noise errors.
4.1.3 Mechanical problems

The big advantage of the static computer installation is that it is non-movable and thus not normally subjected to mechanical shocks and vibration. In the train borne environment the system is obviously subject to severe forces, requiring careful selection of cable connectors and board connectors. This type of mechanical design is not covered however in the material presented in this thesis. The temperature range encountered in the train carriage will be quite considerable and may require compensation, although this may be limited to simple electric fans. If conditions are severe, then the use of military specification devices may be called for, but the expense may be prohibitive.

4.1.4 Power supply requirements

In developing a mobile computer system one aims to reduce the current drain, yielding more compact and lighter power units, perhaps a critical factor in mobile operation. To this end MOS technology is used together with lower power Schottky TTL devices. Of course, for the train-borne system the existing battery packs will provide enough current because the computer requirements should be minute compared to motor and lighting loads. Unfortunately, the 'nominal' 110V train supply varies wildly according to load and charging conditions, ±25% fluctuation not being uncommon. This of course in addition to the high voltage spikes mentioned previously. The low voltage excursion occurs when the high voltage power pickup ceases as the train pantograph passes over a neutral or 'dead' section of the overhead wire. The computer power supply units must respond in one of two ways:
• Maintain normal uninterrupted supplies

• Send an early power supply fail signal to the CPUs, enabling them to shut down in an orderly manner.

While the first option is the most desirable, the second may need to be used in practice to maintain continuous operation. In this case special batteries will need to be incorporated in the computer units to maintain essential memory\(^{(2,3,4)}\). The assumption is made of course that the power supply unit itself is not likely to fail. However, as some components within it are passing high currents failure is a distinct possibility resulting in total system shutdown or even destruction, if suitable over-voltage protection is not provided. A way round this is to provide separate power supplies for each of the redundant computer units, thus reducing the likelihood of a single failure causing catastrophic system failure.

4.2 Processing Requirements - Inputs and Outputs

So far we have discussed the problems facing the designer of a train control computer system, in producing a unit which will survive the harsh on-board train environment. Having arrived at a point where the system is capable of running without actual physical damage for a reasonable time, we must consider the kind of processing it is required to undertake. In keeping with the British Rail Automatic Train Operation (BRATO) project discussed in Chapter 5, we will divide our account of the computer control system into three main topics:-
• Control of the motor and brake lines
• Supervision of the automatic "driver"
• Distance travelled, speed and acceleration calculation from tachometer data.

The second topic - supervision - assumes that the processor is made aware of line speed limits using at the present time, data from trackside transmitters.

4.2.1 Motor control

The four-car electric-multiple unit (EMU) we are considering, is driven by four axle-hung series d.c. motors developing a maximum of 207 hp each, situated one on each axle of the motor coach. The 25 kV a.c. from the overhead wire is fed via an air-blast circuit-breaker to the primary of the main transformer. The secondary winding tappings yield a range of voltages 0 to 1000V. These taps are connected via a rotary tap changer and bridge rectifier to the traction motors. (Fig. 4.2). Being series motors, they have very high torque characteristics at low speed but unless external regulation is used, excessive current is drawn at these low speeds. Electronic current sensing and applied voltage control by means of thyristors will be used extensively from now on, but at the moment nearly all the current stock employ electro-mechanical feedback control methods. The voltage tap-changer consists of a motor driven cam shaft controlled by the drivers' master control handle and by a notching relay in series with the traction motors. This notching relay is current sensing, and only allows the cam shaft to move to a higher voltage tap when the motor current has fallen below
FIG. 4.2 EMU MOTOR CIRCUIT SCHEMATIC
a preset level, i.e. when the train has accelerated to a sufficient speed. (Fig. 4.3).

The computer control system will need to be connected in place of the drivers' master controller, which supplies 'logic' 1 or 0 signals on four train lines. This controller or 'notching' handle has five discrete positions including OFF, which correspond to terminal speeds (6).

**Notch 1.** When selected from rest, a very low voltage tap is connected to the motors and thus this setting is only used for low-speed shunting.

**Notch 2.** When this control setting is used the train will accelerate up to approximately half speed with the notching relay cutting in and out until the tap changer has reached the half-voltage tap.

**Notch 3.** When selected, causes acceleration of the train up to a point just short of full speed, with maximum voltage applied to the motors.

**Notch 4.** A further increase in speed is achieved by applying a shunt resistor across the motor field winding, diverting current and thereby weakening the field.

Intermediate speeds are achieved by notching back from 2, 3 or 4 to 1 which will hold the cam shaft at that position. The train line 'logic' is shown in (Fig. 4.4). It can be seen that, providing the driving computer is fed with speed information, it can operate the train lines in the same way as the manual controller under human control. A signal is also available from the cam shaft, in the form of a pulse, each time there is a change in transformer tapping. It should now be possible for the driving computer to 'notch up' at the best rate consistent with passenger comfort and to suit operational requirements.
FIG. 4.3 NOTCHING RELAY OPERATION

FIG. 4.4 NOTCHING TRUTH TABLE
4.2.2 Brake control

The EMUs use a Westinghouse electro-pneumatic braking system operated by a hand controller on the drivers' desk. This handle has six positions and offers a variable braking rate from fully off at position 1 to fully on at position 2. Use between these two points constitutes normal operation, the other positions only being required if the electro-pneumatic (EP) system fails or for emergency application.

In order to design a suitable computer control system, it is necessary to understand how the basic EP brake works. (Fig. 4.5). In simple terms we have a drivers' pressure demand control handle which uses feedback from the brake cylinder to detect when the brake pressure corresponds to the handle setting. The movement of air in and out of the brake cylinder is controlled by two solenoid valves known as the application and hold valves. With the brakes fully released, with neither valve energised, there is no brake cylinder pressure (BCP) as the holding valve is open and the application closed. To apply the brakes, the controller is moved from position 1, causing the hold valve to close and the application to open. BCP thus builds up until the controller closes the application valve when the pressure matches the demand set by the handle position. Moving the handle back to position 1 opens the hold valve, reducing the BCP and releasing the brakes.

For computer operation of this system we need only two logic output signals, suitably buffered, to drive the solenoid valves. However, the computer will require, even if only as a confidence check, to know the BCP. This signal is of course "analogue" in nature, indicating pressure in the range 0 to 50 psi, and will need conversion via an A/D converter. In theory we should be able to dispense with this signal
FIG. 4.5 SIMPLIFIED EP BRAKE SCHEMATIC

<table>
<thead>
<tr>
<th>BRAKE</th>
<th>A</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLY</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HOLD</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>RELEASE</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

FIG. 4.6 BRAKE LOGIC TRUTH TABLE
altogether, as a measure of braking rate can be determined from the wheel tachometer data. The BCP data can be used in various ways however:—

• When running, as an indication to the driving computer that maximum braking effort has been achieved; this can be related to the drop in track speed indicated by the tachometers; the information may be necessary for smooth operation of a braking algorithm, bearing in mind that brake application is not instantaneous and BCP can take 5 seconds to reach maximum.

• When the train is stationary, the driving computer may, as part of its start-up routine, fully apply the brakes and monitor the BCP to check operation; a holding brake may be necessary when stopped at a station, and the computer will need an indication that this has been applied.

The basic truth table for the solenoid valve operation is shown in (Fig. 4.6). The simple two-solenoid EP brake is not the only one in use on BR, there being the more complex Westcode Controller for example. However, they still operate using 'binary' signals and computer interfacing should be no more difficult.

4.2.3 Speed and distance determination

A fundamental requirement of the automatic train driver is knowledge of actual distance travelled since reset, actual speed, acceleration and may be even the derivative of acceleration - "jerk".

Taking distance first, all sorts of imaginative measurement techniques spring to mind, ranging from photo-electric sleeper counting to satellite tracking. The best compromise between cost and accuracy appears to be
the wheel-mounted slotted disc optical tachometer. This device is bolted to the coach axle box and its drive shaft flexibly coupled to the end of the axle. As the shaft rotates, the disc with its radial slots rotates "chopping" a light beam between an LED and a phototransistor, and yielding a pulse train at the output whose frequency is proportional to the rotational speed. A counter driven by this signal will give a measure of distance travelled. In order to recover direction information, there are usually two concentric rings of slots on the disc arranged $90^\circ$ out of phase. Direction can be determined from the phase relationship of the two output waveforms.

The two signals from the tachometer may be fed to the computer in one of two ways:–

- Direct into an input port of the microprocessor system, with an interrupt to the MPU for every pulse.
- Via a hardware binary counter which, in effect, will partially process the tacho data. The counter will be read and then cleared at fixed intervals by the processor.

The problem of which method to choose depends largely on the number of slots on the disc and on the processing speed of the microprocessor system. Linked to the latter are considerations of how acceleration figures will be produced – by calculation or by further hardware in the form of accelerometers. Initial tests at BR on their Gemini test train with a 12 pole tachometer yielding a pulse for every 228 mm of distance travelled, indicated that more resolution was required\(^7\). Hence further tests have used 250 and 1000 pole tachometers, the latter giving a resolution of less than 3 mm. If it is decided to drive the input port directly and allow the MPU to count the pulses, then with
a 1000 pole tachometer and a speed of 60 mph, the processor will be interrupted by a tacho pulse every 110 μS. This may be acceptable if processing is limited, but the 2-4 μS instruction time of a typical 8 bit microprocessor may preclude anything elaborate as discussed in the section on program requirements. Using an external hardware counter read say every 100 mS eases this problem but introduces an extra piece of hardware to fail. Continuing with the example, the counter would need to be at least 10 bits long in this case. Additional hardware will also be needed to generate the timing interrupt at 100 mS intervals.

4.2.4 Speed supervision

A third main computer function is required besides tacho data processing and control of braking/notching. The train control computer needs to be told timetable requirements and the local speed limits for the section of the track over which the train is travelling. Obviously, an input from the tachometer is required to determine current position and speed data, and we are left with the problem of finding a source of route data. There are several possibilities:-

- Route data contained in pre-programmed Read-Only Memory inserted into the machine on the establishment of a new route. This has the disadvantage of requiring boxes of memories to be kept for every occasion. In addition, no account can be taken of temporary speed restrictions due to engineering works, etc.
- Route data transmitted to each train from a central control station by radio link, and updated as necessary. The problem with this is that it requires the setting up of a fairly elaborate country-wide communications network to feed the central computer. As all trains
become fitted, the task for this master 'brain' may become too involved, but with recent advances in computer technology the idea is not far-fetched. Of course a serious breakdown at central control would cause all trains to revert to manual operation, by far the most serious drawback.

- Route data transmitted by radio link to the train from trackside equipment located at intervals along the track. With this method only local information on conditions immediately ahead need to be stored in the trackside memory. Most of the data will be permanent in nature, such as distance to next signal, gradients, normal speed limits, etc. Changing data such as signal aspects, temporary speed restrictions can be held in Read/Write Memory (RAM) and changed by the local signalling system or by manual trigger when required.

The latter operation seems to be the most practical at present, and both this and the previous alternative will require the inclusion of a radio receiver in the train control equipment to be carried. The output from the receiver to the control computer can drive either a parallel or serial input port, assuming that the data is now in binary form.

4.3 Processing Requirements - Program Software

Having decided upon the kind of input data that will need to be processed, together with the main outputs required, some thought must be given to the operating program (or programs). From this we may well find that it is preferable to split the overall task into smaller sections processed by separate computers. Safety supervision implies
independence of action from the automatic driving section. Loss of autodrive due to processor malfunction will probably mean loss of supervision as well with possibly catastrophic results. This requirement for independence gives us our first split in the hardware and software. With a separate processor with its own program for the supervisor or safety check system, a large software burden is removed from the driving computer. In addition, further outputs from the system can be added to enable the supervisor to apply the brakes independently of the driving unit, if it decides that the latter is exceeding safety limits.

Having taken away the speed supervision work from the driving computer for mainly safety reasons, we can perhaps also relieve the latter unit of the tachometer processing task. This may not be necessary depending on how much processing needs to be done in how short a time interval. The options are:-

- Tachometer input direct to processor, no external hardware accelerometer, all processing done in software. This is the minimum hardware option requiring a very fast microprocessor to do all the multiplication necessary for acceleration calculation in between tacho pulses. It has been suggested that the tacho pulses will be coming in at less than 100 μS intervals during high speed running, so with software multiplication taking of the order of milliseconds, there may be problems.

- Tachometer input direct to processor, but acceleration read directly using say bubble-type accelerometer. In addition, multiplication could be done by hardware multiplier circuits, multiplier chips yielding 16 x 16 bit multiplies in less than 200 nS are available. This solution involves a lot of extra hardware and tests conducted
at Derby have shown that the output from the accelerometer may be tricky to interpret.\(^{(7)}\)

- Tachometer pulses counted on external counter, which is read and cleared at widely (relatively) spaced intervals, say 100 mS. All calculation is done in software. The 100 mS timing or "sample" points should allow plenty of time for the necessary calculation if there are no other tasks. In other words this method will probably involve a dedicated processor separate from the driving unit.

Taking into account the relative cheapness of 8 bit (and now 16 bit) microprocessors, the third option seems to be the most attractive, particularly as each section of software can be developed almost independently. The requirements for each sub-system can now be summarised.

4.3.1 Automatic driver computer

This processor and program will in its simplest form, have outputs linked to the existing train lines and E.P. brake valves, and take distance, speed and acceleration data as inputs from the tachometer unit. It will also receive maximum safe speed or "suggested" speed from the supervisor unit. The software program will convert these inputs into appropriate settings for brake and notch.

4.3.2 Tachometer computer

This processor will sample a tachometer pulse counter at regular, exact intervals and will calculate distance travelled, speed, acceleration and 'jerk', using a four-term algorithm.\(^{(8)}\) It will also deduce direction from the phase relationship of the two tacho pulse signals. The
computed information will be output both to the automatic driver and the safety supervisor unit when requested or at regular intervals.

4.3.3 Safety supervisor computer

This processor acts as a "monitor", taking inputs on speed of train, etc., from the tachometer computer and comparing them with inputs from the trackside data stores via a radio link. Outputs go to the driving computer giving it safe limits for speed. This computer will also have independent control of an emergency brake if it considers the driver to be acting irresponsibly.

4.4 Reliability Considerations

The need for reliability in a totally automatic passenger vehicle is self-evident, as a few train crashes with or without fatalities will soon wipe out public confidence in electronic systems. However, although reliability and safety are closely linked they are not to be considered synonymous. The chief consideration must be safety, hence the dominance of the term "fail-safe" in railway signalling systems for the last 100 years, in theory if not in fact. If we accept that a system can fail and is thus "unreliable", then every effort must be made to ensure that it fails leaving its charge, in this case a train, in as safe a condition as possible. For this reason, the simple train control system described so far (Fig. 4.7) is not adequate to ensure this safety, assuming each of three functions is run by a single microprocessor and associated components. If any of the systems, with the possible exception of the driving computer, develops a fault it may well go undetected. The driving computer
FIG. 4.7 BASIC TRAIN CONTROL SYSTEM
is effectively checked by the supervisor. Hence one form or another of redundancy as discussed in Chapters 2 and 3 must be introduced. The details of two approaches to the problems of safety and reliability will be covered in the next two chapters on the BRATO and CYCLONE projects. For now, however, let us look at what we expect from our train control system in terms of its ability to "get on with the job" with a minimum of manual driver or service engineer intervention.

4.4.1 Self-testing

From the definitions of reliability in Chapter 2, we see that the term self-testing means nothing more than an ability to confirm that a fault exists. The software required to do this is minimal if shut-down on failure is required. This approach is not only somewhat drastic if the fault is only transient, but is of no help to the service engineer in locating the problem. Normally, there should be adequate program space for a more thorough check out.

4.4.2 Self-checking

This approach is more useful than just self-test because not only may the fault be identified and located, some limited corrective action may be taken to correct the resulting error if the fault is transient. Now we see the difference between self-test and self-check - the former may be used as a power-up hardware test allowing nothing to run if any faults are detected, the latter as a periodic processing check during operation. The self-checking approach is tolerant of brief data glitches, usually demanding a re-try if an error occurs, or recovering true data with special error-correcting codes. It is anticipated that the bulk of processing errors in the train
computer(s) will be caused by these passing effects occurring at random and infrequent intervals, and provision must be made to detect and then recover from them.

4.4.3 Fault tolerant

Moving up the sophistication ladder we arrive at a point where the computer can diagnose its own hardware faults and either abandon part of its function, preferably a non-critical one ("degrade gracefully") or switch in spare replacement units. A third option is of course to switch off the faulty unit altogether as outlined in Chapter 2 and continue running on existing "redundant" units. Selection of one of these alternatives will need careful study of safety, availability and cost. If we have an aim of no single failure causing system degradation or safety to be compromised, then a lot of redundant hardware and/or software must be incorporated into the design. A further failure may then cause some non-critical or least-critical function to be lost. That in itself may involve a considerable degree of automatic job "shuffling" if safe running is to resume. It must be remembered that the computer(s) must still function as a self-checking unit so that still further faults may be detected and dealt with accordingly. In the case of our simple train control system model we could lose the automatic driver first and return to manual control leaving the supervisor and tachometer processors to keep the new human operator supplied with data. Loss of speed supervision or tacho data instead could not be tolerated and would lead, indeed must lead, to a return to manual operation. However, if the supervisor no longer functions there is no particular reason why the tachometer should not still work - providing it is still self-checking. A
FIG. 4.8 SYSTEM DEGRADATION CHART
"degradation chart" is thus built up as in (Fig 4.8). The need for the system to be tolerant of at least one hardware failure without loss of function increases the amount of redundant hardware required and may be sacrificed if system reliability in practice suggests that acceptable intervals of fault-free running can be attained.

4.5 Conclusion

This chapter has taken the reader from the idea of a fixed potentially unreliable, and in vehicle terms unsafe, computer control system, through a discussion of some of the problems of mobile operation to a rather skeletal design. At this point, the detail design can go off in a variety of different directions depending on personal preference and likely cost. The review of multiprocessing techniques has illustrated in simple brief form, some of the past and current designs for reliable machines. The following chapters illustrate in some detail two particular proposals each of which has its own merits. Unfortunately, due to the nature of the subject matter design validity from a safety viewpoint can really only be determined ultimately by actual experience.
REFERENCES FOR CHAPTER 4


5.1 Introduction

The British Rail Automatic Train Operation project (BRATO) developed out of earlier attempts at automatic control\(^{(1)}\). Most of these designs were very limited in "intelligence" because of restrictions imposed by the available component technology. The microprocessor has allowed more robust and complex control systems to be developed, but the old requirement of safety remains. Present day signalling equipment is considered to be 'fail-safe', and new designs must either meet or improve upon currently accepted standards\(^{(2)}\). The failure modes of integrated circuits are many and varied\(^{(3)}\) and adequate protection against the disastrous consequences of some of them involves some kind of circuit redundancy. Norton\(^{(4)}\) suggests that the widespread acceptance of redundancy as a means of achieving safety, as well as reliability, will only come about with modification of the safety rules. Indeed the goal of fail-safe operation may well be illusory, as experience has shown that wrong-side failures have occurred in spite of all the precautions. New safety rules will need to include an acceptance of a small probability of catastrophic failure, and the BRATO hardware was built with this in mind. This chapter will summarise the work done at BR, Derby on translating the basic train control system block diagram shown in Chapter 4, into a working design. The special safety features incorporated to prevent damaged processors getting back on line will also be examined.
5.2 The BRATO Subsystems

The physical divisions of the hardware follow those in the diagram of Chapter 4 quite closely (Fig. 5.1), with the functions of autodriver, supervisor and tachometer clearly defined as separate computers. Attention is drawn to this close match between the system function and the hardware realisation because of the differing approach taken in the Cyclone computer development.

5.2.1 The autodriver

The autodriver is a self-contained computer based on the DEC LSI-ll processor with no hardware redundancy features incorporated. The outputs of the device control the train traction and braking systems in response to four types of data (5):-

- Safety data: Gradients, line speeds, signal aspects.
- Performance data: Target speeds, etc., to satisfy some intention such as minimising energy consumption.
- Train controls and sensors: position, speed, acceleration, BCP, motor current, camshaft position and state of driver's control panel.
- Real time: A timekeeping clock is included in the hardware.

The safety and performance data are input from the receiver unit in the form of 256 bit BCH-encoded telegrams transmitted serially. When either autodriving or driving manually, this data will be used to provide a series of monitor functions derived by the software: -

- A data handler to decode the incoming serial data and format the other control inputs.
FIG. 5.1 BRATO SYSTEM SCHEMATIC

Gloster-Disc Tachometer Heads
Mounted on 3 Different Non-Powered Axles.
• A traction monitor to watch the operation of the train under traction and deal with wheel-slip.
• A braking monitor to examine brake operation and watch for wheel-slide.
• A timekeeper to calculate the required speed necessary to bring a late-running train back to time.

When autodriving, a series of control functions are performed according to whether the train is before or after the point at which braking to a target should commence. If before, then the train may be driven in one of three ways:

• With safety data only available, the autodriver will follow the minimum time profile.
• With safety and performance data available and the train running to time, the performance data will be followed to satisfy some precondition such as maximum energy conservation.
• If both safety and performance are available, but the train is running late, the speed determined by the time keeper will be used.

At the braking point and beyond a series of braking modes are used: Initial brake, constant deceleration, brake release and a holding brake when at rest. The autodriver relies entirely on software self-checking procedures as it contains no replicated hardware. If it decides that input data is incorrect, then the manual driver will be warned and required to take over. Similarly, incorrect calculation results should be detected by software and lead to shutdown. If in spite of all this self-checking, a fault leads to the train speed going over the maximum permissible value, then the safety supervisor computer will shut the autodriver down. The use of a non-redundant processor was considered acceptable because of the
presence of the supervision system. The decision to use the LSI-11 processor rather than a 6800 microprocessor system as in both supervisor and tachometer, was based on experience gained from early TACT autodriving trials. This indicated that the software would have to be rather more complex and run faster than at first thought, and was probably beyond the capability of the 4-bit 4004 microprocessors in use at the time. The power of the 16-bit machine was thought to outweigh the disadvantages of using two different processors in the same system. The autodriver receives data from the tachometer processor on two asynchronous serial lines at a rate of 9600 baud. Telegram data is input from the receiver via an 1800 baud synchronous serial link.

5.2.2 The supervisor

The safety supervisor computer is built around three 6800 microprocessor modules arranged to operate in standard TMR fashion. Hence a single failure will not degrade system function, but a further fault will lead to a shutdown of supervision and hence automatic driving. In theory two good processor modules can 'mask' a faulty third unit by voting on the outputs of all three. Thus the bad processor need not be shut down to the extent of having its power supply cut. However, as a safety precaution two mechanisms exist to blow a power supply fuse thus ensuring no possibility of incorrect outputs reaching the train controls; two processors attempt to "assassinate" the third in the event of a difference and the unit at odds with the other two attempts "suicide". Some of the precise hardware details are discussed later. In order to avoid simultaneous and identical corruption of all three processors, each has its own clock and is run out of synchronism with the others.
Program divergence is prevented by interconnections between the processors - 'anti-divergence links' - and thus unnecessary shutdown due to differing outputs avoided.

Each supervisor processor is fed serial telegram data from the receiver unit along independent links. The supervisor and autodriver both receive and decode the same BCH encoded data which contains the safety and performance data supplied by trackside transmitters.

5.2.3 The tachometer processor

The tachometer processor unit is built along the same lines as the supervisor, namely standard TMR. The general principle of operation is the same as well, except that this unit receives and processes pulses from three independent slotted disc tachometer heads. Each head is mounted on a different axle to allow for wheel-slip detection and the use of three heads allows for failure without system loss. The processor program converts the two-phase pulse data into distance, speed and acceleration information transmitted asynchronously along serial links to the supervisor and autodriver. The pulse data is partially processed by a front-end circuit consisting of a Direction-of-Motion sensor followed by a 10-bit up-down counter. The counter clocks up in one train direction and down in the other with the output being read by the microprocessor every 100 ms. Distance travelled is measured by accumulating counts from an origin defined by an End-of-Section (EOS) pulse. The latter is generated by the supervisor wherever the train passes over a track beacon at one end of the track data transmission loop. The output stages yield two serial redundant lines to both the autodriver and supervisor, one from each of two tachometer processors. The output from the third is not used except for checking purposes. In order to achieve
the desired accuracy, account is taken of wheel-slip, wheel wear and wheel
expansion due to brake heating. This takes the form of a calibration
factor which uses prestored maintenance data as well as dynamic correction
based on actual section length data compared with recorded data. It is
planned that the calibration factor should converge to an accurate
value with a very few section lengths of a journey. It is felt
that the combination of triplicated tachometer heads, triplicated pro­
cessors and duplicated serial data lines will give the required degree
of availability.

5.2.4 The receiver

An on-board receiver picks up a carrier signal transmitted from
loops in the "four-foot", which is modulated by a continuous,
synchronous serial data stream of alternating 256 bit telegrams of
safety and performance data. Each BCH encoded telegram consists of
211 data bits, 1 modulo 2 parity bit, 16 bits of synchronising pattern,
16 BCH parity bits and 12 spare bits. Such a code allows the detection
of five single-bit errors per telegram and burst errors of up to 16 bits.
The transmission rate of 1800 baud will allow a high mutilation rate of
telegrams even when running the train at high speed because there will
be time for several copies to be received. The designers feel that this
level of redundancy should result in reliable data transfer.

5.3 The System Monitors

The basic system monitor circuit is shown in Fig. 5.2. The power
supply to each processor and its output interface is controlled by one
such circuit. In this case for processor A of the TMR group, the supply
FIG. 5.2 SYSTEM MONITOR FOR TMR CIRCUIT

FIG. 5.3 OUTPUT DRIVE CIRCUIT
fuse may be blown by processors B and C both attempting to assassinate A by turning on transistors T1 and T2. The offending unit A may attempt suicide by turning on T3. As an additional precaution, unit A from time to time passes current pulses through the fuse by turning on T1, T2 and T3 briefly itself. The pulses are not sufficient to blow the fuse but successful operation can be determined by sensing the voltage drop across R1. A typical output drive circuit is shown in Fig. 5.3, which allows either or both of two processors to drive an output line. The power supply is derived from the system monitor.

5.4 Conclusion

The basic proposal for the BRATO train computer system has been described for purposes of comparison with CYCLONE in the next chapter. The design is seen as a compromise between the need for a highly available system with the train not spending most of its time in the repair shops, and the safety requirement. As a result much effort has been spent making sure that fuses do not blow unnecessarily with program divergence or brief transient effects. Unfortunately, the BRATO project was abandoned by BR just before pilot trials were due to start and so there are few concrete results available. However, the TMR processing systems with their fuse-blowing circuits live on in the solid-state signalling program which still continues.
REFERENCES FOR CHAPTER 5


CHAPTER 6

THE DEVELOPMENT OF CYCLONE,
A COMPACT, SELF-CHECKING MULTIPROCESSOR

6.1 Introduction

This chapter describes the various stages in the development of a prototype multiprocessor module - CYCLONE I - designed with automatic train control as a possible application in mind. The hardware produced, however, has many features that would not be required on a 'production' system. These include the front panel LED matrix display, VDU, teletype and magnetic tape cartridge drivers. In addition, a great deal of utility software was written in order to simplify testing and program development. The machine has been used to study the feasibility of the particular processor configuration in relation to processing throughput, bus contention and response to transient faults. Although the basic system operation concept remained unchanged from its inception, the implementation has seen much change in the light of experience. The prototype module was not built to a mechanically reliable specification as it was considered unlikely that it would move outside the laboratory.

6.2 Design Objectives

The original stated aim of the work described in this thesis was to investigate alternatives to the path taken by the BRATO design team. The latter had designed a system with an approximate hardware correspondence
to that of the functional block diagram described in Chapter 4 (Fig. 4.7).
The main aim of the work presented here was therefore to design a better
system if possible, and the opportunity was taken to experiment with a
radically different approach. If the hardware arrangement of the BRATO
system were to be used as a basis, it was felt that little could
be achieved in the way of original work as the only scope lay with using
different microprocessors and tinkering with the bus structure. It
was therefore decided:-

- To cut down the number of processor-memory subsystems to three.
- To experiment with time-shared processor operation in order to
  realise the transient fault immunity requirement.
- To develop a software checking system to be as near as possible
  "transparent" to the applications programmer.

A major objective set by BR cannot be met by the basic three processor
modules, and that is the requirement that there should be no loss of
system function with a single hard fault. Two approaches to this problem
were considered, and are discussed in detail later. The fail-safe
requirement must be met as far as practicable and so mechanisms to detect
catastrophic system failure had to be included. These need to be kept
as simple as possible, but with near 100% coverage. In addition, the real-
time operation constraint means that such a fault has to be detected
within milliseconds and corrective action taken and be effective,
within a similar time interval. Although the weight restriction for
train borne operation is not serious and relatively speaking, there is
plenty of room for control equipment, it was felt that the emergence of
a compact self-checking multiprocessor could lead to other applications
for Cyclone.
6.3 The CYCLONE Concept

6.3.1 Time-redundant multiprocessing

The principle of the proposed system allows n microprocessor-memory modules to execute n programs simultaneously, which in hardware terms appears to represent non-redundant multiprocessing. If, however, we arrange, through suitable bus control logic, that any processor can run any program then time redundancy techniques can be employed to realise the same fault-location capability as a TMR system requiring 3n processor-memory units. A program or section of program is run n times in succession and the results at the end checked for identity. Hence, a transient fault disrupting one stage can be detected if \( n = 3 \). This may yield an answer to transient problems but if the same processor is used each time, then identical sets of incorrect answers may be passed as correct by a unit with a hard fault. The short solution is to use a different microprocessor each time the same program or program section is run. From here on, it will be assumed that program operation is broken up into blocks or sections each requiring a complete "system cycle" to execute and check (Fig. 6.1). Disregarding system management time overheads, the system cycle is divided into n application program "sectors" and one error checking and recovery sector. At each sector boundary, three tasks are performed by the system management routines in software:-

- Processor checkwords and results are dumped into memory.
- The processors swap program tasks.
- Each processor reloads its registers with the program status of its new task and resumes operation.
FIG. 6.1 TIME REDUNDANCY FOR N PROCESSORS
There is therefore an automatic program rollback at each sector boundary although not in response to an error\textsuperscript{(1)} and not using the same processor. The time penalty incurred by the Cyclone system becomes evident here as every program is repeated at least twice. In addition, the checking software, depending on complexity, uses further time in each cycle, resulting in each program taking more than \( n \) times as long to run as the non-time redundant system equivalent. If an error in the preceding sectors is detected by the checking sector software then the cycle is repeated using the previous starting data. This constitutes the more traditional error triggered rollback and retry part of the system\textsuperscript{(1,2)}. The checking sector is given the job of communicating with the outside world with no direct input/output commands or data being sent from the other sectors. This ensures that I/O movements are not repeated \( n-1 \) times, and only allows I/O if no errors have been found in the preceding sectors.

6.3.2 The choice of bus structure

The selection of a bus structure to meet the requirements of reliability and throughput for Cyclone involved much thought, as the number of variables is considerable. Bowen and Buhr give a very good summary of the basic types in their book on multiprocessing\textsuperscript{(3)}. In an attempt to restrict the choice, it was decided to go for a minimum hardware approach on the straightforward basis that there would be less to fail. This particular criterion is often the best in the maze of available complex structures, but usually asserts itself when absolute failure rate figures are added up for an individual design. The various fully-interconnected and loop types were rejected because it was estimated that the overhead in switch hardware to meet the variable processor-memory assignment would
be to great. Hence the global bus with central switch configuration was chosen (Fig. 6.2). The central global bus introduces a hardcore reliability problem, but it was considered that its contribution to overall system unreliability was likely to be negligible when compared with that of the LSI components. The shared bus does in fact make a positive contribution towards minimising the effects of transient faults. This is because only one processor will be driving the bus at any instant, and so any fleeting data corruptions occurring upon it will only damage one sector of one application program. Given that transient and intermittent faults are very difficult to locate\(^{(4)}\), the technique of reducing their likely effect is seen as the best one to implement. Two main assumptions are made therefore:

- Errors due to the transients or intermittents are always detected, before input/output operations.
- Sufficient processing time is available to permit retry.

The second assumption is endangered by the use of a common bus system, with a processing "bottleneck" leading to slower operation. Whether or not this is a problem depends on a variety of factors such as the number of processors involved and the particular microprocessors' bus requirements. The common bus system introduces extra hardware in the form of arbitration logic to sort out simultaneous requests. The bus arbiter has to be implemented in the simplest way, but in the case of Cyclone is made more complex because of the lack of fixed processor-memory links. Several practical realisations of multi-microprocessor bus structures are discussed by Rozsa\(^{(5)}\) and Moss\(^{(6)}\), with a more general analysis of arbitration methods being carried out by Bowen\(^{(3)}\) and Thurber\(^{(7)}\). The latter divide the techniques into two broad classes - Centralised and Decentralised with subdivisions of Daisy Chain, Polling and Independent Requests. It was decided to use centralised control, for although another potential
FIG. 6.2  GLOBAL BUS WITH CENTRAL SWITCH

FIG. 6.3  INDEPENDENT BUS REQUEST SYSTEM
hardcore problem is introduced, the experimental flexibility offered by a single arbiter together with the resulting simplification of the processor boards outweighs the disadvantage. A priority allocation system had to be devised, but not one that assigned fixed priorities to particular processors. Consider a section of one program run \( n \) times by \( n \) different processors in \( n \) successive sectors. It follows that different bus request priorities will lead to sector 0 seeing more processing done than sector \( n \), assuming descending priority allocation. The error checking software relies on identical sets of results for each sector, and so the priorities must also be changed at each sector boundary. This leads to the establishment of program, as opposed to processor, priority operation. This requirement renders the daisy-chaining system unusable. The centralised polling method was considered, but in order to keep request-to-grant delays for the use of a vacant bus, as close to zero as possible, independent request operation was finally selected. The basic form of this for three processors is shown in Fig. 6.3. In this configuration, each CPU has a bus request output and bus grant input connected to the arbiter. There is also a wire-ORed bus busy line from each unit. When given control of the bus, a unit raises the busy line until it has finished. The Cyclone logic is slightly different in that the arbiter or bus control logic generates independent busy signals to the other units if there is a bus contention and at no other time (Fig. 6.4). The first system fails if a processor fails to remove its 'busy' signal, the second if a request line signal becomes permanently set. The specific hardware used to produce the latter signal has been designed to counter this threat on Cyclone. The only delays inherent in the Cyclone bus control system are gate propagation delays, which as they do not total up to a clock phase interval, do not have any effect on system speed.
FIG. 6.4 CYCLONE BUS CONTROL LOGIC
A processor unable to obtain bus access is held up only for as long as it takes the unit with access to get off.

6.3.3 Optimising for throughput

If the three functions for train control, of autodriver, supervisor and tachometer were to be realised as individual programs, then as a suitable starting point, it was assumed that the common bus would need to service three processors at least. Deriving exact figures for bus usage given a particular configuration is all but impossible as it not only depends on the individual programs, but also on their interaction with each other. A traditional yardstick is the Bus Utilisation Factor (BUF) defined as:-

- The Bus Utilisation Factor is the ratio of the number of bus access cycles to the total number of cycles to fetch and execute a program instruction.

In addition, we shall refer to System Throughput:-

- System Throughput is the total number of instructions executed by all processors in one second.

It is fairly obvious that one cannot keep increasing overall throughput by simply adding more and more processors to the bus. Eventually bus saturation occurs and depending on the bus allocation system in use, some units will see hardly any access. The oft-quoted rule-of-thumb for determining the optimum number is illustrated in Fig. 6.5. This says that if the BUF = 1/m for a particular processor, then m processors may share the same bus yielding a factor m improvement in system throughput. Any further increase in the number of processors will yield no further gains. This approach is true if all processor instructions are the same,
with bus access cycles distributed within the instruction such that after an initial sorting out transient, processors gain access in sequence with no interference. Before moving on, some symbols will be defined:-

\[ T = \text{System Throughput (Instructions/sec)} \]
\[ T_1 = \text{Single Processor Throughput} \]
\[ T_S = \text{System Throughput (No interference)} \]
\[ T_I = \text{System Throughput (With interference)} \]
\[ N = \text{Number of Processors} \]
\[ X = \text{Number of bus access cycles/instruction} \]
\[ Y = \text{Total number of cycles/instruction} \]
\[ U = \text{Bus Utilisation Factor} \]
\[ C = \text{Clock Rate} \]

For a single processor: \[ T_1 = \frac{C}{Y} \]

For \( N \) processors: \[ T_S = \frac{NC}{Y} \]

This last formula gives us the straight diagonal line of Fig. 6.5 to the point where \( N = \frac{1}{U} \). For a real microprocessor, \( U \) is seldom a convenient figure, and varies from instruction to instruction. Hence even on a three processor system, for example, with an average \( U \) of 0.33, interference still occurs and the improvement in throughput of 3 is not achieved. An estimate of the worst-case number of cycles per instructions lost as a result of bus contention can be calculated. This assumes that if \( N \) processors normally need \( X \) bus access cycles per instruction, then there will be \( X(N-1) \) wasted cycles per instruction per processor. Hence the new throughput is given by:-

\[ T_I = \frac{NC}{Y + X(N-1)} \]
FIG. 6.5 BEST-CASE EFFECT OF BUF ON THROUGHPUT

FIG. 6.6 EFFECT OF INTERFERENCE ON THROUGHPUT
\[ NC = \frac{y(1 + u(N-1))}{y} \quad \text{as} \quad U = \frac{x}{y} \]

A useful number to have is the ratio of this figure with that of a single processor system.

\[ \frac{\tau_1}{\tau} = \frac{\frac{N}{1 + U(N-1)}}{x} \]

Using this formula, a family of curves for different values of \( U \) can be drawn (Fig. 6.6). Interference effects can be deduced, and we see a considerable loss in throughput takes place unless \( U \) is very small. This suggests that microprocessors with an efficient internal architecture, requiring few extra clock cycles beyond those for bus access, are not suitable for common-bus multiprocessor operation. Some common 8-bit MPUs such as the 6800 and 8080 have high values for \( U \) and thus their use in multiprocessor applications is limited to "private-bus" designs. It was decided therefore to use three microprocessors for Cyclone, each with as low a \( U \) factor as possible. This implies using a device with a low efficiency, but of course the object of the exercise is to spread the computing power for reliability and safety reasons, even though the same throughput might be achieved with a single fast MPU.

6.3.4 Task switching at sector boundaries

Arranging for a processor to stop running one program and start another is relatively straightforward, but getting two or more MPUs to exchange programs with no loss of data requires a rather more involved hardware and program structure. The basic requirements were as follows:-
Any bus switchover must occur in an interval of no bus activity to prevent possible data corruption.

All processor status must be preserved across the boundary to allow accurate repetition.

Hardware and software synchronising will be required to ensure that all sectors are of the same length.

Three main methods of effecting the switchover were considered, each requiring both hardware and software to implement, in varying relative proportions:

1. Changeover signalled from within the program. This method requires the application programmer to insert breakpoints in the form of software interrupts, or pulse output flag instructions. Hardware is therefore triggered to switch the route logic. Software routines control restart.

2. Changeover signalled by external hardware in the form of pulse to a sense input pin on the MPU chip. The application program will need to examine the state of the sense input frequently. As before, software controls restart.

3. Changeover is initiated by hardware interrupt request. No special instructions are required in the main program, but restart is handled in software by the interrupt routine.

The first and second methods were rejected because of the complications to the main programs, the former requiring careful program structuring to set the time intervals, and the latter wasting time reading the sense input. It was decided therefore to use hardware interrupts, derived from a divider chain off the system clock. This divider would be programmable...
to allow experimentation with different sector lengths. Although the 
interrupt can signal each processor that a sector boundary has arrived, 
it cannot itself trigger the bus control logic because of the need for 
all bus activity to cease first. The signal to do this must come from 
each processor when it has completed it's register dump and other 'house-
keeping' activities. The MPU must enter a programmed halt state 
suspending operation, indicate this fact, and then restart in response 
to a signal from the control logic. At this stage suitable hardware 
had to be designed, and a microprocessor selected. The choice of the 
latter would determine how much extra hardware would be required, 
to the extent of interrupt priority logic, clock stopping logic, etc.

6.4 Hardware Development

6.4.1 The choice of microprocessor

The BRATO team had decided that a 16-bit machine would be necessary 
to run the autodrive functions, and so a 16-bit microprocessor was 
to be selected. Unfortunately the choice at that time (1978) was rather 
limited, there being only two easily obtainable single chip types: The 
National Semiconductor PACE and the Texas Instruments TMS9900. The 
9900 architecture was not thought to suit the bus ideas suggested so far. 
The main drawback was that the device keeps all its registers in separate 
RAM. Hence the bus could get clogged with inter-register data transfers 
that do not need bus access on other devices, such as the PACE. In 
fact the more processing that is carried out without recourse to external 
data storage, the better. This yields the need for a comprehensive 
internal register set with matching instructions, and perhaps a small
on-chip push-down stack. The PACE architecture meets both these requirements, and in addition contains interrupt priority decoders, programmed halt instruction with corresponding output signal, and a variety of sense input, and flag output pins. All the instructions are single length, which not only makes programming easier, but aids error recovery as there is no data mixed in with the instruction words. A 2 MHz clock frequency gives an average instruction time of 10.5 μS, or 21 cycles with 3.7 being the average number of bus access cycles per instruction. This gives an average $U = 0.176$. In fact, a typical program making extensive use of the internal register and stack will use instructions with an average time of 8 μS, and with a value of $U < 0.1$, better throughput may be obtained than expected from the pessimistic calculations using average values. The register-in-memory structure of the 9900 does however reduce the processor overhead on task switching, as register dumps are not necessary[8]. However, the flexibility offered by the PACE machine was thought to be more important when considered with the features already discussed. In the early stages of development, cost was of major importance, particularly as at least three microprocessors would be required. Hence the then-new low-cost NMOS version of PACE was selected for use in the Cyclone design. This device is also known as the INS8900[9,10].

6.4.2 Synchronising problems and solutions

In keeping with the minimum hardware philosophy, system timing for a three-processor Cyclone module is derived from a common master clock oscillator. This also makes the synchronising of processors easy, although to achieve identical throughput for consecutive program sectors
with different MPUs, special measures were needed. The first proposal was as follows:-

- The hardware timer generates an interrupt to signal the sector boundary. The interrupt handler routine then dumps check results to memory.
- A programmed halt instruction is executed by all processors, and when this state has been entered each MPU produces a 'Halt' signal. A logical AND of all three signals tells the bus control hardware that all bus activity has been suspended, that it is safe to carry out task switching, and to reset the timer.
- As the bus switching takes place, a restart signal is sent to all three MPUs, to take them out of their programmed halt state.
- Each reloads all its registers from RAM with the new task data and then executes a Return from Interrupt instruction.

In theory all sector lengths should be the same within one instruction, but it was found that many error cycles were being produced caused by unequal sectors. The automatic error retry would usually be successful on the second or third attempt, but it was felt that the situation was far from acceptable. It became clear that synchronisation at sector boundaries would have to be to within one clock cycle for guaranteed error-free normal operation. The halt signal pin on the 8900 is normally high but when activated, produces a 7/8 duty cycle pulse train. All three MPUs must be synchronised to the same pulse edge and this is achieved by waiting until all have entered the halt state, and then "freezing" the clock signal to each as the appropriate halt signal edge appears. Restart is triggered when the three processors are in alignment. This operation is potentially hazardous in that the
8900 is a dynamic processor, i.e. the clock signal must not stop for more than a few microseconds, otherwise internal status is lost. The logic is designed to ensure that the time limit is not exceeded, and no data loss traceable to this cause has been observed to date. It was also decided that the timer for the interrupts would not be restarted until all boundary changeovers had finished. Hence a second program halt instruction is inserted into the interrupt handler, just prior to the return. The two-stage operation can now be summarised as follows:—

- The interrupt timer generates an interrupt and causes check word dump
- The first halt instruction causes switchover
- Status re-load takes place
- The second halt re-synchronises the processor and restarts the timer prior to execution of the return from interrupt.

The precise logic to perform these functions will be described in Chapter 7.

6.4.3 Loss-of-processor tolerance

As a consequence of little hardware redundancy being incorporated in the basic Cyclone module, it was accepted from the start that single hard failures would cause system shutdown. However later investigation showed that by optimising the existing timing logic and with the addition of a little extra SSI, some classes of single failure need not cause total loss of operation. In particular, the MPU boards were singled out to see if total processor failure could be tolerated. It was found that providing the offending MPU signal could be isolated, one program would
have to be sacrificed to enable the other two to run normally. Isolation could be done in one of two ways: Switch off the MPU board power, or turn off the tri-state control signals to all the MPU data bus buffers and input/output control signals. For the purposes of prototype development, the second approach was taken as the easiest to implement. An as yet undefined fail signal from each MPU board reports processor loss to the timing logic which switches immediately to two-out-of-three running. A further processor loss causes total system shutdown. The interrupt software had to be extensively modified to cope with this new situation, bearing in mind the existence of "blank sectors" within the remaining two programs. The loss of throughput with one MPU and program going down is partially compensated for by the reduction in bus contention and hence increased speed of the other two programs. In practice, loss of program will mean loss of function, and a decision would have to be made whether continued system operation is possible given the loss of a critical function. Nevertheless this "graceful degradation" facility was achieved with little or no impact on hardware complexity or overall throughput and so its inclusion was deemed worthwhile. The fail signal from each processor board would be derived in practice from fuse-blowing safety circuits of the type proposed for the BRATO project, using a similar assassination and suicide failure response philosophy.

6.5 Software Development Tools

6.5.1 The "soft" front panel

For maximum flexibility, it was decided to provide most of the front-panel operator control functions making extensive use of software, instead of hard-wired logic. On power-up the only keys operational are:-
• **RESET.** This key initialises the complete system into a dormant state until either of the next two keys is pressed.

• **START.** Pressing this key takes the machine out of reset and runs any program that exists from location 0 onwards in each store.

• **LOAD.** This key triggers the bootstrap loader logic which starts the MPU selected by rotary panel switch, forcing it to fetch its first instruction from location BO00. This is the address of the common EPROM where utility routines reside including the loader. A program is then loaded in the ASCII-encoded hexadecimal form from the paper tape reader.

The bottom of store is reserved for interrupt vector addresses and various program constants. These are loaded off any program tape that is used. In addition, for the following keys to work, the monitor level of the interrupt system must be enabled by the program when it is run. This allows access to the system monitor, editor and magnetic tape cartridge handler routines.

• **MONITOR.** This key causes all three MPUs to receive a level 3 interrupt, which if enabled sends the processors to location BO02 in store. The monitor program then displays all four accumulators, the program counter and the flag register of the switch-selected MPU, on the panel LED matrix. The remaining keys are examined for activity and if there is none, the monitor returns control to the main program after causing a timing monostable to be triggered. The monostable generates a further level 3 interrupt after one instruction of the main program is executed. In this way program operation is monitored.

• **EXIT.** Pressing this key terminates monitor action.
• JUMP. This allows the processor to jump to any address set on the
word generator keys.

• READ. Data may be examined in any location set on the word generator
keys.

• WRITE. Data set on the word keys may be written directly into the
store location specified by the word keys in a two-stage-operation.

• STEP. Pressing this key initiates single step operation with one
further push to execute each instruction of main program.

A more detailed description of program operation together with a
listing is given in Appendix I.

6.5.2 The bootstrap loader

At the start of the project, no suitable development software for
PACE was available for running on the utility computers. Hence
all software was hand-written and manually assembled from mnemonic form
into 4-digit hexadecimal code. An off-line teletype was then used to
produce an ASCII coded hexadecimal program paper tape. The loader
program converts this data into binary suitable for storing in the
memory. The capacity to change the load address, ignore comments and
rub-out codes was also included.

6.5.3 The hex editor

To assist program development, a simple editor was written to reside
in common utility EPROM. This program works in conjunction with a teletype
or VDU and allows hexadecimal printout of store contents, store
modification, insertion and deletion. Starting at address 00AO, the
program is entered from the system monitor via the jump key. Both the
teletype and paper tape reader interfaces are built into the prototype
system rack, and live on the input/output card.

6.5.4 Magnetic tape cartridge handler

Some time after the initial prototype design had been built, it
was decided to add a means of dumping programs and data directly from
the Cyclone stores to a magnetic storage medium. Various systems were
investigated before the DEC TU58 mini-cartridge system was adopted.
Floppy disc was rejected because of cost and the need to write a
suitable operating system. Ordinary cassette was considered to be too
slow and unreliable. The mini cartridge drive came with a microprocessor
based controller board and interfaces to Cyclone via a standard RS 232
serial data link. Hence interface hardware on the tape card is minimal
and software simplified because of the controller 'intelligence'. Each
cartridge holds 0.25 Mbyte of formatted data in 512 addressable blocks,
using, on Cyclone, a 9600 baud rate. The handler starts at address
B150, accessed from the monitor and allows loading, dumping, self-test
and new cartridge re-tensioning of tape. The Appendix contains the
driving details together with a listing.

6.6 Checking Software Development

6.6.1 Checking philosophy

The basic Cyclone I checking system requires three sets of identical
'results' for each segment of each program. These results can be the entire
contents of all processor registers or just the program counter. The
decision of how much to dump and check is a very important one if a suitable compromise is to be achieved between processing throughput and system reliability. It was decided to check the contents of the three most used registers as a guide to any possible corrupted operation: The program counter and accumulators one and zero. The early timing cycle is illustrated in Fig. 6.7 with three level 5 interrupts signalling task changing and a level 4 interrupt indicating the start of the fourth or error check sector. It was originally intended that the error check sector would reside at base level, but difficulties in maintaining the accurate timing requirements led to wasted processing time. Hence the error checking now takes place all on level 4 (Fig. 6.8). When a level 4 interrupt is received, the check words from sector 2 are dumped as for level 5, but in addition, all processor status is stored in a scratchpad RAM area called 'NEW'. If checking indicates no error, the NEW block is moved to a RAM area known as 'OLD'. When starting on sectors 0, 1 and 2 of the next cycle, it is from the latter area that each processor is reloaded. Of course in the event of an error being detected (although versions 2 and 3 of levels 4 and 5 software allow certain errors) the block transfer does not take place and so the same cycle is repeated automatically. This is a Backward Error Recovery mechanism of the basic fixed recovery point type as described by Anderson et al. The aim of no faulty (or multiple) input/output with the 'outside world' is achieved in the most part by using a system of request flags set by the application program. These are only serviced and cleared in level 4 if no repeat cycle is necessary. There are three main versions of the level 4 and 5 interrupt handlers of increasing sophistication. All allow interrupts on level 2 to take place and so with the monitor on level 3, the interrupt structure in decreasing
FIG. 6.7 EARLY CYCLONE TIMING CYCLE

FIG. 6.8 FINAL CYCLONE TIMING CYCLE
The priority order is as follows:-

Level 2: 100 mS Tachometer timer interrupt
Level 3: Monitor
Level 4: Error checking, recovery and input/output
Level 5: Sector boundary task changing

The facilities offered by the various versions of the interrupt handlers will now be briefly summarised.

6.6.2 Version 1 software

The early versions of the software required all three MPUs to be working and all three sector results had to be identical. If a difference still existed after five successive repeat cycles, then the system shutdown. It was felt that this could be improved upon using the redundant data.

6.6.3 Version 2 software

This intermediate version allowed transient errors in sectors 0 or 1 (but not both in the same cycle). In normal no-error operation a more efficient checking algorithm produced a 3% improvement in overall system throughput. This figure is further improved over that of version 1, if there are transient errors affecting the system. MPU failure still caused total system shutdown.

6.6.4 Version 3 software

The final version keeps the features of the previous handlers, but with a slight cost to throughput on normal running, it allows total failure of one MPU and loss of one program without system shutdown. Use is made of common RAM at address 9000 to allow the system to start up
and run with only two processors available. This particular feature may be deemed undesirable in practice on the grounds that the mission should not begin with failed processors present.

6.6.5 Reset recovery and checkout

On power-up all three MPU's jump to a checkout routine which attempts to verify processor and memory integrity before program cycling begins. The working area of the scratchpad RAM is cleared, and the unused area filled with a test pattern. This was to allow later level 4 software to check for the likelihood of store corruption in the event of an error cycle. After appropriate messages have been sent to the teletype, main program operation begins. If a fault causes cycling to stop, a watchdog monostable times out and forces a brief system reset. A hardware error register is clocked at the same time and is read by the reset checkout routine to distinguish between normal and error resets. On the prototype, various error messages are displayed and a time log printed. However, RAM is not cleared, and thus after successful restart the MPUs can carry on processing the original data. This feature should allow successful recovery from the classes of transient fault which occur in the synchronising hardware itself.

6.7 Duplex CYCLONE

6.7.1 Single hard-fault tolerance

The basic Cyclone I module is capable of ignoring or recovering from a variety of different single or multiple transient faults. This ability was the main force behind the original design, as it was considered vital to reliable operation in the harsh environment of a
railway locomotive. However, the additional requirement of single hard-fault tolerance outlined in Chapter 4 cannot be met by the basic module. In order to achieve this goal, two approaches can be used:

- Add extra hardware redundancy in the form of standby spare MPU boards, etc.
- Use multiple Cyclone modules arranged in hot or cold standby configuration.

Tentative examination of the problems involved with the first method suggests that while possible, the cyclic and prioritised structure of Cyclone would lead to an unacceptable increase in the synchronising hardware, if spares were totally assigned. Some relief from this situation is given if say spare MPU boards were assigned to particular active units. However, it was felt that the multiple Cyclone approach would be easier to implement and more acceptable to potential operating authorities. Hence a duplex arrangement is suggested (Fig 6.9).

6.7.2 Theoretical reliability improvement

From Chapter 2 the reliability of a duplex system made up of modules each with reliability $R_m$ is:

$$R_s = R_m^2$$

This is based in the assumption that the continued operation of one module after the other has failed cannot be allowed for safety reasons. However $R_m$ in the case of Cyclone, is for a self-checking computer module which will shutdown in the event of non-recoverable fault conditions. Hence continued operation of one module is acceptable and the reliability equation now becomes:

$$R_s = R_m^2 + 2R_m (1 - R_m)$$

$$= 2R_m - R_m^2$$
FIG. 6.9 DUPLEX CYCLONE OPERATION

FIG. 6.10 RELIABILITY OF DUPLEX CYCLONE
The improved reliability curve can be seen in Fig. 6.10. The function still has a value of 0.75 at the TMR crossover point of $R_m = 0.5$. This does not of course take into account the reliability of the switching monitor logic which shuts off one Cyclone module and connects the other to the train lines.

6.7.3 **Switching monitor complexity**

The spare Cyclone is kept in hot standby, running in parallel with the active module to minimise changeover delays. The output of the main watchdog timer could be used to trigger switchover, but this may lead to unnecessary loss of modules due to recoverable transient errors. Hence a second monostable is proposed with a period longer than the time required to complete an error reset checkout and restart. Failure to restart would constitute module failure. Not all train functions need to be switched over. The emergency brake operated by the supervisor program could be connected in OR-configuration with both processor modules so that should the active unit lose control and the switch-over logic not operate, the standby unit could still halt the train. With these points in mind, it is though that each module contains enough fault-sensing intelligence to make the monitor switch complexity small enough to be reasonably insignificant when compared with the processing units.

6.8 **Conclusions**

This chapter has described the development of the basic Cyclone principle of time redundancy from a simple concept to a working prototype multiprocessor module. In order to meet single failure requirement, an expanded system with two identical Cyclone modules has been suggested.
The basic development software and hardware have been discussed in outline only, and the following chapters provide a much more detailed description.
REFERENCES FOR CHAPTER 6


CHAPTER 7

HARDWARE DESCRIPTION OF THE PROTOTYPE

MULTIPROCESSOR MODULE - CYCLONE I

7.1 Introduction

The following chapter is a detailed circuit description of the current Cyclone I prototype hardware, containing sufficient information to enable further development work to be carried out or engineering prototypes to be built. The account is divided into six main sections:

* Timing and synchronisation
* Bus control and priority assignment
* Processor
* Address decode and memory
* Input/output
* Control panel

The relationship between these sections is illustrated in the block diagram of the system (Fig. 7.1). These functions have been realised on thirteen 8" x 4.5" wire-wrap development boards which plug into edge connectors on a back "mother-board" in a 19" rack unit. The LED matrix display and its drivers are distributed over eight of the boards, with two columns of six LEDs to each card-front. In addition, the word-generator keys are also arranged two to a card-front on the same eight boards. All the control buttons and rotary select switches lie on a wide card-front attached to the panel buffer board, and input/output is via D-type connectors on the appropriate I/O boards. Most of the logic is built
FIG. 7.1  CYCLONE I PROTOTYPE BLOCK DIAGRAM
using 74 series Low Power Schottky TTL components and the total power supply requirement is between 3.5 and 4 amps depending on the state of the LED display matrix.

7.2 Functional Description

7.2.1 Timing and synchronisation

A block diagram of the timing and sync hardware is shown in Fig. 7.2. The complete circuit diagrams of the two boards, nos. 11 and 12, are given in Figs. 7.3 and 7.4 respectively. The master clock is built around three inverters of X15 and a 2 MHz crystal, so that the INS 8900 microprocessor is run at its' maximum clock rate. The individual MPU clock signals, \( CLK_1, CLK_2 \) and \( CLK_3 \) are derived from this master oscillator using a clock generator formed from gates X11 and D-type flip-flops X16 and X12. The generator circuit for each MPU may be shutdown by the corresponding \( STOP \) signal while the master clock signal is low. When the \( STOP \) signal is eventually released, the flip-flops provide a single phase delay to allow busses to stabilise. This latter feature is only necessary when the processor stop signals produced on the bus control and arbitration board, i.e. \( STOP_1 \), etc. activate. A buffered clock signal \( BCLK \) drives a switch-programmable divider chain, formed by X1, X2 and X3, which provides a maximum division factor of 4096. The final carry output of this divider provides a master interrupt signal which is steered to MPU level 4 or 5 inputs by logic on the control card. On-board DIL switches allow fine and coarse adjustments of the interrupt intervals. Operation of the sync circuitry is rather more involved and reference to the timing diagram (Fig. 7.5) is recommended. On board 11, an interrupt pulse presets the two D-types, X8 and the output X8/9 forces and holds a load condition on the interrupt divider counters, CID. The divider
FIG. 7.2 TIMING AND SYNC BLOCK DIAGRAM
chain cannot now restart until this load is lifted. Meanwhile, the three processors have encountered the first programmed halt instruction in the interrupt handlers 4 and 5. The normally high NHALT output of each MPU now goes low with a 7/8 duty cycle. These signals are inverted and buffered to produce HALT1, etc on board 12. When HALT1, 2 and 3 are all high at the same time, SYNC is produced, SYNC going high clocks D-type X13 on board 11 and causes HALTN to go low. This enables gates X2 on board 12, allowing the next high-low transitions of HALT1, 2 and 3 to clock their respective J-K flip-flops. As each J-K is clocked the output activates its particular MPU clock stopping signal, STOP. However, when the last MPU has clocked its J-K, monostable X14 is triggered, yielding the processor restart signal CONTIN. All this logic ensures that all three MPUs are now synchronised to the same clock edge in their respective instruction cycles. CONTIN is clocked through X13 on board 11 and the first flip-flop X8 changes state and clocks in turn the sector counter X4. Each processor now continues with the next part of the interrupt handler, the change in the sector counter outputs having brought about task swapping via the bus control board. The timing divider chain is still held in 'load' state at this point, however. A pulse on CONT clears down all the J-K flip-flops resetting the system in preparation for the second halt instruction of the interrupt handler. When this is executed by all three MPUs the same basic synchronising sequence takes place again, but this time the D-types X8, board 11 finally release the load signal on the divider counters, allowing them to count towards the next interrupt point.

There are two sections of fault recovery logic associated with the timing and sync boards:-
FIG. 7.3
BOARD 11  CLOCK AND TIMING LOGIC
FIG. 7.4
BOARD 12  SYNC AND TACHO LOGIC
Sync gating formed by gates X8, X17 and X3, X16 on board 12

Watchdog timer X19 and ancillary components on board 12

In normal three-processor operation, X8 and X3 gates are disabled by the inactive FAIL signal. The four input gates X16 and X17 in each case perform a logical -AND function, a single output only going high when all three MPU inputs have gone high. However, if FAIL activates as a result of processor loss (PSF1, etc activating), both sets of gates switch over to 2-out-of-3 working, allowing the two remaining processors to synchronise properly. The action of the watchdog timer is illustrated in Fig. 7.6. Normal system operation is signalled by continuous pulses on the HALTN line and loss of this signal indicates that normal cycling has ceased for one reason or another. The first monostable of X19 board 12 is used in re-triggerable mode such that its Q output remains low all the while HALTN keeps pulsing the A input. Loss of signal causes time-out and the triggering of the second reset monostable in X19. This provides the reset pulse to all the MPUs and other parts of the system. The watchdog time-out clocks error register X18, board 12, the status of which is read by the initialisation reset checkout programs in order to determine whether reset was normal or fault-induced. This register is cleared by manual or power-up reset. The watchdog system is sufficient to deal with most transients that happen to cause a loss of cycling or 'jam' condition, and normally recovery occurs on the first error reset pulse. Depending on the state of HALTN, more serious faults can either result in two or more attempts at restart, or none at all, both resulting in total, but safe, system failure. Providing that 'runaway' processors can be trapped before they reach the interrupt handlers (using methods discussed at the end of Chapter 3), then no faulty outputs should occur.
FIG. 7.5 SYNCHRONIZATION TIMING
There are three sections of logic on board 12 related to the particular user applications of time-keeping and tachometer pulse counting:-

- A divider chain formed by X5, X9, X10, X11, X13 and X7 giving a 10 Hz square wave from the buffered system clock signal BCLK.

- An externally clocked 8-bit counter X12, buffered on to the low byte of the system bus by X4, and automatically cleared when read, by X14.

- A direction decoder X20 which takes a pair of two-phase tachometer signals and has a 1 or 0 output depending on their phase relationship.

The outputs of a slotted-disc type tachometer TØ1 and TØ2 both drive the direction decoder while TØ1 clocks the counter. The counter is given an address in store, D009, use of which activates the peripheral select signal PS9. The level 2 interrupt handler is entered by each MPU every 100 mS in response to an interrupt from the 10 Hz divider output. This short section of program, as well as counting the interrupts to keep track of real-time, also reads the tachometer counter and as it does so, clears it. Hence a measure of distance travelled and velocity can be determined by the main program. Only one interrupt handler, the one attached to the tachometer processor program actually contains the read counter instruction because of the hardware auto-clear facility.

7.2.2 **Bus control and priority assignment**

The block diagram of the bus control logic is shown in Fig. 7.7 and details of the hardware contained on board 7 in Fig. 7.8. Reference should also be made to the timing diagram of Fig. 7.9. All data and addresses of all three processors are multiplexed on to a single 16-bit
HALTN

FIG. 7.6 SUCCESSFUL WATCHDOG RESTART

FIG. 7.7 BUS CONTROL BLOCK DIAGRAM
wide highway, using a simple bus arbitration philosophy. A complete processor bus-access consists of five clock phase intervals: One address valid phase, one inactive phase and three read or write data phases. An MPU of whatever priority at that time, once given access retains it until it has finished that access even if a higher priority MPU requests bus usage part way through. The latter unit has its clock stopped until the bus is free again. The MPU address strobe NADS is used as the bus request signal, hence NADS1 is inverted and becomes REQ1. As an example of operation consider a request from MPU1 in sector 0, which means that it is running program 1. The request signal REQ1 is fed via a multiplexer to the top priority input of a priority encoder chip, X9. The top priority output activates and the program select signal S1 activates by going high. S1 is fed through a second stage of multiplexers to appear as the MPU buffer enabling signal, BUFSEL1. Meanwhile, D-type flip-flop X12 has latched the priority encoder outputs and switched off the first stage of multiplexers to prevent further requests getting through from the other MPUS. The trailing edge of the read or write strobe, IDSl or ODS1, signals the end of the bus access by clocking X12 and releasing the encoder and multiplexers. If MPU2 had requested access part way through the above, then gates X1, X2, X3 would have detected that the bus was already in use, and generated the clock stopping signal STP2. As soon as BUFSEL1 de-activates, BUFSEL2 comes on allowing the 'frozen' address on the MPU outputs to drive the store decoders. The clock stopping logic allows the MPU to continue running one clock phase later to allow the system bus to stabilise. Address data is latched on the address decoder and store boards by the trailing edge of the system address strobe ADS produced on board 7.
FIG. 7.8

BOARD 7  BUS CONTROL LOGIC
FIG. 7.9  BUS CONTROL TIMING
The two banks of multiplexers on either side of the priority encoder have their select inputs controlled by the sector counter outputs SCO, SCl. In this way MPU priorities can be changed with the sector changes, and a processor directed to a particular store to find the program appropriate to that sector.

The reset latches also reside on board 7, and are controlled by a variety of signals. The master RESET resets all three MPUs, in which state they are held until individually released by ST1, ST2 and ST3, or simultaneously by STT. The LOAD button also provides an initial general reset via MON.

7.2.3 Processor

There are three identical processor boards, nos. 8, 9 and 10 (Fig. 7.10), each containing an INS 8900 microprocessor chip and its buffers. Unfortunately, in spite of its NMOS structure, the MPU chip requires three power supply voltages: +5V, +12V and -8V. The RS232 serial links elsewhere, however, also require +12V and -12V, and these are brought on to the MPU board, the -12V being reduced by a zener diode and resistor combination. The clock waveform needs to be 12V peak-to-peak and so the TTL level is boosted by buffer inverter X9. The MPU features its own clock control logic operated via the EXTEND pin, but this was not used as the timing requirements were thought to be too complex. The experimental flexibility offered by external circuitry proved to be advantageous. The data/address bus is buffered by four bidirectional tri-state driver chips Xl-X4 controlled by gates X6, X7. The buffers are switched using the address and data strobes provided by the MPU, but can only be fully enabled on to the system
FIG. 7.10

BOARDS 8, 9 AND 10

MPU
bus by the bus control logic signals BUFSEL, etc. The remaining
output signals are buffered by tri-state chip X10 which can be switched
to high impedance by MPU failure detection circuits driving PSF low.
The whole processor is then effectively isolated because with its address
and data strobes shut off by X10, no bus requests can be made and hence
BUFSEL does not activate to switch on the data bus buffers.

The remaining logic on the MPU board consists of the level 2 or
timer interrupt latch. To avoid unnecessary error cycles being generated
by jumps to the level 2 handler every 100 ms, the level 2 enable is set
and cleared at the end of the level 4 interrupt checking routines. This
requires the interrupt to be latched externally by X8 until enabled, and
then cleared by pulsing the MPU Flag 12 pin 19 under program control.

The 8900 has two zero or base page address modes selectable by a pin,
BPS. When grounded, base page addressing accesses the bottom 256 locations
in store, but when pulled up the base page is split in half between the
bottom 128 and the top 128 locations. The latter option is permanently
selected on Cyclone and the convention adopted of using the bottom
locations as constant storage (it would normally be EPROM anyway),
and the top as scratchpad RAM.

7.2.4 Address decode and memory

The address decoders live on board 6 (Fig. 7.11) together with
most of the bootstrap loader logic. The top three bits of the system bus
are latched by the address strobe ADS, and then drive the inputs of the
three main store decoders X3, X6, X7. Each store board is assigned its
own address decoder, enabled by S1, S2 or S3, the program selects from
the bus control board. On the prototype Cyclone I module, some of
the chip select signals from each decoder are OR-ed to allow access to
common address areas. These are:-
FIG. 7.11

BOARD 6 ADDRESS DECODE AND LOAD LATCH
• Addresses 9000-907F. Common RAM, available to allow inter-program exchange and processor integrity evaluation.

• Addresses B000-B7FF. Common EPROM, containing library utility routines for development work.

• Addresses D000-D0FF. Memory-mapped peripheral space.

The peripheral enable signal PEN, enables a further pair of devices X8 and X9, arranged as a 4 to 16 line decoder using latched bus signals DBO-DB3. The outputs of these decoders consist of sixteen peripheral select signals, although only PS0 to PS11 are used on the prototype.

The loader latch is set by the MON signal from the LOAD button, and has the effect of shutting off the main program store chip selects that appear in response to address 0000, and forcing the common EPROM on to the bus instead, by activating LDEN. The appearance of the EPROM addresses causes LDE to activate, clearing the loader latch and allowing normal store operation to continue.

Program store boards 3, 4 and 5 (Fig. 7.12) are all similar except for board 3 which contains the utility EPROM instead of the panel LED drivers and switch buffers. Each main program area has two sets of I.C. sockets wired in parallel permitting operation with either 1K words of static RAM using four 4K 2114 RAM chips, or 2K words of EPROM using two 16K 2716 chips. Each set is plugged in as required. Also on each board are 256 words of scratchpad RAM assigned to the particular program store on that board. Addresses off the system bus lines DBO-DB1 are latched into X4, X5 by address strobe ADS, and data buffered on and off the same bus by tri-state tranceiver chips X1 and X2.
FIG. 7.12
BOARDS 3, 4 AND 5 MEMORY
7.2.5 Input/output

The first I/O board (board 1, Fig. 7.13) contains the paper-tape interface and TTY/VDU interface circuits. Both interfaces require two peripheral select signals, one for device status and the other for data movement. Only one is in operation at any one time, device changeover being accomplished by setting or clearing bit 8 on PS8: 0 = PTR, 1 = TTY. Latch X9 holds this bit, and is cleared on reset, requiring teletype operation to be selected by program at run time.

The paper tape reader used is a General Electric PTR 66IA photo-electric, stepper-motor drive unit producing eight data bit channels RDO-RD7 and a sprocket hole signal SPR, and requiring a trigger pulse to move the tape on one character, TRIG. Tape movement is triggered by the first examination of status taking RD and PS8 low setting X8 pin 5 high. Until the sprocket signal has clocked X8 back again, this and subsequent status reads will indicate reader busy, with no further trigger pulses being generated until data is read on PS0.

Communication with a terminal is via a bidirectional serial link to RS232 standard and with switchable baud rate: 110 bauds for teletype and 1200 baud for a VDU. A standard LSI UART component, an AY3-1015, is used for the parallel-serial conversion, formatting and error checking. When selected, device status is read on PS0 and data written and read on PS8. Data format is hard-wired on the board, with the clock being derived from a baud-rate generator on the other I/O board. The RS232 levels are produced and received on the serial lines by discrete transistor and resistor circuits, which although simple, are perfectly adequate for laboratory work.

It should be noted that the processor read/write signals IDS1, ODS1 etc., are combined on this board to yield common strobes RD and WR. This is
FIG. 7.14

BOARD 13  TAPE AND COMMON MEMORY
permissable because the individual strobes from the three processors are all mutually exclusive with the bus control philosophy described. In the event of processor failure, pull-down resistors on the bus control board ensure that permanent common strobes are not generated by these OR gates.

The second I/O board, board 13 (Fig. 7.14) houses the magnetic tape cartridge interface and the common memory at address 9000. The same type of UART is used for serial communication, but this one is software programmable, and operates at 9600 baud. The TU58 tape unit will transmit and receive at four times this rate, but to keep reliable operation and use an RS232 standard line, the lower speed was selected. The UART status and command registers are accessed by PS10 and data movement controlled by PS11. A programmable baud-rate generator, X5, provides all the clocks necessary for the various serial channels. A switch on the panel attached to this board allows a selection between high and low baud rates for VDU and TTY respectively.

Sharing the tri-state buffers of the UART on board 13 is the common RAM. This consists of two 128 x 8 bit 6810A chips with an octal latch, X3, to hold the seven low order address bits.

For the purposes of prototype development, all programs may use any of the available peripherals, as outputs from the three main address decoders are OR-ed together. In practice, this will probably not be necessary on a production unit as programs may only use dedicated peripherals in some cases. Separation of the device selects is also desirable from a reliability point of view, to prevent faulty program operation affecting unrelated devices.

Finally, one I/O 'peripheral' is distributed over eight other boards. This consists of the 96 LED matrix display using six independent selects, PS1-PS6, and the 16 switch word-generator on PS7. Each LED is
FIG. 7.13
BOARD 1  INPUT / OUTPUT
FIG. 7.15 LED LATCHES AND SWITCH BUFFERS

L1 - L12 = PANEL LEDs
latched and each switch tri-state buffered on to the system bus (Fig. 7.15).

7.2.6 Control panel interface

The main control panel consists of a wide card-front bolted to board 2 (Fig. 7.16), with two rotary switches and nine push-buttons mounted upon it. The push button functions have been described in Chapter 6 and so only signal details will be given here. All the push-to-make buttons are de-bounced by simple RC filter circuits before driving the interface logic, or being buffered straight out onto the system bus. 'Monitor' is latched first by X6 and X5 latches before being buffered out. The X6 latch remains set all the while a program is run under the monitor in continuous mode. Pressing 'step' clears the X6 latch, allowing the program to be stepped through one instruction at a time with repeated closure of the push-switch. Pressing the 'start' button clears both X5 and X6 latches, and also the reset latches on the bus control board by activating STT. All three processors will run all programs at full speed when operated.

The interrupt steering logic for levels 3, 4 and 5 resides on this board in the form of monostables X7 and steering gates X8. Interrupts from the divider chain on the timing board come in on INT and are stretched into a longer pulse. The most significant bit of the sector counter, SC1, is used, when high, to signal a level 4 instead of level 5 interrupt. When the monitor is run, 4 and 5 are disabled and only level 3 interrupts may be generated on NIR3. The first monostable is now used in conjunction via MOT, to produce a one-instruction delay before firing NIR3. This then realises the step and continuous monitor functions, depending on which is selected.
FIG. 7.16

BOARD 2 CONTROL PANEL INTERFACE
Only one program may be run in non-error check mode using the monitor. The MPU select switch determines which by grounding one of the JCl3 sense inputs to the MPUs. The monitor program looks at this input and causes unselected processors to cycle in a do-nothing loop. This switch is also used to select which program store is loaded via the bootstrap loader.

The mode select switch can be used to 'force' permanent assignment and operation of any processor-program store configuration for a given sector.

'Exit' is used to 'kill' monitor operation by briefly inhibiting the interrupt trigger signal and is used to speed up operation once the mag-tape handler or hex editor have been entered from the monitor.

7.3 Conclusion

This chapter has explained in some detail, the hardware built for the prototype module Cyclone I over the period 1979-1981. Because of the very close relationship between hardware and software in the operation of the machine, both in development and in full cyclic-redundancy modes, full appreciation of the system is not possible until the next chapter on software is read. The machine hardware is versatile however, and can run as a normal, non-redundant multiprocessor if required for program evaluation or development. A detailed description of the software developed to transform Cyclone into a self-checking multiprocessor now follows.
8.1 Introduction

When the Cyclone concept was first considered, one of the aims was to try and ensure system 'transparency' to the applications programmer. This meant that the latter would not have to worry about inserting his own check routines or rollback points. Ideally he would be unaware of the error checking system software, except in certain speed critical applications where the extra overhead would make itself felt. In practice it was found to be extremely difficult to isolate the checking software and prevent it from influencing the main program structure. For very simple test routines, the design goal would be achieved, but once large, complex programs were tried, then it became clear that the user would have to obey some system rules. This chapter outlines the programmer's 'eye-view' of the Cyclone I system and includes a discussion of the special rules, together with detailed descriptions of checking software and evaluation programs developed so far.

8.2 System Structure - Programmer's Viewpoint

8.2.1 System memory map

The allocation of addresses to program stores, scratchpad RAM and input/output is shown in Fig. 8.1. This memory map is the same for all three program functions. Each main program RAM begins at location 0000, as the 8900 draws its first instruction from that address. However, locations 2 to 8 are reserved by the MPU for interrupt pointers. On
FIG. 8.1 CYCLONE I SYSTEM MEMORY MAP
Cyclone, the pointers for interrupt levels 2, 3, 4 and 5 are required, using locations 3, 4, 5 and 6 respectively. There then follows a random list of constants used by all software, before the start of main program at store location O03A. The 128 words of common memory at 9000 onwards may be accessed by all three programs and a system of partitioning has been devised to allow information interchange between programs. As mentioned in Chapter 7, the error checking system may use this RAM to pass and examine code words between processors, permitting the assassination of faulty units by voting. The utility EPROM at B000 onwards is accessible by all programs as well. Listings of routines contained therein together with driving instructions are situated in Appendix I. A glance at these listings will reveal many useful general-purpose subroutines such as Teletype handlers, Binary to Hexadecimal converters, etc. Some main programs however, such as the Reset Handler, do not use the subroutines contained in this utility store in order to avoid common mode failure. Sharing of subroutines within the utility area has led to efficient use of the available space. The common peripherals are located from D000 and their command and status word formats are given in Figs. 8.2 and 8.3. Finally, there are 256 words of scratchpad RAM located between FFOO and FFFF. These scratchpads are 'tied' to a particular program function and cannot be accessed by any other. Their operation is of key importance to the checking system because they hold all the sector change dumps and input/output requests.

8.2.2 The INS8900 instruction set and addressing modes

The instruction set summary is included as Appendix II. There are 45 basic instruction types, all one word long with five possible direct addressing modes:—
PS0  PANEL STATUS - TTY STATUS, DATA

<table>
<thead>
<tr>
<th>15</th>
<th>PTR 7</th>
<th>PTR 6</th>
<th>PTR 5</th>
<th>PTR 4</th>
<th>PTR 3</th>
<th>PTR 2</th>
<th>PTR 1</th>
<th>PTR 0</th>
<th>TBMT</th>
<th>DAV</th>
<th>MON</th>
<th>WRITE</th>
<th>JUMP</th>
<th>READ</th>
<th>LOAD</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PS8  PTR STATUS - KEYBOARD DATA

<table>
<thead>
<tr>
<th>15</th>
<th>R.BUSY</th>
<th>TTY7</th>
<th>TTY6</th>
<th>TTY5</th>
<th>TTY4</th>
<th>TTY3</th>
<th>TTY2</th>
<th>TTY1</th>
<th>TTY0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PS8  TTY SELECT - TTY OUTPUT

FIG. 8.2  PERIPHERAL WORD FORMATS
The image contains a table and a diagram related to peripheral word formats. The tables and diagram are labeled as follows:

**PS9 Tachometer Register**

<table>
<thead>
<tr>
<th>TBMT</th>
<th>DAV</th>
<th>TSB</th>
<th>FE</th>
<th>NP</th>
<th>NB1</th>
<th>NB2</th>
<th>EPS</th>
<th>TACH7</th>
<th>TACH6</th>
<th>TACH5</th>
<th>TACH4</th>
<th>TACH3</th>
<th>TACH2</th>
<th>TACH1</th>
<th>TACH0</th>
</tr>
</thead>
</table>

**PS10 Mag Tape Control, Error Register**

<table>
<thead>
<tr>
<th>TBMT</th>
<th>DAV</th>
<th>TSB</th>
<th>FE</th>
<th>NP</th>
<th>NB1</th>
<th>NB2</th>
<th>EPS</th>
<th>ER3</th>
<th>ER2</th>
<th>ER1</th>
<th>ER0</th>
</tr>
</thead>
</table>

**PS11 Mag Tape Data**

<table>
<thead>
<tr>
<th>TBMT</th>
<th>DAV</th>
<th>TSB</th>
<th>FE</th>
<th>NP</th>
<th>NB1</th>
<th>NB2</th>
<th>EPS</th>
<th>MT7</th>
<th>MT6</th>
<th>MT5</th>
<th>MT4</th>
<th>MT3</th>
<th>MT2</th>
<th>MT1</th>
<th>MT0</th>
</tr>
</thead>
</table>

**FIG. 8.3 Peripheral Word Formats**
• Immediate
  e.g. LI ACO,30

• Register
  e.g. RADD ACO,AC2

• Program-Counter Relative
  e.g. JMP PC,LOOP

• Base Page
  e.g. LD BP,ACO,05

• Indexed
  e.g. ST AC2,ACO,01

In addition, some instructions feature indirect addressing in conjunction with the last three modes:

• Indirect PC-relative
  e.g. LD@ PC,25

• Indirect Base Page
  e.g. ST@ BP,4E

• Indirect Indexed
  e.g. JSR@ AC2,00

With the exception of the register modes, the second half, or least significant byte, of the instruction word is either data (immediate mode) or an address modifier. The modifier or displacement byte allows the final address to be located +127 to -128 (decimal) about the point specified by the main pointer source. Before going any further, the internal architecture of the 8900 from the programmer's point of view should be examined (Fig. 8.4). The 16-bit architecture makes for very easy address manipulation using the four multi-purpose accumulator registers ACO-AC3. Most register instructions operate on all of these, while some have specific features. The branch-on-condition instruction only refers to ACO contents, and hence this register forms the general-purpose accumulator as found in other microprocessors. AC1 is often used for temporary storage, while AC2 and AC3 are available as index registers. A very powerful feature of the 8900 is its internal ten-word push-down stack. Although used extensively by Cyclone software, it is rare to find more than four words on the stack at any one time. Hence the level 1 stack overflow/underflow interrupt is not used, and there has been no need to
FIG. 8.4  INS8900 REGISTER STRUCTURE
include a special stack handler program (1). Finally there is a flag
register with flag bits that are set or cleared by some operations, and
also by specific instructions. Some of these flags drive output pins
directly and are used on Cyclone to control the level 2 interrupt latch.
Other flag pins will be used in conjunction with the available sense inputs
to enable inter-processor communications for test purposes, without
using the system bus.

The MPU response to interrupt is very simple resulting in the stacking
of the program counter only and the clearing of the master interrupt
enable (IEN). This has allowed experimentation with the amount of data
stored on interrupt rather than having to be content with the operation
provided by the chip itself as is the case with many other MPU devices.
In addition, all requests are sorted out by on-chip hardware, leaving the
programmer with the easy task of providing vector addresses in the bottom
store locations as described before.

8.3 Program Design Rules

8.3.1 Input/output operation

The triplication of all main program steps meant that a set of design
rules needed to be established before embarking upon application projects.
The first has been briefly mentioned before, but is restated for
completeness:

- All input/output from the module must take place at one point
  in the system cycle.

From an operational point of view this is an obvious requirement because
otherwise triplicated commands to peripherals could have disastrous
effects. Triplicated input is not so serious, but may lead to unnecessary error cycles if input data changes midway through a cycle. The safety side is of course equally important, and by preventing direct program access to input/output lines, the chances of faulty control of peripheral devices are significantly reduced. For output, this requires the use of two flag words: MFLAG and OFLAG, the message and output flags respectively (Fig. 8.5). In order to request an output, the application program first stores the data in a specific scratchpad location for the device. Then it sets the appropriate bit in OFLAG, and also makes a note of the message type by setting a bit in MFLAG. The program then loops until the OFLAG bit is cleared, signalling successful output. The MFLAG bit is manipulated as required by the application program itself.

Inputs are handled by the IFLAG word. In this instance, the main program may sense data available by inputting status words directly before entering a loop waiting for successful data input by the I/O handler. An example of this is given by the TTY keyboard handler where the UART status is read at intervals by the main program at a particular point in the main command loop. The change of status may occur midway through a system cycle such that sector 0 sees no data available but sectors 1 and 2 do. Hence an error cycle is generated, but in the case of keyboard entry, these cycles are so few and far between as to be insignificant in their effect on throughput. More frequent data entry may require status words to be received in the same way as the data.

3.3.2 Scratchpad RAM usage

When using the scratchpad as a temporary store for variables, it may be necessary to use a separate location for each sector in the system cycle. This variable tripllication is governed by the following rules:
186

**OFLAG**

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IFLAG**

<table>
<thead>
<tr>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MFLAG**

**FIG. 8.5 FLAGWORD FORMATS**

<table>
<thead>
<tr>
<th>SECTOR</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**FIG. 8.6 MPU DUTY CHART**
Triplication of a variable is required if it is likely to be read and modified within one sector of program operation.

Triplication is not required if the variable is read-only or write-only during program sector operation. In such instances the corresponding writes and reads take place in the level 4 interrupt handler at the end. Examples include data to be sent out to a peripheral device.

If a system cycle boundary can be guaranteed to separate a read and write of a variable in program operation, then triplication is not necessary. An example might be the case where, once set, a variable cannot be read until an I/O operation has taken place.

This has meant that all triplicated variables are accessed using indirect addressing and require an address pointer to be adjusted at every sector boundary. Extra checking is carried out on most of the triplicated variables (which include the I/O and Message flag words) to provide an extra safety measure. Checking simply consists of comparing the three corresponding sector words in each case, looking for differences and taking action if any are found.

The address pointers to triplicated variables must be set up in the start-up and initialisation phase of the application program.

Initialisation must also arm the level 4 and 5 interrupt systems to enable cycling.

The last two requirements are relatively trivial once-only operations, but are nevertheless vital. If the interrupt enables are cleared at any time by a transient fault, then cycling stops and all input/output is suspended at once. This condition is quickly spotted by the watchdog timer and an error reset forced.
8.4 Cyclic Operation

8.4.1 Software overview

A Cyclone program store will contain four main sections of software:

- A reset checkout routine that determines system integrity prior to start-up, and recovers from watchdog time-outs.
- The application program.
- The sector change and task swapping routine, run on receipt of a level 5 interrupt.
- The error-checking and input/output routine run in response to a level 4 interrupt.

For timing test purposes a simple counter program to run in all three available stores has been written, outputting the count to the panel LEDs via the interrupt handler system. In order to prove the feasibility of real-time operation, a multi-task tachometer program has been written using the 100 ms timer interrupts. Flow charts and listings for the various handlers and programs are given at the end of the chapter. These should now be studied in conjunction with the following detailed descriptions.

8.4.2 The level 5 interrupt handler

Flowcharts for versions 2 and 3 are given together with a listing for 3 only. On entry into this routine, the first thing to be done is to decrement and test for zero the location in scratchpad called FAIL. This is a counter set to value 3 by initialisation and is used to detect program loss due to processor failure. The program lost is that which cannot run its level 4 end-of-cycle routines because of the failed MPU.
An examination of the processor program duty chart (Fig. 8.6) indicates the link between particular program and MPU failures. Hence for example a faulty MPU1 will not allow program 2 to run because most of the level 4 operation takes place in sector 2. The FAIL counter is steadily decremented to zero by successive level 5 operations, but is normally reset to 3 by a level 4 handler routine, before zero is reached. After the first reset cycle is completed, the three MPUs would normally restart where they left off in their respective programs. However, if one unit has failed, one scratchpad RAM will not contain starting data because of the missing level 4 execution. At the beginning of the second cycle one MPU will attempt restart with whatever happened to be in RAM and a 'wild processor' may result. The FAIL count reaching and being kept at zero causes normal restart to be skipped and instead, a jump to location 2 is executed. This location contains a jump-to-self instruction which effectively keeps the MPU out of harms' way until the next interrupt. In this way, execution of the affected program is eliminated. If normal execution of the level 5 routine is permitted, then the check words are saved next. In this case these are the contents of the program counter (PC), ACO and ACL accumulator registers. The RAM pointer SETCNT is used in conjunction with indirect addressing to store the words, and is decremented after each store instruction. Next, the stack is popped four times to clear it and make room for new data after the sector change. All the remaining indirect store pointers are now decremented. The ones shown in the listing are for the particular multi-task tachometer program described later and include the real-time clock location pointers. Any applications programmer must modify this area of the level 5 routine to suit the particular main program, but this is a very simple task involving adding extra decrement-and-skip-if-zero (DSZ) instructions. A programmed HALT is now executed
to allow a sector change to occur. A new processor will now commence
operation from this point in the routine onwards, and the first thing it
will do is establish whether the current cycle is the first one after a
reset. No data is available for reloading at a sector change during this
first cycle, so the program counter is simply reloaded with the main
program start address. The previously executed reset checkout routines
will have set the RESET flag location in common RAM, to O001. This
reset flag is put in common memory so that in the event of MPU failure
and subsequent first-sector loss by one program, the new MPU taking over
will see the flag set by the other processors. The flag is cleared
by the first level 4 handler operation. If not in a reset cycle, the program
reloads all the internal MPU registers and four words of the stack from an
area of scratchpad RAM called 'OLD' (Fig. 8.7). The level 5 interrupt
system is then re-armed and another HALT instruction executed to resynchronise
all three processors. Finally we have the return from interrupt instruction
(RTI) which pulls the new program counter contents from the stack and
restarts the main program.

8.4.3 The level 4 interrupt handler
From reset, after two level 5 interrupts, a level 4 interrupt is
generated to signal the end of sector 2 main program operation and the start
of error checking and input/output. On entry the three checkwords are
dumped as for level 5 and so are all the register contents and five stack
words (including the last PC contents). The main MPU register contents
are stored in an area of scratchpad RAM known as 'NEW'. The next section
beginning at label .:CHECK is the checking routine made up of the checkword
tests followed by the IOM flagword tests. Originally, the sector change
HALT instruction was situated immediately after the register dumps (see
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>DATA</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>ADDRESS</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFB</td>
<td>SETCNT</td>
<td>FFE5</td>
<td>AC1</td>
<td>FFCF</td>
<td>ADDFL</td>
<td>FFB9</td>
<td>IFLAG0</td>
</tr>
<tr>
<td>FA</td>
<td>FAIL</td>
<td>E4</td>
<td>AC2</td>
<td>CE</td>
<td>IFLAG0</td>
<td>B8</td>
<td>IFLAG1</td>
</tr>
<tr>
<td>F9</td>
<td>ACO S0</td>
<td>E3</td>
<td>AC3</td>
<td>CD</td>
<td>IFLAG1</td>
<td>B7</td>
<td>IFLAG2</td>
</tr>
<tr>
<td>F8</td>
<td>AC1 S0</td>
<td>E2</td>
<td>FLAGS</td>
<td>CC</td>
<td>IFLAG2</td>
<td>B6</td>
<td>OFLAG0</td>
</tr>
<tr>
<td>F7</td>
<td>PC S0</td>
<td>E1</td>
<td>PC</td>
<td>CB</td>
<td>OFLAG0</td>
<td>B5</td>
<td>OFLAG1</td>
</tr>
<tr>
<td>F6</td>
<td>ACO S1</td>
<td>E0</td>
<td>STK+1</td>
<td>CA</td>
<td>OFLAG1</td>
<td>B4</td>
<td>OFLAG2</td>
</tr>
<tr>
<td>F5</td>
<td>AC1 S1</td>
<td>DF</td>
<td>STK+2</td>
<td>C9</td>
<td>OFLAG2</td>
<td>B3</td>
<td>MFLAG0</td>
</tr>
<tr>
<td>F4</td>
<td>PC S1</td>
<td>DE</td>
<td>STK+3</td>
<td>C8</td>
<td>MFLAG0</td>
<td>B2</td>
<td>MFLAG1</td>
</tr>
<tr>
<td>F3</td>
<td>ACO S2</td>
<td>DD</td>
<td>STK+4</td>
<td>C7</td>
<td>MFLAG1</td>
<td>B1</td>
<td>MFLAG2</td>
</tr>
<tr>
<td>F2</td>
<td>AC1 S2</td>
<td>DC</td>
<td>ERCNT</td>
<td>C6</td>
<td>MFLAG2</td>
<td>B0</td>
<td>HRS0</td>
</tr>
<tr>
<td>F1</td>
<td>PC S2</td>
<td>DB</td>
<td>TACH0</td>
<td>C5</td>
<td>HRS0</td>
<td>AF</td>
<td>HRS1</td>
</tr>
<tr>
<td>F0</td>
<td>ACO</td>
<td>DA</td>
<td>LOUT</td>
<td>C4</td>
<td>HRS1</td>
<td>AE</td>
<td>HRS2</td>
</tr>
<tr>
<td>EF</td>
<td>AC1</td>
<td>D9</td>
<td>TEMP</td>
<td>C3</td>
<td>HRS2</td>
<td>AD</td>
<td>MIN0</td>
</tr>
<tr>
<td>EE</td>
<td>AC2</td>
<td>D8</td>
<td>TIN</td>
<td>C2</td>
<td>MIN0</td>
<td>AC</td>
<td>MIN1</td>
</tr>
<tr>
<td>ED</td>
<td>AC3</td>
<td>D7</td>
<td>SWIN</td>
<td>C1</td>
<td>MIN1</td>
<td>AB</td>
<td>MIN2</td>
</tr>
<tr>
<td>EC</td>
<td>FLAGS</td>
<td>D6</td>
<td>TOUT</td>
<td>C0</td>
<td>MIN2</td>
<td>AA</td>
<td>SEC0</td>
</tr>
<tr>
<td>EB</td>
<td>PC</td>
<td>D5</td>
<td>ADSFG</td>
<td>BF</td>
<td>SEC0</td>
<td>A9</td>
<td>SEC1</td>
</tr>
<tr>
<td>EA</td>
<td>STK+1</td>
<td>D4</td>
<td>ADSEC</td>
<td>BE</td>
<td>SEC1</td>
<td>A8</td>
<td>SEC2</td>
</tr>
<tr>
<td>E9</td>
<td>STK+2</td>
<td>D3</td>
<td>ADMIN</td>
<td>BD</td>
<td>SEC2</td>
<td>A7</td>
<td>SFLAG0</td>
</tr>
<tr>
<td>E8</td>
<td>STK+3</td>
<td>D2</td>
<td>ADHRS</td>
<td>BC</td>
<td>SFLAG0</td>
<td>A6</td>
<td>SFLAG1</td>
</tr>
<tr>
<td>E7</td>
<td>STK+4</td>
<td>D1</td>
<td>ADMFL</td>
<td>BB</td>
<td>SFLAG1</td>
<td>A5</td>
<td>SFLAG2</td>
</tr>
<tr>
<td>E6</td>
<td>ACO</td>
<td>D0</td>
<td>ADIFL</td>
<td>BA</td>
<td>SFLAG2</td>
<td>A4</td>
<td>ERFLG</td>
</tr>
</tbody>
</table>

FIG. 8.7 SCRATCHPAD MEMORY MAP
flow charts for L4 VSN.2 at end), but this was moved when version 3 was written allowing single MPU failure. The checking procedure adopted is illustrated in version 3 charts 1 and 2. The first procedure developed, involved the comparison of the three corresponding sets of check and IOM words, and a repeat cycle if any differences were detected. Version 2 software was more tolerant and allowed errors in the sector 0 and 1 dumps.

First of all, the checkwords for sectors 0 and 2 are compared. If these are identical then no further checkwords are examined, and a similar test is carried out on the IOM flagwords. Hence this yields a fast exit from the checking routines if:-

- No errors exist and all three program sectors are identical.
- There is an error in the sector 1 results only.

If a difference is found then the check routine moves on to compare sectors 0 and 1. If these checkwords are equal then it is assumed that the error must lie in sector 2. Version 2 software then aborted the cycle and called for retry. If however 0 and 1 are not the same then a further check is carried out, comparing the results of sectors 1 and 2. If they are not equal then the error is once again assumed to be in sector 2 and the old routine would have abandoned the cycle. Equality means that sector 2 results are acceptable and the cycle as a whole is passed as correct and no retry is ordered. This means that odd 'glitches' occurring during sectors 0 and 1 of main program operation do not cause cycle loss, thus speeding up processing even when in a noisy environment. Version 3 possesses a further test feature, starting at :S2FAIL in the listing, which caters for sector loss due to MPU failure. Previously, an error in sector 2 was deemed to be fatal because the 'NEW' dumps might have been corrupted. Now an
MPU loss will cause one of the checkword dumps to be lost. If this missing
dump belongs to sectors 0 or 1 then the checkword dumps for sector 2 will
end up occupying the space for those of sector 1. The area allocated to
2 will still have the dump for the previous cycle when the checking routines
are commenced. Hence the latter will assume sector 2 is at fault because
the results for the two cycles will probably be different. The new routine
compares the checkwords it finds in the sector 1 dump with the values in
the sector 2 NEW dump. If they are the same then the program assumes that
sector 2 is in fact intact and no error cycle is produced. Once again
however, if sector 2 results are found to be faulty, then a retry is
automatic. Once the cycle is accepted as being correct enough for
input/output, the error cycle counter ERCNT is set to five and a block
move commenced, replacing the contents of the current OLD RAM area with
those recently dumped into the NEW area. With the starting register contents
for the next cycle ready for loading, the next section to be executed
concerns communication with peripherals, starting at :RET. A fast exit
is provided in the event of no output being required, to the test for input
request routine. Outputs to the LED matrix are dealt with first,
followed by teletype requests. If the TTY status word indicates that the
device is not ready, then the output request cannot be granted and the flag
remains uncleared into the next cycle. The section starting with label :TTS
is used to switch the TTY/PTR select flip-flop on the I/O board to teletype.
The input data section is now reached after clearing all three output
request flagwords. Notice that as for the output flagword, an indirect
addressing mode is used off the as yet unreset pointer. This allows for
sector loss and ensures that the current flagword is read. A fast exit
for no input request is similarly provided. Input from word generator keys
and teletype keyboard is catered for. Data for the latter is assumed to be ready if the appropriate request bit is set, because the main program tests TTY status directly.

After the input/output is taken care of, we move on to the remaining 'housekeeping chores' at :SKIP. The FAIL counter is reset back to 3 and another block move takes place, this time moving the IOM flagwords to the 'last correct IOM' area of RAM. This area of memory is used to replace the existing flagwords if the cycle is aborted and is about to be re-run. Hence a record of the last correct flags used is kept and updated with each correct cycle. All the various store address pointers are now reset in preparation for the next cycle, whether it be normal or retry.

The level 2 timer interrupt system is now briefly enabled to allow any pending interrupts, latched externally to come through. The level 4 interrupt system is also re-armed and the program jumps to :REFIL in the level 5 routines to begin restart following the sector change from 2 to 0.

The error handling routines on the end of the listing are only entered when a cycle is aborted, and allow four successive error cycles before starting a step-by-step attempt at recovery. When ERCNT reaches zero after the loss of five cycles all system variables in scratchpad RAM between FFC6 and FFF9 are cleared and a return address of 0001 stored to cause program restart and reinitialisation. If a sixth successive cycle is lost, all main program variables are cleared as well and the program reinitialised. A further error cycle will cause the processor to wait in a small loop for the watchdog timer to time-out and force total system reset. The block move beginning at :MOV3 moves the last correct IOM flagwords to the current flag area for error cycles occurring prior to ERCNT reaching zero.
8.4.4 The reset handler routine

The reset handler routine is entered, naturally enough, straight after a system reset and begins by setting the reset flag in location 9000 of common RAM to 0001. As described in the chapter on Cyclone development, simple processor instructions are used at first to check the four accumulators ACO-AC3 with a jump-to-self instruction executed if faulty operation is detected. The flag register and internal push-down stack are similarly checked next, and if the errors are detected and execution reaches thus far, the error register in the synchronisation logic at location DOOA, is read. If it is zero then we have normal manual or power-up reset and operation continues with the next section starting at :SET which clears the working scratchpad RAM area and puts data = 5555 into the unused locations. It is proposed that further versions of level 4 handlers will check this unused store area for corruption when an error is detected, on the basis that there is a strong probability that the used area is untouched if tests indicate no errors. If an error reset condition exists, then this section is skipped and execution resumes at :ERESET. For the purposes of demonstration, ERROR is displayed on the LED matrix and a time log printed on the teletype or VDU. Whatever the reason for reset, the following sections from :NORM onwards are common to both. To have reached this far implies that no hard fault exists within the CPU and so an appropriate message is output to the VDU. The message identifies the particular CPU concerned. A simple alternating bit pattern scratchpad RAM test is now performed with positive or negative messages displayed on the VDU according to the result. If negative, then once again the machine halts in a short loop, otherwise the LED matrix is cleared and control is transferred via an RTS instruction to the instruction at location 0001, usually a jump to main program.
Worthy of mention is the TTY/VDU handler subroutine beginning at CHOUT. This subroutine enables three programs to share the same output channel by assigning priorities in software. On entry, the programs check status to see if the device is ready. If it is, then they check the MPU internal flag bit LINK which is initially clear. The priority system now comes into play by causing the program to enter delay loops of varying duration according to the MPU running them.

On exit from the delay, the program once again checks status and if the channel is still available, sets the LINK flag and proceeds with output. Further entry into the subroutine will involve skipping the delay until LINK is cleared by the calling program at the end of message output. In this way, the other MPUs are locked out until the current message is completed.

8.5 Test Programs

8.5.1 Simple cycle check program

In order to try out the handler routines and evaluate the effect of varying sector lengths on throughput, a simple test program was written, which runs in all three program stores. The listing is given at the end of this chapter, but note that the initial instructions and constants are not shown as they are given in full with the multi-task tachometer program listing which follows. Hence the label :INIT corresponds to location 003A in program memory. The initial routines select TTY/VDU output, enable the level 4 and 5 interrupts and set up the checkword dump pointer, SETCNT. The FAIL counter is set to 3, the error cycle counter ERFLG cleared and finally all the remaining dump pointers are initialised.

The program proper begins at :TIME which marks the start of a real-time clock handler. This time routine maintains an accurate record of elapsed
time formatted as hours, minutes, seconds and tenths of seconds for the use of the error reset handler when displaying a time log. The 100 mS level 2 interrupt handler increments the SFLAG locations addressed by pointer ADSFG. This counter is cleared when it reaches ten and the seconds counter SECS incremented. The count then ripples through to hours as required. The label :OUT marks the start of the second half of the main program. The current OFLAG is checked to see if any outputs are pending and if not then the message flagword MFLAG is examined. Now this section of the program increments a counter in ACL every alternate system cycle and displays the count on LED matrix row 1. On the other alternate cycle, the contents of the error cycle counter are displayed on row 2, it can be seen how the MFLAG contents are not cleared by the level 4 handler and are used to determine which row is to be displayed next. Also, all communication with the 'peripheral' LED matrix is via the LED buffer memory location LOUT.

This program is repeated in stores 2 and 3 except that program 2 displays on LED rows 3 and 4, and program 3 on rows 5 and 6. It has been used to evaluate two aspects of Cyclone performance:-

- The relationship between sector length as determined by the interrupt timer switches and the amount of processor output.
- The number of error cycles resulting from various quantities of power supply interference.

System performance while running this program will be discussed in the next chapter.
8.5.2 The multi-task tachometer program (MTT)

The multi-task tachometer program was developed to prove that the time redundancy employed in Cyclone would not prevent real-time operation of the type required for automatic vehicle control. The program takes the following inputs from peripheral devices:

- Speed data in the form of an 8-bit binary number, loaded from a counter in hardware read and cleared every 100 ms. This counter is clocked by one phase of a 2-phase tachometer head output.
- Direction data on an MPU sensor input where Forward = 1, Reverse = 0. This information is decoded from the 2-phase signal by a hardware decoder.
- Real-time clock data in the form of data held in a location in store scratchpad which is incremented every 100 ms.
- VDU keyboard input either setting the clock to an initial value, or requesting a display of the current time.

All these inputs are processed to yield the following peripheral outputs:

- A continuously updated LED matrix display giving the speed in mph and an arrow indicating direction.
- A VDU display of current time or time elapsed in hours, minutes and seconds when requested.
- A VDU display indicating operator error in the event of incorrect keyboard action.

The listing includes the base page constants used by other programs as well. These constants occupy locations 0009 to 0039 in store, although the split base-page addressing would allow further locations up to 007F to be used. The initialisation routines in the particular listing given at the back
of this chapter are for version 2 interrupt software, and would normally be replaced by those shown attached to the simple cycle check program for version 3. The timekeeper starting at :LOOP is the same as for the previous program, but from here on the various routines become rather complicated. The relatively simple operations of input/output with the teletype or VDU are made more involved because only one input and/or output may occur per cycle, and so a system of message flags is used to tell the main program which character to print next, whether it be teletype input echo or part of the operation error message. The message handler prints out until a full-stop marker is found. The input routines accept a 'T' as a time request, numbers to set the time buffer in AC3, and a newline (N/L) character to transfer AC3 to the time locations in scratchpad. Any other input causes "Operator Error" to be printed.

The LED display subroutine at :DISPY builds up a two-digit number on the panel, together with a direction arrow. One row of the display is updated every cycle, using the tachometer and sense inputs combined with the row number to form an address in AC2. Subroutine :CHAR performs this task and the display line is built up until ready for output whereupon it is transferred to the LED output buffer LOUT. The character generator for the display runs from location 0200 onwards.

The time printing routine at :PRTIM prints the real-time values in the form HH:MM:SS (N/L). After an input 'T' on the keyboard, the routine is entered at :PRTIM which sets up MFLAG for printing the time generally and a space specifically. The section at :PRTIM3 is used to set the output flag OFLAG before returning to the main program command loop to wait for the end of the system cycle. After printing the routine is always re-entered at :PRTIM2 which tests to see which character needs to be printed next. Finally, :EXIT prints a newline and clears the message flag bit for time output. With the message flag bit cleared, the program only
returns to the routine again on T being pressed again.

8.5.3 The level 2 timer interrupt handler

Both the simple cycle check and MTT require the level 2 handler which normally resides at the top end of store. When entered at :L2, the external interrupt latches attached to each MPU are cleared, followed by the interrupt enable itself. Only one of the three level 2 handlers then reads the tachometer register, partly because only one main program will usually be interested, and partly because the auto-clear in hardware will only allow the register to be read once. The triplicated 100 ms interval counters SFLAG are then incremented before the RTI instruction is executed.

8.6 Conclusion

Having covered both hardware and software description of the Cyclone prototype, the reader should now be able to appreciate how the close inter-dependence of the two areas is characteristic of the checking philosophy used. While the programs appear to be increased in complexity by the need for message flags, etc, the basic design rules are reasonably easy to follow and are few in number. The sacrifices made in theoretical maximum throughput for a non-redundant system, and the solutions to synchronising problems may be compared with those in other far more complex systems such as that described by Ihara, et al (2). The special interrupt handling software for Cyclone can be improved in capability and speed still further and suggestions for more work in this area will
be discussed in Chapter 11. Meanwhile, the next chapter will present some of the Cyclone performance data derived by using the check programs described.
LEVEL 4 INTERRUPT HANDLER VSN. 2  CHART 1.
LEVEL 4 INTERRUPT HANDLER VSN. 2  CHART 2
LEVEL 4 INTERRUPT HANDLER VSN. 2  

CHART 4.

E

TTY OIP SERVICE ROUTINE

PRINTER READY?

No

H

Yes

SEND "TOUT" TO TTY PRINTER

G

F

PANEL W/O SWICH I/F REQUEST?

Yes

INPUT SWITCH DATA TO "SWIN"

No

TTY Z/I/P REQUEST?

Yes

INPUT TTY DATA TO "TIN"

No

CLEAR ALL INPUT REQUEST FLAGS

J
LEVEL 4 INTERRUPT HANDLER VSN. 2  CHART 5.
INTERRUPT

DUMP PC, ACO
AC1 FOR
CHECKING

HALT AND WAIT
FOR
SECTOR CHANGE

CLEAR
STACK

DECREMENT
ALL STORE
POINTERS

IN
RESET CYCLE?
Yes

K

PUSH ADDRESS
0001
ON TO STACK

No

LOAD UP NEXT
PROGRAM STATUS
FROM
"OLD" RAM AREA

RE-ARM LI
INTERRUPT
SYSTEM

HALT AND WAIT
FOR
SYNCHRONISED
START

RETURN
FROM
INTERRUPT

LEVEL 5 INTERRUPT HANDLE VSN. 2
INTERRUPT

DUMP ACO, AC1
For checking

DUMP REGISTERS
AND STACK IN "NEW"

COMPARE RESULTS
OF SECTORS 0 & 2

2 OUT OF 3 AT
LEAST OK. SKIP
TO I/O CHECK

RESULTS IDENTICAL?

COMPARE RESULTS
OF SECTORS 0 & 1

RESULTS IDENTICAL?

COMPARE RESULTS
OF SECTORS 1 & 2

RESULTS IDENTICAL?

LEVEL 4 INTERRUPT HANDLER VSN. 3  CHART 1
LEVEL 4 INTERRUPT HANDLER VSN. 3

CHART 2.
LEVEL 4 INTERRUPT HANDLER VSN. 3  CHART 3.
LEVEL 5 INTERRUPT HANDLER VSN. 3
.TITLE - LEVEL 5 INTERRUPT HANDLER VSN. 3

ACFA :L5 DSZ BP,FAIL ;L4 FAILURE?
1904 JMP PC,CONTU ;PROGRAM OK
8CFA ISZ BP,FAIL ;PROGRAM ABORT
5002 LI ACO,02
DOEB ST BP,ACO,OLD+4
1913 JMP PC,REFIL
BOFB :CONTU ST@ BP,SETCNT ;SAVE ACO
ACFB DSZ BP,SETCNT
5C40 RCPY ACO,AC1
BOFB ST@ BP,SETCNT ;SAVE AC1
ACFB DSZ BP,SETCNT
6400 PULL ACO
BOFB ST@ BP,SETCNT ;SAVE PC
ACFB DSZ BP,SETCNT
6400 PULL ACO ;CLEAR STACK
6400 PULL ACO
6400 PULL ACO
6400 PULL ACO
ACD5 DSZ BP,ADSPG ;DEC TIMER COUNTER ADD
ACD4 DSZ BP,ADSEC ;DEC SECONDS COUNT ADD
ACD3 DSZ BP,ADMIN ;DEC MINUTES COUNT ADD
ACD2 DSZ BP,ADHRS ;DEC HOURS COUNT ADD
ACD1 DSZ BP,ADMFL ;DEC MFLAG POINTER
ACD0 DSZ BP,ADIFL ;DEC IFLAG POINTER
ACCF DSZ BP,ADOFL ;DEC OFLAG POINTER
0000 :REFIL HALT ;SECTOR CHANGE
A162 LD@ PC,RESET ;RESET CYCLE?
4514 BOC NREQO,REST
C0E7 LD BP,ACO,OLD ;LOAD NEXT STACK CONTENTS
6000 PUSH ACO
C0E8 LD BP,ACO,OLD+1
6000 PUSH ACO
C0E9 LD BP,ACO,OLD+2
6000 PUSH ACO
C0EA LD BP,ACO,OLD+3
6000 PUSH ACO
C0EB LD BP,ACC,OLD+4
6000    PUSH ACO
6000    CRF ACO
CCED    LD BP,AC3,OLD+6 ;LOAD NEXT AC3
CS8E    LD BP,AC2,OLD+7 ;LOAD NEXT AC2
C4EF    LD BP,AC1,OLD+8 ;LOAD NEXT AC1
COFO    LD BP,AC0,OLD+9 ;LOAD NEXT AC0
3500    :RETN PFLG IE5
3580    SFLG IE5 ;RE-ARM L5 INTERRUPT
0000    HALT ;RESYNCHRONIZE
7C00    RTI
DOEB    :RESET ST BP,AC0,OLD+4 ;RESET CYCLE
5000    LI ACO,00
B14A    ST@ PC,RESET ;CLEAR RESET FLAG
19b8    JMP PC,REPIL+3
.TITLE - LEVEL 4 INTERRUPT HANDLER VSM. 3

D0E6 :L4 ST BP,ACO,NEW+9 ;SAVE ACO
D0FB ST BP,SETCNT ;SAVE CHECK COPY
ACFB DSZ BP,SETCNT
5C40 RCPY ACO,AC1
B0FB ST BP,SETCNT ;SAVE AC1
D0E5 ST BP,ACO,NEW+8
ACFB DSZ BP,SETCNT
6400 PULL ACO
B0FB ST BP,SETCNT ;SAVE PC
D0E1 ST BP,ACO,NEW+4
6400 PULL ACO
D0E0 ST BP,ACO,NEW+3 ;SAVE STACK
6400 PULL ACO
D0DF ST BP,ACO,NEW+2
6400 PULL ACO
D0DE ST BP,ACO,NEW+1
6400 PULL ACO
D0DD ST BP,ACO,NEW
D0E4 ST BP,AC2,NEW+7 ;SAVE AC2
DCE3 ST BP,AC3,NEW+6 ;SAVE AC3
0400 CFR ACO
D0E2 ST BP,ACO,NEW+5 ;SAVE FLAG REG
53F0 LI AC3,FO
7B01 :CHECK AISZ AC3,01
C300 LD AC3,AC0,00
C706 LD AC3,AC1,06
5840 RXOR ACO,AC1 ;COMPARE SECTORS 0 & 2
4503 BOC NREQ0,TEST2 ;ERROR?
FC12 SKNE BP,AC3,CONST10 ;FINISHED?
1912 JMP PC,ICCHK ;SKIP TO I/O CHECK
19F8 JMP PC,CHECK
7B01 :TEST2 AISZ AC3,01
C300 LD AC3,AC0,00
C703 LD AC3,AC1,03
5840 RXOR ACO,AC1 ;COMPARE SECTORS 0 & 1
4503 BOC NREQ0,TEST3 ;ERROR?
FC15 SKNE BP,AC3,CONST13 ;FINISHED?
1924 JMP PC,S2FAIL ;POSSIBLE SECTOR 2 FAILURE
215

19F8:  JMP  PC, TEST2
53F0:  TEST3  LI  AC3, F0
7B01:  AISZ  AC3, 01
C300:  LD  AC3, ACO, 00
C703:  LD  AC3, AC1, 03
5840:  RXOR  ACO, AC1  ; COMPARE SECTORS 1 & 2
451D:  BOC  NREQO, S2FAIL  ; POSSIBLE SECTOR 2 FAILURE
FC12:  SKNE  BP, AC3, CONST10  ; FINISHED?
1901:  JMP  PC, IOCHK  ; GO ON TO I/O CHECK
19F8:  JMP  PC, TEST3+1
53C3:  IOCHK  LI  AC3, 03  ; CHECK I/O REQUESTS
7B03:  AISZ  AC3, 03
C300:  LD  AC3, ACO, 00
C702:  LD  AC3, AC1, 02
5840:  RXOR  ACO, AC1  ; COMPARE 0 & 2 REQUESTS
4503:  BOC  NREQO, TEST4  ; ERROR?
FC2E:  SKNE  BP, AC3, CONST38  ; FINISHED?
191E:  JMP  PC, CANCEL  ; SKIP FURTHER CHECKS
19F8:  JMP  PC, IOCHK+1
C302:  TEST4  LD  AC3, ACO, 02
C701:  LD  AC3, AC1, 01
5840:  RXOR  ACO, AC1  ; COMPARE 0 & 1 REQUESTS
4504:  BOC  NREQO, TEST5  ; ERROR?
FC10:  SKNE  BP, AC3, CONST38  ; FINISHED?
190B:  JMP  PC, S2FAIL  ; POSSIBLE SECTOR 2 FAILURE
7BFD:  AISZ  AC3, FD
19F8:  JMP  PC, TEST4
C300:  TEST5  LD  AC3, ACO, 00
C701:  LD  AC3, AC1, 01
5840:  RXOR  ACO, AC1  ; COMPARE 1 & 2 REQUESTS
4505:  BOC  NREQO, S2FAIL  ; POSSIBLE SECTOR 2 FAILURE
FC2E:  SKNE  BP, AC3, CONST38  ; FINISHED?
190F:  JMP  PC, CANCEL  ; NO FATAL ERRORS
7B03:  AISZ  AC3, 03
19F9:  JMP  PC, TEST5

9000:  RESET  (RESST)  ; RESET FLAG VECTOR
C0F6:  S2FAIL  LD  BP, ACO, F6  ; MISSING SECTOR CHECK
C4E6    LD BP, AC1, E6
5840    RXOR ACO, AC1
4557    BOC NREQO, ERRORX ; SECTOR 2 LOSS, ABORT CYCLE
C4E5    LD BP, AC1, E5
5840    RXOR ACO, AC1
4553    BOC NREQO, ERRORX ; SECTOR 2 LOSS, ABORT CYCLE
C4E5    LD BP, AC1, E5
4553    BOC NREQO, ERRORX ; SECTOR 2 LOSS, ABORT CYCLE
C4E5    LD BP, AC1, E5
4553    BOC NREQO, ERRORX ; SECTOR 2 LOSS, ABORT CYCLE
5005    : CANCEL LI ACO, 05
DODC    ST BP, ACO, ERCNT ; RESET ERROR CYCLE COUNTER
53DC    : MOVE LI AC3, DC ; BLOCK MOVE, NEW TO OLD AREA
7B01    AISZ AC3, 01
C300    LD AC3, ACO, 00
D30A    ST AC3, ACO, 0A
FC33    SKNE BP, AC3, CONST43 ; FINISHED MOVE?
1901    JMP PC, RET
19FA    JMP PC, MOVE+1
AOCF    : RET LD@ BP, ADOFL ; GET O/P REQUEST FLAGWORD
4117    BOC REQ0, EXITY ; ANY REQUESTS?
A82F    AND BP, CONST39
4104    BOC REQ0, TTO ; TTY O/P REQUEST?
CC09    LD BP, AC3, CONST1 ; LED MATRIX O/P
6B00    RADD AC3, ACO
C3A9    LD BP, ACO, LOU1 ; GET LED ROW DATA
D300    ST AC3, ACO, 00 ; DISPLAY DATA
AOCF    : TTO LD@ BP, ADOFL
2C0C    SHR ACO, 6, 0
410B    BOC REQ0, CLEARF+1 ; NO TTY O/P REQUESTS?
4407    BOC BIT1, TTS ; TTY SELECT?
A009    : TTOA LD@ BP, CONST1
A80D    AND BP, CONST5
410A    BOC REQ0, EXITY ; TTY PRINTER READY?
C0D6    LD BP, ACO, TOUT ; GET TTY O/P DATA
MOB OR EP,CONST6
SEND DATA TO TTY

BOOB ST@ BP,CONST3

1902 JMP PC,CLEARF

ACOE :TTS LD@ BP,CONST6

BOOB ST@ BP,CONST3

;SELECT TTY

5000 :CLEARF LI ACO,00

DOCB ST BP,ACO,OFLAG0

DOCA ST BP,ACO,OFLAG1

DCC9 ST BP,ACO,OFLAG2

ACDO :EXIT LD@ BP,ADIPL

410C BOC R2Q0,MOVIO

4604 BOC BIT1,TTN

COO9 :SWT LD BP,AC3,CONST1

7BC7 AISZ AC3,07

C300 LD AC3,ACO,00

DOD7 ST BP,ACO,SWIN

A00B :TTN LD@ BP,CONST3

A80F AND BP,CONST7

DOD8 ST BP,ACO,TIN

;STORE KEYBOARD DATA

5000 :OUTX LI ACO,00

DOCE ST BP,ACO,IFLAG0

DOCD ST BP,ACO,IFLAG1

DOCC ST BP,ACO,IFLAG2

53BC :MOVIO LI AC3,EC

;BLOCK MOVE, FLAGWORDS

7B01 AISZ AC3,01

C300 LD AC3,ACO,00

D3EB ST AC3,ACO,EB

FC38 SKNE BP,AC3,CONST4S

1901 JMP PC,RETZ

19PA JMP PC,MOVIO+1

5003 :RETZ LI ACO,03

;RESET S2 FAIL COUNTER

D0FA ST BP,ACO,FAIL

50BC LI ACO,EC

;RESET ALL POINTERS

D0D5 ST BP,ACO,ADSPG
218

50BF LI ACO, BF
DOD4 ST BP, ACO, ADSEC
50C2 LI ACO, C2
DOD3 ST BP, ACO, ADMIN
50C5 LI ACO, C5
DOD2 ST BP, ACO, ADHRS
50C8 LI ACO, C8
DOD1 ST BP, ACO, ADMFL
50CE LI ACC, CE
DODO ST BP, ACO, ADIFL
50CB LI ACO, CB
DOCF ST BP, ACO, ADOPF
50F9 LI ACO, F9
DOFB ST BP, ACO, SETCNT

3400 PFLG IE4
3280 SPLG IE2 ; ENABLE TIMER INTERRUPT
3980 SPLG IEN ; MASTER INTERRUPT ENABLE
3200 PFLG IE2 ; INHIBIT TIMER INTERRUPT
3480 SPLG IE4 ; RE-ARM L4 INTERRUPT
9825 JMP* BP, REFL ; EXIT TO RELOAD ROUTINES

ACDC :ERRORX DSZ BF, ERCNT ; DEC ERROR CYCLE COUNT
8CA4 ISZ BP, ERFLG ; INC ERROR CYCLE TOTAL
5C00 NOP
C0DC LD BP, ACO, ERCNT
4208 BOC NSIGN, CLRIO ; MORE THAN 6 ERROR CYCLES?
410B BOC REQ0, TRY1 ; 5 ERROR CYCLES?
53A4 MOV3 LI AC3, A4 ; MOVE LAST CORRECT FLAGS
7B01 AISZ AC3, 01
C300 LD AC3, ACO, 00
D315 ST AC3, ACO, 15
FC39 SKNE BP, AC3, CONST49 ; FINISHED MOVE?
19DC JMP PC, RETZ
19FA JMP PC, MOV3+1
7000 :CLRIO CAI ACO, 00
45FF BOC NREQ0, CLRIO+1 ; WAIT FOR TIME-OUT
53B9 :TRY2 LI AC3, B9 ; CLEAR RAM INCL FLAGWORDS
1901 JMP PC, TRY1+1
53C5 :TRY1 LI AC3, C5 ; CLEAR RAM EXCL FLAGWORDS
5000 LI ACO,00 ;CLEAR ROUTINES
7801 AISZ AC3,01
D300 ST AC3,AC0,00
FC27 SKNE BP,AC3,CONST31 ;FINISHED CLEARING?
1901 JMP PC,SET
19FB JMP PC,TRY1+2

5001 :SET LI ACO,01
DOEB ST BP,ACO,OLD+4 ;SET RETURN ADDRESS = 0001
19CD JMP PC,RETZ

.END
.TITLE RESET HANDLER ROUTINE  WGM 1981

5390  :START  LI  AC3,90  ;AC3 = COMMON MEMORY ADDRESS
2B10  SHL  AC3,8,0
5001  LI  ACO,01
D300  ST  AC3,ACO,RESET  ;SET RESET FLAG

5055  :RECO  LI  ACO,05  ;TEST ACO
2020  ROL  ACO,16,0
78AB  AISZ  ACO,AB
19FF  :STOP  JMP  PC,STOP  ;HALT IF ERROR
5155  LI  AC1,55  ;TEST AC1
2120  ROL  AC1,16,0
79AB  AISZ  AC1,AB
19FF  :STOP1  JMP  PC,STOP1  ;HALT IF ERROR
5255  LI  AC2,55  ;TEST AC2
2220  ROL  AC2,16,0
7AAB  AISZ  AC2,AB
19FF  :STOP2  JMP  PC,STOP2  ;HALT IF ERROR
5355  LI  AC3,55  ;TEST AC3
2320  ROL  AC3,16,0
7BAB  AISZ  AC3,AB
19FF  :STOP3  JMP  PC,STOP3  ;HALT IF ERROR

50CB  :FLAGS  LI  ACO,CB  ;TEST FLAG REGISTER
0800  CRF  ACO
5000  LI  ACO,00
0400  CFR  ACO
70CC  CAI  ACO,CC
45FF  :STOP4  BOC  NREQC,STOP4  ;HALT IF ERROR
0800  CRF  ACO
50FF  LI  ACO,FF
0400  CFR  ACO
20C2  ROL  ACO,1,0
7004  CAI  ACO,04
45FF  :STOP5  BOC  NREQC,STOP5  ;HALT IF ERROR
3580  SFLG  IE5  ;ARM L5 INTERRUPTS
5100 :STACK LI AC1,00 ;TEST STACK OPERATION
500A LI ACO,0A
7901 :FILL AISZ AC1,01
6100 PUSH AC1
78FF AISZ ACO,PF
19FC JMP PC,FILL
6400 :EMPTY FULL ACO
5840 RXOR ACO,AC1
45FF :STOP6 BOC NREQ0,STOP6 ;HALT IF ERROR
79FF AISZ AC1,PF
19FB JMP PC,EMPTY

CC09 LD BP,AC3,CONST1
C30A LD AC3,ACO,EREG ;READ ERROR REGISTER
A835 AND BP,CONST45
450E BOC NREQ0,ERES6 ;NORMAL OR ERROR RESET?
5055 :SET LI ACO,55 ;SET RAM TEST PATTERN
2810 SHL ACO,8,0
7855 AISZ ACO,55
52FF LI AC2,PF
2A10 SHL AC2,8,0
1902 JMP PC,OVER
5000 :CLR LI ACC,00 ;CLEAR WORKING RAM AREA
7A01 AISZ AC2,01
D200 :OVER ST AC2,ACO,00
F839 SKNE BP,AC2,CONST49 ;REACHED WORKING AREA?
19FB JMP PC,CLR
7A01 AISZ AC2,01
19FB JMP PC,OVER
1932 JMP PC,NORM

50EO :ERES6 LI ACO,EO ;DISPLAY "ERROR" ON LEDS
2810 SHL ACO,8,0
D301 ST AC3,ACO,01 ;FIRST ROW
5080 LI ACO,80
2810 SHL ACC,8,0
D302 ST AC3,ACO,02 ;SECOND ROW
D303 ST AC5,ACO,03 ;THIRD ROW
50BE LI ACO,EE
2810 SHL ACO,8,0
222

D304 ST AC3,ACO,04 ;FOURTH ROW
5089 LI ACO,89
2810 SHL ACO,8,0
782A AISZ ACO,2A
D305 ST AC3,ACO,05 ;FIFTH ROW
5089 LI ACO,E9
2810 SHL ACO,8,0
785A AISZ ACO,3A
D306 ST AC3,ACO,06 ;SIXTH ROW

155B JSR PC,HESG
2FD C/R ;PRINT "ERROR RESET LOGGED AT"
22D2 ER
294F RO
2920 RSP
2945 RE
29C5 SE
2A20 TSP
264F LO
23C7 GG
22C4 ED
1041 SPA
2A20 TSP
102E SP.

5303 :TIIMET LI AC3,03 ;PRINT REAL TIME LOG
52C3 LI AC2,03
C200 :DIGIT LD AC2,AC0,00 ;GET HOURS, ETC.
156F JSR PC,BCDMA ;CONVERT TO BCD
2408 ROR ACO,4,0
7830 AISZ ACO,30 ;ADD ASCII OFFSET TO MSD
155C JSR PC,CHOUT ;GOTO C/P SUBROUTINE
2C18 SHR ACO,12,0
7830 AISZ ACO,30 ;ADD ASCII OFFSET TO LSD
1559 JSR PC,CHOUT
7APD AISZ AC2,FD ;DEC POINTER
7BFF AISZ AC3,FF ;DEC COUNTER
1901 JMP PC,COLOA
1903 JMP PC,NORM
503A :COLONA LI AC0,3A ;PRINT ":"
1553 JSR PC,CHOUT
19F1 JMP PC,DIGIT
153C :NORM JSR PC,MESG
2F0D \tC/R ;PRINT "CPU1(2,3) OK"
21D0 CP
21B1 U1
104F SPO
25AE X.
3800 PFLG LINK
52FF :RAM LI AC2,FF ;SCRATCHPAD RAM TEST
2A10 SHL AC2,8,0
5155 LI AC1,55
2910 SHL AC1,8,0
7955 AISZ AC1,55 ;FORM TEST WORD
CE00 :NEXTA LD AC2,AC3,00 ;SAVE RAM LOCATION CONTENTS
D600 ST AC2,AC1,00 ;STORE PATTERN
C200 LD AC2,AC0,00 ;RELOAD PATTERN
2002 ROL AC0,1,0
D200 ST AC2,AC0,00 ;STORE ALTERNATE BIT PATTERN
C200 LD AC2,AC0,00 ;RELOAD PATTERN
2002 ROL AC0,1,0
5840 RXOR AC0,AC1 ;COMPARE WITH AC1
D800 ST AC2,AC3,00 ;RETURN OLD RAM CONTENTS
4515 BOC NREQO,ERRORA ;EXIT ON ERROR
7A01 AISZ AC2,01 ;INC POINTER
19F4 JMP PC,NEXTA
1524 :RAMOK JSR PC,MESG ;PRINT "RAM1(2,3) OK"
2F0D \tC/R ;PRINT "RAM1(2,3) OK"
2941 RA
26B1 MI
104F SPO
25AE K.
3800 PFLG LINK
224

5000 :CLED LI ACO,00 ;CLEAR LED DISPLAY
C009  LD BP,AC3,CONST1
D301  ST AC3,AC0,01
D302  ST AC3,AC0,02
D303  ST AC3,AC0,03
D304  ST AC3,AC0,04
D305  ST AC3,AC0,05
D306  ST AC3,AC0,06

5001 :RETB LI ACO,01
6000  PUSH ACO
5000  NOP
8000  RTS ;GOTO MAIN PROGRAM

1511 :ERRORA JSR PC,MESSG ;PRINT "RAM1(2,3) PAD FAIL"
2F0D  tC/R ;PRINT "SYSTEM SHUTDOWN"
2941  RA
26B1  MI
1050  SPP
20C4  AD
1046  SPF
20C9  AI
2620  LSP
29D9  SY
29D4  ST
22CD  EM
1053  SPS
2455  HU
2A44  TD
27D7  OW
272E  N.
19FF  :STOP7 JMP PC,STOP7

1E00  :MESSG XCHRS AC2 ;MESSAGE SUBROUTINE
C200  LD AC2,AC0,00 ;GET FIRST CHAR PAIR
240E  ROR ACO,7,0
A80F  AND BP,CONST7
F01D  SYNE BP,AC0,CONST21 ;CHAR = "."?
190C  JNP PC,END
F01E  SYNE BP,ACC,CONST22 ;CHAR = "*"?
6840  RADD  AC0,AC1
8000  RTS

.END
.TITLE SIMPLE CYCLE CHECK PROGRAM

(USES BASE PAGE CONSTANTS)

5080 :INIT LI ACO,80 ;INITIALIZE TELETYPewriter
B0CB ST BP,CONST3
3400 PFLG IE4
5CO0 NOP
50F9 LI ACO,F9
DOFB ST BP,ACO,SETCNT ;INITIALIZE DUMP POINTER
5003 LI ACO,03
DOFA ST BP,ACO,FAIL ;INITIALIZE FAIL COUNTER
3480 SPLG IE4 ;ENABLE L4 INTERRUPT
3580 SPLG IE5 ;ENABLE L5 INTERRUPT
5039 LI ACO,39
DOEC ST BP,ACO,FLAGS
5055 LI ACO,55
DOEB ST BP,ACO,PC

50CE :START LI ACO,CE ;SET UP IFLAG POINTER
DOFO ST BP,ACO,ADIPL
50CB LI ACO,CB
DOCF ST BP,ACO,ADOFL ;SET UP OFLAG POINTER
50CE LI ACO,C8
DOD1 ST BP,ACO,ADMFL ;SET UP MFLAG POINTER
50BC LI ACO,BC
DOD5 ST BP,ACO,ADSEC ;SET UP ADSEC POINTER
50BF LI ACO,BF
DOD4 ST BP,ACO,ADSEC ;SET UP ADSEC POINTER
3380 SPLG IE3
3980 SPLG IEN ;ENABLE MASTER INTERRUPT
5CC0 NOP
AOD5 :TIME LD@ BP,ADSFRC ;GET TIMER COUNTER
7856 AISZ ACO,F6 ;COUNT = 10?
4317 BOC MSIGN,OUT ;COUNT > 10?
5000 LI ACO,00
B0D5 ST@ BP,ADSFRC ;YES, CLEAR COUNT
AOD4 LD@ BP,ADSEC ;GET SECONDS COUNT
7801 AISZ ACO,01 ;INC SECONDS COUNT
B0D4    ST0    BP,ADSEC
78C4    AISZ   ACO,C4   ;COUNT = 60?
4B10    ECC    NSIGN,OUT  ;COUNT > 60?
5000    LI      ACO,CO
B0D4    ST0    BP,ADSEC  ;YES, CLEAR COUNT
A0D3    LD0    BP,ADMIN  ;GET MINUTES COUNT
7801    AISZ   ACO,O1   ;INC MINUTES COUNT
B0D3    ST0    BP,ADMIN
78C4    AISZ   ACO,C4   ;COUNT = 60?
4B09    ECC    NSIGN,OUT  ;COUNT > 60?
5000    LI      ACO,CO
B0D3    ST0    BP,ADMIN  ;YES, CLEAR COUNT
A0D2    LD0    BP,ADHRS  ;GET HOURS COUNT
7801    AISZ   ACO,O1   ;INC HOURS COUNT
B0D2    ST0    BP,ADHRS
7828    AISZ   ACO,E8   ;COUNT = 24?
4B02    ECC    NSIGN,OUT  ;COUNT > 24?
5000    LI      ACO,CO
B0D2    ST0    BP,ADHRS  ;YES, CLEAR COUNT
A0CF    :OUT    LD0    BP,ADOFL  ;LAST OUTPUT SUCCESSFUL?
45FE    ECC    NREQO,OUT  ;LOCIF IF NOT
A0D1    LD0    BP,ADMF  ;GET MESSAGE FLAG
4406    ECC    BIT1,ROWA  ;LED COUNT DISPLAYED LAST?
5002    LI      ACO,O2  ;YES, SELECT NEXT LED ROW
B0D1    ST0    BP,ADMF  ;SET UP MESSAGE FLAG
B0CF    ST0    BP,ADOF  ;SET UP OUTPUT FLAG
COA7    LD      BP,ACO,ERFLG  ;GET ERROR CYCLE COUNT
DODA    ST      BP,ACO,LOUT  ;SET LED OUTPUT BUFFER
19DC    JMP     PC,TIME

5001  :ROWA    LI      ACO,O1  ;SELECT FIRST LED ROW
B0D1    ST0    BP,ADMF  ;SET UP MESSAGE FLAG
B0CF    ST0    BP,ADOF  ;SET UP OUTPUT FLAG
7901    AISZ   AC1,O1  ;INC LED COUNT
D4DA    ST      BP,AC1,LOUT  SET LED OUTPUT BUFFER
19D6    JMP     PC,TIME
.END
.TITLE MULTI-TASK TACHOMETER PROGRAM    WGM 1980

(USES CHARACTER GENERATOR BEGINNING AT 0200)
(USES TWO-PHASE TACHOMETER INPUT)

9829  :INIT   JMP   PC,CHK   ;GOTO RESET CHECKOUT
1938   JMP   PC,TACH   ;GOTO MAIN PROGRAM
19FF   :STOP   JMP   PC,STOP   ;MPU FAILURE STOP

03E6   :L2V   (L2)   ;LEVEL 2 INTERRUPT VECTOR
B002   :L3V   (L3)   ;LEVEL 3 INTERRUPT VECTOR
0334   :L4V   (L4)   ;LEVEL 4 INTERRUPT VECTOR
0300   :L5V   (L5)   ;LEVEL 5 INTERRUPT VECTOR
0000   
0000   

(BASE PAGE CONSTANTS, COMMON TO ALL PROGRAMS)

DO00   (CONST1)
DO01   (CONST2)
DO08   (CONST3)
0040   (CONST4)
0080   (CONST5)
0100   (CONST6)
007F   (CONST7)
FFC6   (CONST8)
000A   (CONST9)
FFP3   (CONST10)
0010   (CONST11)
000D   (CONST12)
FFF6   (CONST13)
0041   (CONST14)
003A   (CONST15)
0044   (CONST16)
0049   (CONST17)
004D   (CONST18)
0050   (CONST19)
03PE   (CONST20)
002E   (CONST21)   ; "." 
005E   (CONST22)
230

B121 (INTEL)
B127 (OUTTEL)
B10B (MESS)
B131 (GHEX)
B141 (FHEX) B141
FFEA (CONST28)
0319 (REFIL)
FFFO (CONST30)
FFF9 (CONST31)
000F (CONST32)
0136 :CHK (CHECK)
003C (CONST34)
0018 (CONST35)
0054 (CONST36)
03ED (ERRMSG)
FFCC (CONST38)
0007 (CONST39)
FFF8 (CONST40)
FPDF (CONST41)
FFBC (SECS)
FEE6 (NEW+9)
FUNCTION (CONST49)
FFB9 (CONST49)

(THIS VERSION OF MTT USES VERSION 2 INTERRUPT SOFTWARE)

5080 :TACH LI ACO,80
B00B STB BP,CONST3 ;INITIALIZE TELETYPYE
C009 LD BP,AC3,CONST1
C30A LD AC3,AC0,CA ;READ ERROR REGISTER
B835 SKAZ BP,AC0,CONST45 ;ERROR RESET?
9629 JMP CHECK ;GOTO ERROR CHECKOUT
50F9 LI ACO,F9
DOFB ST BP,AC0,SETCNT ;INITIALIZE DUMP POINTER
5000 LI ACO,CO
40FA ST BP,AC0,RESET ;CLEAR RESET FLAG
231

53A4 LI AC3,A4
7B01 :CLOOP AISZ AC3,01 ;CLEAR SCRATCHPAD
D500 ST AC3,ACO,00
FC38 SKNE BP,AC3,CONST48
1901 JMP PC,START
19FB JMP PC,CLOOP

50CE :START LI ACO,CE
DOD0 ST BP,ACO,ADIFL ;SET UP IFLAG POINTER
50CB LI ACO,CB
DDCF ST BP,ACO,ADOFL ;SET UP OFLAG POINTER
50CC LI ACO,CC
DODI ST BP,ACO,ADMFL ;SET UP MFLAG POINTER
50CD LI ACO,CD
DOD5 ST BP,ACO,ADSPG ;SET UP SFLAG POINTER
50CE LI ACO,CE
DODF ST BP,ACO,ADSEC ;SET UP SECONDS POINTER
5000 NOP
3380 SFLG IE3
3980 SFLG IEN ;MASTER INTERRUPT ENABLE
3480 SFLG IE4 ;LEVEL 4 INTERRUPT ENABLE
3580 SFLG IE5 ;LEVEL 5 INTERRUPT ENABLE

1563 :LOOP JSR PC,DISFY ;GOTO SPEED DISPLAY ROUTINE
AOD5 LD@ BP,ADSPG ;GET TIMER COUNTER
700B CAI ACO,OB
4513 BOC NREQO,SKIP ;SFLG = 10?
BOD5 ST@ BP,ADSPG ;YES, CLEAR COUNT
AOD4 LD@ BP,ADSEC ;GET SECONDS COUNT
7801 AISZ ACO,01 ;INC SECONDS COUNT
BOD4 ST@ BP,ADSEC
703D CAI ACO,3D
450D BOC NREQO,SKIP ;SECONDS = 60?
BOD4 ST@ BP,ADSEC ;YES, CLEAR SECONDS COUNT
AOD3 LD@ BP,ADMIN ;GET MINUTES COUNT
7801 AISZ ACO,01 ;INC MINUTES COUNT
BOD3 ST@ BP,ADMIN
703D CAI ACO,3D
4507 BOC NREQO,SKIP ;MINUTES = 60?
BOD3 ST@ BP,ADMIN ;YES, CLEAR MINUTES COUNT
AOD2: LD# BP, ADHRS ;GET HOURS COUNT
7801: AISZ ACO, 01 ;INC HOURS COUNT
BOD2: ST# BP, ADHRS
7019: CAI ACO, 19 ;HOURS = 24?
4501: BOC NREQQ, SKIP ;YES, CLEAR HOURS COUNT
BOD2: ST# BP, ADHRS

A009: SKIP LD# BP, CONST1 ;GET TTY STATUS
A60C: AND BP, CONST4 ;TEST FOR KEYBOARD INPUT
4519: BOC NREQQ, SETTIME
AOCF: LD# BP, ADOFL ;GET O/P REQUEST FLAGS
45E4: BOC NREQQ, LOOP ;O/P REQUIRED/SUCCESSFUL?
AOD1: LD# BP, ADML ;GET MESSAGE FLAGS
2C06: SHR ACO, 3, 0
41E1: BOC REQQ, LOOP ;ANY FURTHER MESSAGES?
441E: BOC BIT1, TIME1 ;LAST O/P TTY ECHO?
4634: BOC BIT2, PRTIM1 ;PRINTING TIME?
C300: MESS LD AC3, ACO, 00 ;GET ERROR MESSAGE CHAR.
F01D: SKNE BP, ACO, CONST21 ;IS IT "."?
1906: JMP PC, END1
7B01: AISZ AC3, 01 ;INC POINTER

DOD6: OUT ST BP, ACO, TOUT ;SET UP TTY OUTPUT BUFFER
AOCF: LD# BP, ADOFL
A4CC: OR BP, CONST4
BOCF: ST# BP, ADOFL ;SET UP TTY O/P REQUEST FLAG
19D6: JMP PC, LOOP

AOD1: END1 LD# BP, ADMFL ;GET MESSAGE FLAGS
A82F: AND BP, CONST39 ;CLEAR TTY MESSAGE FLAGS
BOD1: ST# BP, ADMFL
19D2: JMP PC, LOOP

AOD1: ERROR LD# BP, ADMFL ;SET ERROR BIT IN MESSAGE FLAG
A435: OR BP, CONST45
BOD1: ST# BP, ADMFL ;GET ERROR MESSAGE POINTER
CC2D: LD BP, AC3, CONST37
19EE: JMP PC, FESS
AOD0 :SETIME LD# BP,ADIFL ;GET I/P REQUEST FLAGS
A436 OR BP,CONST46
B0D0 ST# BP,ADIFL ;SET TTY I/P REQUEST FLAG
AOD0 :DIN LD# BP,ADIFL
45FE BOC NREQO,DIN ;WAIT FOR TTY DATA INPUT
AOD1 LD# BP,ADMFL
A413 OR BP,CONST11
B0D1 ST# BP,ADMFL ;SET ECHO BIT
507F LI ACO,7F
A6D8 AND BP,TIN ;GET DATA FROM BUFFER
19E7 JMP PC,OUT ;ECHO DATA TO TTY
AOD1 :TIME1 LD# BP,ADMFL
A834 AND BP,CONST44
B0D1 ST# BP,ADMFL ;CLEAR ECHO FLAG BIT
C0D8 LD BP,ACO,TIN ;GET DATA FROM BUFFER
FC2C SKNE BP,ACO,CONST36 ;CHARACTER = "T"?
195A JMP PC,PRTIM ;GOTO TIME O/P ROUTINE
F011 SKNE BP,ACO,CONST9 ;CHARACTER = N/L?
1907 JMP PC,END2 ;PROCEED TO CONVERT
E010 ADD BP,ACO,CONST9 ;ADD 10
42E6 BCC FSIGN,ERROR
E011 ADD BP,ACO,CONST9 ;SUBTRACT 58
43E4 BCC NSIGN,ERROR
2B08 SHL AC3,4,0
68C0 RADD AC3,ACO ;FORM FULL DATA WORD
19B3 :EXT JMP PC,LOOP

1508 :END2 JSR PC,CONVD ;CONVERT INPUT ECD TO BINARY
B0D3 ST# BP,ADMIN ;SET MINUTES COUNT
2F10 SHR AC3,8,0
1505 JSR PC,CONVD ;CONVERT TO BINARY
B0D2 ST# BP,ADHRS ;SET HOURS COUNT
5000 LI ACO,00
B0D4 ST# BP,ADSEC ;CLEAR SECONDS COUNT
19F7 JMP PC,EXT
1955 :PRTIM1 JMP PC,PRTIM2
5000: CONVD

5400  RAND ACO,AC3

6000  PUSH ACO

5070  LI ACO,70

5400  RAND ACO,AC3

2C02  SHR ACO,1,0

5B00  RCPY AC2,AC0

5070  LI ACO,70

5400  RAND ACO,AC3

2C06  SHR ACO,3,0

6880  RADD ACO,AC2

6600  PULL AC2

6880  RADD ACO,AC2

8000  RTS

ACOF :DISPY

B82F  SKAZ BP,CON39

8000  RTS

AOD1  LD@ BP,ADMF1

7801  AISZ ACO,01

BOD1  ST@ BP,ADMF1

A82F  AND BP,CON39

7008  CAI ACO,08

4503  BCC NREQO,DISP1

AOD1  LD@ BP,ADMF1

A827  AND BP,CON31

BOD1  ST@ BP,ADMF1

COOF :DISP1

EXOD  ADD BP,ACO,CONST5

A8DB  AND BP,ACO,TACH

2C04  SHR ACO,2,0

154B  JSR PC,SCDH+1

510F  LI AC1,OF

5500  RAND AC1,ACO

2C08  SHR ACO,4,0

151D  JSR PC,CHAR

C20C  LD AC2,ACO,00

280A  SHL ACO,5,0

6000  PUSH ACO

5C40  RCPY ACO,AC1

A0CF :DISPY

LD@ BP,ADMF1

;SPEED DISPLAY SUBROUTINE

B82F  SKAZ BP,CON39

;LED ROW DISPLAYED YET?

8000  RTS

AOD1  LD@ BP,ADMF1

7801  AISZ ACO,01

BOD1  ST@ BP,ADMF1

A82F  AND BP,CON39

7008  CAI ACO,08

4503  BCC NREQO,DISP1

AOD1  LD@ BP,ADMF1

A827  AND BP,CON31

BOD1  ST@ BP,ADMF1

COOF :DISP1

LD BP,ACO,CONST7

EXOD  ADD BP,ACO,CONST5

A8DB  AND BP,ACO,TACH

;READ TACHOMETER BUFFER

2C04  SHR ACO,2,0

154B  JSR PC,SCDH+1

;CONVERT TO BCD

510F  LI AC1,OF

5500  RAND AC1,ACO

2C08  SHR ACO,4,0

151D  JSR PC,CHAR

C20C  LD AC2,ACO,00

280A  SHL ACO,5,0

6000  PUSH ACO

5C40  RCPY ACO,AC1
1518 JSR PC,CHAR ;GET DISPLAY ROW FOR LSD
6400 PULL ACC
A600 OR AC2,00 ;FORM PART DISPLAY ROW
6000 PUSH ACO

5225 :DIRN LI AC2,25 ;DIRECTION ARROW ROUTINE
2A06 SHL AC2,3,0 ;FORM BASE ADDRESS
A0D1 LD@ BP,ADNFL
A82F AND BP,CONST39
78FF AISZ ACC,PF ;SUBTRACT 1
6A00 RADD AC2,AC0
2A02 SHL AC2,1,0
4E01 BCC JC14,DIRN1 ;SENSE FORWARD OR REVERSE
7A01 AISZ AC2,01
6400 :DIRN1 PULL ACO
280E SHL ACO,7,0
A600 OR AC2,00
DODA ST BP,ACO,LOUT ;SEND ROW TO LED O/P BUFFER
A0D1 LD@ BP,ADMFL
A82F AND BP,CONST39
5800 RCPY AC2,AC0
A0CF LD@ BP,ADOFL
A830 AND BP,CONST40
6880 RADD AC0,AC2
B0CF ST@ BP,ADOFL ;SET UP OUTPUT FLAGWORD

5240 :CHAR LI AC2,40 ;ROW ADDRESS SUBROUTINE
6A00 RADD AC2,AC0
2A06 SHL AC2,3,0
A0D1 LD@ BP,ADMFL
A82F AND BP,CONST39
78FF AISZ AC0,PF ;SUBTRACT 1
6A0C RADD AC2,AC0
8000 RTS
5CC0 NOP
5CC0 NOP
AOD1 :PRTIMLD@BP,ADMFL ;TIME PRINTING ROUTINES
A437 OR BP,CONST47 ;SET TIME BIT IN MFLAG
A40C OR BP,CONST4 ;SET SPACE BIT IN MFLAG
BOD1 ST@BP,ADMFL
53C5 LI AC3,C5 ;SET AC3 TO (HRS)
5220 LI AC2,20 ;PRINT " "
D8D6 :PRTIM3STBP,AC2,TOUT ;SET UP TTY OUTPUT BUFFER
ACCF LD@BP,ADOFL
A40C OR BP,CONST4 ;SET UP FLAG FOR TTY O/P
B0CF ST@BP,ADOFL
19A1 JMP PC,EXT

AOD1 :PRTIM2LD@BP,ADMFL
2CCC SHR ACO,6,0 ;FINISHED PRINTING?
41CC BOC Req0,EXIT ;PRINT COLUMN NEXT?
4627 BOC BIT2,BCDL ;PRINT LSD NEXT?
441B BOC BIT1,BCDL ;PRINT MSN NEXT?
150E JSR PC,BCDM ;PRINT MSN
AOD1 LD@BP,ADMFL
A40D OR BP,CONST5
BOD1 ST@BP,ADMFL ;SET MESSAGE FLAG BIT
500F LI ACO,OF
5480 RAND ACO,AC2
DOD9 ST BP,ACO,TEMP
2808 SHR AC2,4,0
7A30 AISZ AC2,30 ;ADD ASCII CODE OFFSET
19BC JMP PC,PRTIM3
AOD1 :EXITLD@BP,ADMFL
A831 AND BP,CONST41
BOD1 ST@BP,ADMFL ;CLEAR TIME BIT IN MFLAG
520A LI AC2,CA ;PRINT N/L
1987 JMP PC,PRTIM3
C300 :BCDMLD AC3,ACO,00 ;BINARY - BCD SUBROUTINE
5200 LI AC2,00
3760 :BCD1 SFLC CRY
9011 SUBB BP,CONST9 ;SUBTRACT 10
4E02 BOC INSIGN,NEXTD
7A01 AISZ AC2,01
19FB JMP PC,BCDM1
2A00 :NXTD  SHL  AC2,4,0
780A   AISZ  ACO,OA  ;ADD 10
6A00   RADD  AC2,ACO
5C80   RCFY  ACO,AC2  ;TWO DIGIT BCD IN ACO & AC2
8000   RTS

C8D9 :ECDL  LD  BP,AC2,TEMP
7A30   AISZ  AC2,30  ;ADD ASCII CODE OFFSET
7BFD   AISZ  AC3,FD  ;DEC AC3 BY 3
51FF   LI  AC1,FF
A0D1   LD99  BP,ADMFL
A40E   OR  BP,CONST6
F032   SKNE  BP,AC3,CONST42  ;AC3 = FFEC?
513F   LI  AC1,3F
5440   RAND  ACO,AC1
B0D1   ST99  BP,ADMFL  ;SET UP FLAGWORD
19D0   JMP  PC,PRTIM3

A0D1 :COLON  LD99  BP,ADMFL
513F   LI  AC1,3F
5440   RAND  ACO,AC1
A40C   OR  BP,CONST4
B0D1   ST99  BP,ADMFL
523A   LI  AC2,3A  ;PRINT ":"
19C9   JMP  PC,PRTIM3

.END
.TITLE LEVEL 2 TIMER INTERRUPT HANDLER

(USED BY SIMPLE CYCLE CHECK & MTI)

3COO: :L2 PFLG F12 ;CLEAR INTERRUPT LATCH
3200 PFLG IE2 ;CLEAR INTERRUPT ENABLE
6300 PUSH AC3
6000 PUSH ACO

(NEXT THREE LINES APPLY TO STORE 1 PROGRAM ONLY)

CC09 LD BF,AC3,CONST1
C309 LD AC3,AC0,09 ;GET TACHO 100 HS COUNT
D0DB ST BP,AC0,TACHO ;DATA TO TACHO BUFFER

8CBE ISZ BP,SFLAG0 ;INC REAL-TIME CLOCK
8CBB ISZ BP,SFLAG1
8CBA ISZ BP,SFLAG2
6400 PULL ACO
6700 PULL AC3
7CC0 RTI
REFERENCES FOR CHAPTER 8


CHAPTER 9

AN EXAMINATION OF MODULE PERFORMANCE

9.1 Introduction

A realistic quantitative analysis of Cyclone I performance is extremely difficult, if not impossible to produce, especially if careful consideration is given to that part of the reliability definition which says, 'under stated conditions for a stated period of time'. The stated conditions would be those found in the cab of a BR electric locomotive and the interval would have to be several weeks or months at least. Unfortunately little data on locomotive electrical noise hazards exists, apart from that referenced in Chapter 4, and so laboratory work has involved using assumed worst-case conditions, (within reason). This chapter describes the prototype module performance under steady or normal operating conditions with no induced noise present, and again with power-rail noise generated artificially.

9.2 Operating with Fault-Free Conditions

The simple check program described in the last chapter was used to establish how system throughput is affected by the length of each program sector. At first glance, it may seem reasonable to assume that increasing the checking frequency by reducing the sector length would cause system throughput to fall. The graph of Fig. 9.1 indicates that in fact, the reverse is true. The explanation for this is quite simple and illustrates how Cyclone is better suited to control applications requiring frequent output and input from peripherals, than to low frequency output number-crunching.
Figure 9.1: Program output against sector length.
The simple check program requires output, in this case to the panel LED matrix, every system cycle. The amount of processing to be carried out in the main program is very small and so the MPU spends most of its time waiting for output. The system throughput is measured in terms of the counting rate displayed on the panel, and as the count is incremented on alternate cycles, the greater the cycle rate the faster the count. The graph shows a linear improvement in throughput with decreasing sector length to a bottom limit imposed by the length of the initialization routines. Another effect is that the three counts displayed, one for each program, remain in step in spite of the program priority previously discussed. This reflects the fact that the top priority program uses any bus access advantage in executing a few more wait loops per cycle than those of lower priority. The main advantage of increasing the checking frequency is of course that errors are given less time to propagate and are detected sooner, thus increasing the probability of successful corrective action. However, programs with a high processing to output ratio will suffer a considerable loss in execution speed, a sacrifice which may or may not be tolerable depending on specific requirements. To provide some reference figures, a seven line program was put into each main store, which runs in non-cycle mode with a binary count output to the LED matrix. The time for the count to reach FFFF from zero was recorded:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Processors</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Cycle</td>
<td>3</td>
<td>1.81</td>
</tr>
<tr>
<td>Calculated, no CPU</td>
<td></td>
<td>1.7</td>
</tr>
</tbody>
</table>

The calculated time is only slightly less than that measured, indicating that the estimate derived in Chapter 6 of a bus interference factor of \(0<0.1\) for these small programs is accurate. The system throughput is:

\[ T_s = 325860 \text{ instructions/sec} \]
This is of course throughput in terms of the amount of processing carried out, but if we return to Fig. 9.1 it can be seen that with the shortest sector length in full-cycle mode the time from zero to FFFF is:

- Cycle Mode, 3 processors : 380 secs

This figure allows for the count output being alternated with the error cycle count output on the display. The throughput remains the same as for non-cycle mode, but the output rate is drastically reduced. The continual output requests have absorbed a large proportion of the available CPU time indicating that Cyclone would be more efficient if given more data processing to perform before output. For optimum performance therefore, the time taken to execute the main loop within the application program must be determined, and the sector length set to be just a fraction longer.

In many control applications such a loop exists and the machine cycle can be tailored accordingly, reducing the unproductive wait-for-I/O time to a minimum. There will be an inevitable compromise between the three main program's requirements, and fine-tuning will need to be performed once the system is running.

The graph of Fig. 9.1 shows how alterations to the level 4 and 5 interrupt handlers have affected the output rate. The significant improvement from version 1 to version 2 is due to the introduction of the fast-exit system in the error checking routines allowing errors in sectors 0 or 1 without cycle loss. The drop in output rate from version 2 to 3 is due to the latter's ability to cope with total failure of a CPU.
9.3 Performance with Faults

9.3.1 CPU failure

The version 3 software allows graceful system degradation when a processor board is switched off or physically removed from the rack. The effect is to reduce system throughput by almost one third by losing one program, but to increase the output rate for the remaining two programs, bringing it back to that of version 2. The reduced bus conflict accounts for this improvement. At the moment, the basic module is vulnerable to hard faults in components other than the MPUs, but the design philosophy generally ensures that the system meets fail-safe requirements.

9.3.2 Transient faults - the interference box

In order to study the effects of power supply interruptions and noise spikes, an interference generator (Plate 4) was built and wired in series with the +5V line to the Cyclone module. The circuit diagram is given in Fig. 9.2. A master oscillator formed from an NE555 chip in astable mode, produces a square wave variable in frequency from about 3 Hz to 80 Hz. This output is either fed directly to a variable pulse-width monostable or via a 15 bit pseudo-random pulse generator according to a panel switch setting. The monostable produces pulses which may be varied in width from about 30 μs to 300 μs. This output then drives a heavy current output stage inserted in the main power rail. CMOS and bipolar devices have been used to allow the unit to run off the main power supply input which is set at about 7V to compensate for the $V_{CE}$ drop.

9.3.3 Cyclone response to power supply transients

The specific effects of power supply transients on the machine are unpredictable, but the use of the interference box has allowed a list of
FIG. 9.2 POWER SUPPLY INTERFERENCE CIRCUIT
visible responses to be drawn up. As a result modifications to the software have been made to cope with some but not all of them. Detected transient errors lead to three main classes of module response: Full recovery, recovery with reduced main program throughput, and no recovery. Taking the full recovery situation first, this can be brought about from the following:-

- Transient 'hits' while system is running in sector 0 or 1. Usually no error cycle is produced while using version 2 or 3 software.

- Transient hits in sector 2. This almost invariably leads to an aborted cycle and retry.

- Transient corrupts a non-critical location or group of locations in scratchpad RAM. Error cycle causes recovery of correct data and locations are reset to correct values.

Tests have shown that the bulk of transient effects fall into these latter three categories until the sector length is reduced to the order of 1 ms. After this point the probability of critical corruption of data movements in the level 4 handler increases greatly. None of these recovery modes involves program re-initialisation, but sometimes RAM corruption causes six successive error cycles resulting in clearance of all non-main program variables and program restart. Hence we have two more full recovery conditions:-

- Critical RAM is corrupted so that normal error recovery cannot cope. Program is restarted, but with original variables.

- Critical RAM is corrupted such that all main program variables need to be cleared as well, before program restart.
These latter two recovery modes are required in about 10% of transient
fault cases. Note that so far only affected programs suffer any rollback
or restart action. Single transients rarely affect all three programs
simultaneously.

Occasionally a transient will cause a more serious stoppage:-

- Clearance of all main program variables fails to produce
  recovery so the program is aborted by the watchdog timer.

- A transient clears interrupt enables in the MPU or causes loss
  of interrupt resulting in watchdog time-out.

In these situations all programs are restarted after successful completion
of an MPU and RAM checkout routine. Recovery should be total, but time
is lost while the checkout takes place

So far, transients have done nothing more than 'glitch' the processing
flow briefly with little or no evidence of its happening after recovery.
However, the next class of recovery conditions observed can be termed
partial recovery, because error cycles are still produced long after the
transient has disappeared.

- RAM corruption leads to 'bursts' of error cycles, each burst being
  less than the six necessary to bring about the store clearance
  mentioned before.

Whether or not this situation is acceptable depends on the processing
throughput requirement. Generally, processing is carried out correctly
and output allowed, but at a lower rate than normal, it has been observed
that further transients quite often cause the retry boundary to be exceeded,
forcing restart and subsequent full recovery.

Finally, we have the no-recovery category where all program output
ceases because of continual error cycles. Two modes have been identified:-
• Processor is trapped in a small closed program loop, unable to exit, and cycling continues normally with none of the error detection facilities aware of the problem.

• Processor is trapped in a similar loop, but this time continually attempts to restart the main program, but without success.

Both these modes involve normal level 4 and 5 operation, except that level 4 each time refuses to allow any output as it considered the sector results incorrect each time. These program 'stall' conditions as before may be limited to one or two programs and also as before, another subsequent transient may cause restart.

All these tests were conducted with a glitch width of 30 μs initially, which yielded a rail drop of at least 1.5 - 2V from the nominal 5V at the individual device pins. On board decoupling capacitors consisting of four 0.1 µF disc ceramics and one 10 µF electrolytic per board absorb some of the transient impact as does the effect of 'auto-decoupling' described by Catt et al (1). Glitches of less than 10 µS duration generally pass by unnoticed by the module while those interruptions beyond 100 µS tend to generate non-recoverable faults. The level of supply interruptions is considered to be severe, particularly as the effect is introduced between the power supply and the module. In practice, the supply filters should stop the bulk of these from reaching the system.

9.4 Conclusion

Experiments with the interference box have shown that even the basic laboratory model of Cyclone I can deal with severe power supply corruption and usually avoids incorrect output conditions. Generally, the scratchpad RAM is the area of circuitry most likely to latch-in and render transient
faults semi-permanent. With this in mind every effort should be made to protect at least this section of the system from power loss of any duration while the rest can be left with minimal hardware protection. Hence heavy reliance is placed upon the software checking and recovery routines to effect a return to normal working. Cyclone software has been developed to a point where it can deal with most of first recovery situations described but fails on some of the latter ones. This deficiency should not prove too difficult to overcome given a closer analysis of the precise failure modes.

As for normal system throughput, the bus structure has shown itself to be ideally suited for handling the three microprocessors, and the theoretical evaluation obtained earlier concerning bus requirements appears to be correct. Programs such as the Multi-Task Tachometer revealed that given more actual number-crunching to do, the system efficiency is rather better than that suggested by the simple cycle check program. For comparison purposes, the reader is directed to a description of a time-sliced processor system also based on the PACE MPU(2).
REFERENCES FOR CHAPTER 9


CHAPTER 10

CONCLUSIONS, COMMENTS AND SUGGESTIONS FOR FURTHER WORK

10.1 Conclusions and Comments

The reader is invited to recall the introduction to this thesis where it was stated that research would be conducted from an engineering viewpoint rather than from one which was purely academic in nature. An examination of the reference lists will show that work has focussed on practical realisations of fault-tolerant computing ideas as well as the theoretical bases behind them. In this aspect, the experience gained while working at British Rail, Derby, proved invaluable in bringing about an understanding of the real-world problems encountered by the railway engineers.

The preliminary work consisted of studying the various theoretical structures of high-reliability and fault-tolerant machines. At this stage, the various blocks making up a system do not necessarily involve the so-called high-technology of microprocessors, but the summary presented in Chapter 2 allows the reader to evaluate the relative probabilities of correct operation in different basic designs. Even without complete knowledge of the practical problems involved, clear pointers towards some kind of TMR system emerge.

A survey of many different types and designs of both laboratory prototype and applied systems was carried out, taking in a very wide range from telephone exchanges to space shuttle computers. The movement towards distributed processing and away from fault-avoiding uniprocessing with the advent of large scale integrated circuits was illustrated by
the examples discussed. The use of TMR in its various forms figures largely in many of the designs. However, before consideration was given to a new train control design, a study of the particular hazards encountered by train-borne electronics, together with operational requirements was carried out.

The practical problems facing the design engineer were discussed in Chapter 4. The magnitude of the electrical noise present proves to be one of the toughest problems to solve, bearing in mind that any new equipment will be fitted to railway locomotives with very little existing noise shielding or suppression. The kind of performance expected of a train control computer was examined in addition to aspects of safety and reliability, and the conclusion reached that multi-microprocessing should be used. Basic control functions of automatic driving, speed and safety supervision, and tachometer processing were identified. These individual functions could be translated into software for three separate loosely-coupled microcomputer systems. At this stage, these processors were in 'block' form only, and their precise form yet to be determined by safety constraints.

The detail design for an automatic train control system devised at British Rail, Derby by the BRATO project team was looked at in Chapter 5. The TMR approach to reliability was used extensively for two of the functions described, but not for the autodriver as it was felt that the speed supervisor would prevent safety hazards, and triplication of the LSIIl processor would be prohibitively expensive to implement. Unfortunately, no firm conclusions can be drawn as to the systems ability to sustain reliable operation in service, due to the project's demise just prior to pilot trials. However it is hoped that the experience gained in some of the early trials testing brake algorithms, etc., may
be used in the future. Certainly the writer acquired valuable knowledge while building the train simulator for BR and while taking part in some of the test train operations.

The design for the Cyclone system stemmed from a desire to investigate the use of time redundancy techniques coupled with multiprocessing. The aim was to make use of spare processor time to reduce the amount of hardware needed to achieve the required reliability. It was suggested that the processing throughput necessary to implement the control algorithms could be obtained by using 16 bit microprocessors instead of the 8 bit types used in two thirds of the BRATO system. Chapter 6 described how the initial idea was developed and prototype hardware constructed. At this stage, the considerable saving in potentially unreliable LSI hardware became readily apparent. However, the question of bus conflicts causing excessive time wastage and maybe even 'lock-ups', required extensive soak testing to answer.

Software development proceeded alongside hardware design as the two are closely linked in Cyclone. Chapters 7 and 8 described in detail the current state of both, concluding that the programmer would need to obey some rules regarding use of scratchpad memory and input/output. Total transparency of the error checking software to the applications programmer was found to be impracticable with the current hardware, but the conclusion was drawn that the extra overhead in understanding required was within acceptable limits.

The analysis of Cyclone performance in Chapter 9 demonstrated how the applications programmer must set the sector length to optimise system throughput. Performance relative to unchecked non-cyclic operation was found to be considerably reduced when running the simple cycle check
program, but the conclusion was reached that more complex programs with a greater ratio of processing to output would show much improved efficiency. The ability of Cyclone to cope with induced transient faults was demonstrated by using a power supply interference generator. The various failure modes were discussed, an examination of which showed that the existing hardware/software could deal with or recover from most transients.

Cyclone I cannot recover from most permanent or hard faults, but will at least shutdown safely in single hard failure tolerance, a duplex arrangement was suggested, with simple output switching. The checking system in Cyclone would allow it to operate in the simplex mode if one system module failed, thus greatly improving the theoretical reliability for a duplex system. Additional security is provided by allowing both modules independent control of an emergency brake. The work described in this thesis is seen as a fitting adjunct to that carried out by Milroy\(^1\) and Thomas\(^2\) on railway control algorithms and techniques. The Cyclone study has brought together ideas for reliable hardware engineering which should allow sophisticated control software to perform correctly and safely. Without a guarantee of safe operation all the work carried out on efficient train operation, maximising fuel economy, optimum time-tabling, etc., will be wasted as one serious accident under automatic control would badly damage public confidence.

10.2 Suggestions For Further Work

Further work on the Cyclone I prototype should involve improving and streamlining the error checking software. At present there are several known inefficiencies in the basic format of version 3, particularly in scratchpad.
memory usage. Eliminating these and other problems will improve overall system throughput and reduce the time spent in level 4.

The first prototype was intended for laboratory work only and is in no way engineered to take the rigours of a real train cab. Hence development of system hardware should continue with the construction of two Cyclone modules which may be subjected to real train conditions. The design of the duplex system deserves close attention particularly in the area of the failure sensing and switching. When constructed, train control software incorporating the algorithms developed at LUT and BR can be evaluated and performance studied when worked in conjunction with the error-checking system. At this point the precise levels of interference and noise that Cyclone will tolerate before train operation is impaired can be determined.

There is no reason why the Cyclone principle should be limited to train control applications, and an alternative future for the design may well lie in other mobile control fields where very high processing throughputs are not required. Having said that, a new generation of powerful 16 and 32 bit microprocessors is becoming available at the time of writing (1981) and a preliminary study suggests that one or more may be suitable as a replacement for the 8900 used in Cyclone I. An updated Cyclone II could then be built with much improved throughput, but still with the same number of LSI devices. The new microprocessors and memory devices, with their very high levels of integration are likely to suffer more than their predecessors with transient or soft errors due to radiation effects (Chapter 3) and hence the special ability of the Cyclone system could go a long way to ensuring fault-free outputs at the computer-outside world interface.

Finally, the continued research in high-reliability computer operation may provide an answer to the dismal but unfortunately often accurate quotes which preface this thesis.
REFERENCES FOR CHAPTER 10

1. MILROY, I.P.: "Aspects of Automatic Train Control",

2. THOMAS, P.D.: "Longitudinal Control for Guided Transport",
PLATE 1. THE CYCLONE I PROTOTYPE SYSTEM

PLATE 2. CYCLONE I PROTOTYPE MODULE
PLATE 3. THE MAG TAPE UNIT

PLATE 4. THE POWER INTERFERENCE BOX
APPENDIX I

UTILITY SOFTWARE

Using the TU58 Magnetic Tape Cartridge System

Handler Routines begin at Address: B150

From Power ON:-

1. Switch "MODE" knob to 1
2. Place "MONITOR INITIALISE" Tape in PTR
3. Press "LOAD" (LD)
4. Repeat steps 2 and 3 for MPUs 2 and 3
5. Press "START" (ST)
6. Press "MONITOR" (MO)
7. Set WG keys to B150
8. Press "JUMP" (JP)
9. Press "MONITOR INHIBIT" (IT)

Handler should ring terminal bell and prompt with:-

READY
+

Valid Commands are:-

N  Fast Forwards tape to the end and then rewinds to retension the tape
S  Triggers TU58 Self-Test
R  Enter data read mode. Handler responds with requests for the starting block no. (0-511), starting address for loading (in hex) and number of words to be loaded (in decimal)
W  Enter data dump mode. Handler requests information as for data read, but also asks if verification is required.
Go to the hex editor routines

TU58 Fail Codes:

1  Failed Self Test
2  Partial Operation (End of Tape)
6  Hard Read Error (Tape Damage)
8  Invalid Unit No.
9  No Cartridge
11 Write Protect On
17 Data Check Error
32 Seek Error (Block not found)
33 Motor Stopped
55 Invalid Block No.

One Cartridge holds 512 blocks of 512 bytes each.

Using the Hexadecimal Program Editor

Editor routines begin at Address: BOAO

Entry as for TU58, except WG key address is BOAO

Editor should prompt with:

> 

Valid commands are:

P XXXX <N/L>  Prints data in hexadecimal from the specified address.
              Press N/L to stop.
M XXXX <N/L>  Allows data at specified location to be changed.
I XXXX <N/L>  Inserts data at specified address, moving following
              data up.
D XXXX <N/L>  Deletes data from specified address, moving following
              data down.
.TITLE HEXADECIMAL PROGRAMME EDITOR WCM 1979

(COMMANDS: P(PRINT), M(MODIFY), I(INSERT), D(DELETE))
(USES BASE PAGE CONSTANTS)

5000:COMLP LI ACO,00
DOFD ST BP,ACO,FLAG
500A LI ACO,CA
9420 JSR@ BP,OUTEL ;PRINT L/F
500D LI ACO,CD
9420 JSR@ BP,OUTEL ;PRINT C/R
503E LI ACO,3E
9420 JSR@ BP,OUTEL ;PRINT " > "
941F JSR@ BP,INTEL ;GET COMMAND
F018 SKNE BP,ACO,CONST16 ;COMMAND D?
1958 JMP PC,DEL
F019 SKNE BP,ACO,CONST17 ;COMMAND I?
192C JMP PC,COM
F01A SKNE BP,ACO,CONST18 ;COMMAND M?
192C JMP PC,CON+2
F01E SKNE BP,ACO,CONST19 ;COMMAND P?
1908 JMP PC,PRINT
F011 SKNE BP,ACO,CONST9 ;N/L?
198D JMP PC,COMLP
1557:ERROR1 JSR PC,MESS ;PRINT " ERROR"
1045 SPE ;PRINT " ERROR"
2952 RR
27D2 OR
2P2E ↩
19B7 JMP PC,COMLP
1577:PRINT JSR PC,GHEX ;GET ADDRESS
19F8 JMP PC,ERROR1 ;ERROR RETURN
500D LI ACO,0D
156A JSR PC,OUTEL ;PRINT C/R
5008 LI ACO,08
DOFE ST BP,ACO,BCOUNT
5028 LI ACO,28
1566 JSR PC,OUTEL ;PRINT "(" 
5C80 RCPY ACO,AC2
262

157E  JSR   PC,PHEX                  ;PRINT ADDRESS
5029  LI    ACO,29
1562  JSR   PC,OUTEL                ;PRINT "")"
5020  :PRINTA LI    ACO,20
1560  JSR   PC,OUTEL                ;PRINT " "
C200  LD    AC2,ACO,00                ;GET DATA
6EC0  RXCH  AC2,AC3
1577  JSR   PC,PHEX
6EC0  RXCH  AC2,AC3
COPD  LD    BP,ACO,FLAG
4310  BOC   BITO,MOD                ;IN MODIFY ROUTINE?
441D  BOC   BIT1,INS                ;IN INSERT ROUTINE?
A009  LD@   BP,CONST1               ;PRINT ROUTINE
A80C  AND   BP,CONST4
45D7  BOC   NREQO,COMLP+8           ;KEYBOARD BREAK?
7A01  AISZ  AC2,01                  ;SKIP IF END OF STORE'
1901  JMP   PC,CONT
19CC  JMP   PC,COMLP
ACFE  :CONT  DSZ   BP,BCOUNT       ;END OF PRINT LINE?
19EF  JMP   PC,PRINTA
500A  LI    ACO,OA
154F  JSR   PC,OUTEL                ;PRINT L/F
19E2  JMP   PC,PRINT+2
5C00  :COM  NOP
8CFD  ISZ   BP,FLAG
8CFD  ISZ   BP,FLAG
19DC  JMP   PC,PRINT
5020  :MOD  LI    ACO,20            ;PRINT " "
1548  JSR   PC,OUTEL                ;GET CHARACTER?
1541  JSR   PC,INTEL
F011  SKNE  BP,ACO,CONST9           ;N/L INPUT?
19BE  JMP   PC,COMLP
6EC0  RXCH  AC2,AC3
5200  LI    AC2,00
1550  JSR   PC,CHEX+4                ;GET NEW DATA
19CD  JMP   PC,ERROR+1              ;ERROR RETURN
6EC0  RXCH  AC2,AC3
DECO ST AC2,AC3,00 ;STORE NEW DATA
7A01 AISZ AC2,01
19D1 JMP PC,PRINT+2
19B5 JMP PC,COMLP
5020 INS LI AC0,20 ;INSERT ROUTINE
153A JSR PC,OUTEL ;PRINT " "
1533 JSR PC,INTEL ;GET CHARACTER
F011 SKNE BF,AC0,CONST9 ;N/L INPUT?
19B0 JMP PC,COMLP
6E00 RXCH AC2,AC3
5200 LI AC2,00
1542 JSR PC,GHEX+4 ;GET DATA
19BF JMP PC,ERROR1 ;ERROR RETURN
6E00 RXCH AC2,AC3
5DC0 RCPY AC1,AC3
C01C LD BP,AC0,CONST20 ;GET STORE SIZE
D0FC ST BP,AC0,DCOUNT
A0FC RET LD@ BP,DCOUNT
CCFC LD BP,AC3,DCOUNT
D101 ST AC3,AC0,01
F8FC SKNE BP,AC2,DCOUNT ;FINISHED BLOCK MOVE?
1902 JMP PC,NEXT
ACFC DSZ BP,DCOUNT
19F9 JMP PC,RET
D600 NEXT ST AC2,AC1,00 ;STORE NEW DATA
7A01 AISZ AC2,01
19B9 JMP PC,PRINT+2
199D JMP PC,COMLP
152D DEL JSR PC,GHEX ;DELETE ROUTINE
19AE JMP PC,ERROR1 ;ERROR RETURN
C201 LD AC2,AC0,01
D200 ST AC2,AC0,00
F81C SKNE BP,AC2,CONST20 ;FINISHED BLOCK MOVE?
1997 JMP PC,COMLP
7A01 AISZ AC2,01
19FA JMP PC,DEL+2
264

1EOO :MESS XCHRS AC2 ;MESSAGE SUBROUTINE
C200 LD AC2,AC0,00
240E ROR ACO,7,0
A80F AND BP,CONST7
F01D SKNE BP,AC0,CONST21 ;IS IT ","?
190C JMP PC,END
F01E SKNE BP,AC0,CONST22 ;IS IT "+"?
500A LI ACO,CA
1503 JSR PC,OUTEL ;PRINT CHARACTER
C200 LD AC2,AC0,00
A80F AND BP,CONST7
F01D SKNE BP,AC0,CONST21 ;IS IT ","?
1905 JMP PC,END
F01E SKNE BP,AC0,CONST22 ;IS IT "+"?
500A LI ACO,CA
150C JSR PC,OUTEL ;PRINT CHARACTER
7A01 AISZ AC2,01
19BF JMP PC,MESS+1
7A01 :END AISZ AC2,01
1EOO XCHRS AC2
8000 RTS
5000 NOP

5000 :INTEL NOP ;TTY I/O SUBROUTINES
5000 NOP ;TTY INPUT SUBROUTINE
A009 :LOOP LD@ BP,CONST1
A80C AND BP,CONST4
41FD BOC REQ0,LOOP ;DATA AVAILABLE?
AC0B LD@ BP,CONST3 ;YES, GET DATA
5D00 :OUTEL RCPY AC1,AC0 ;TTY OUTPUT SUBROUTINE
A009 :LOOP1 LD@ BP,CONST1
A80D AND BP,CONST5 ;GET TTY STATUS
41FD BOC REQ0,LOOP1 ;PRINTER READY?
5C40 RCPY ACO,AC1
A40E OR BP,CONST6 ;ADD TTY SELECT BIT TO DATA
8003 ST@ BP,CONST3 ;SEND DATA TO PRINTER
A80F AND BP,CONST7
BO0A ST@ BP,CONST2 ;DISPLAY CHAR ON LEDS
8000 RTS
265

5200 : GHEX LI AC2,00 ; HEX INPUT SUBROUTINE
15EE JSR PC, INTEL ; GET CHARACTER
F011 SKNE BP, ACO, CONST9 ; IS IT N/L?
8001 RTS+1 ; NORMAL RETURN
E010 ADD BP, ACC, CONST9 ; SUBTRACT 58
4203 BOC PSIGN, HEX
E011 ADD BP, ACO, CONST9 ; ADD 10
4205 BOC PSIGN, FOUND
8000 : ERROR RTS ; ERROR RETURN
E012 : HEX ADD BP, ACO, CONST10 ; SUBTRACT 13
42FD BOC PSIGN, ERROR
E013 ADD BP, ACO, CONST11 ; ADD 16
4BF8 BOC NSIGN, ERROR
2A08 : FOUND SHL AC2, 4, 0
6A00 RADD AC2, ACO
19F1 JMP FC, GHEX+1

5204 : PHEX LI AC2, 04 ; HEX OUTPUT SUBROUTINE
D8FF ST BP, AC2, ACOUNT
5E60 RCPY AC2, ACO
2208 : PHEX1 ROL AC2, 4, 0
500F LI ACO, OF
5480 RADD ACO, AC2
E015 ADD BP, ACO, CONST13 ; SUBTRACT 10
4B05 BOC NSIGN, DEC
E016 ADD BP, ACO, CONST14 ; ADD 65
15DC : PHEX2 JSR PC, OUTEL ; PRINT DIGIT
ACPP DSZ BP, ACOUNT
19F7 JMP PC, PHEX1
8000 RTS
E017 : DEC ADD BP, ACO, CONST15 ; ADD 58
19FA JMP PC, PHEX2

END
.TITLE - DEC TU58 MAC TAPE HANDLER  WGM 1980

5000  :INIT  LI  ACO,00
DOFF  ST  BP,ACO,COMBUF  ;CLEAR COMMAND BUFFER
5070  LI  ACO,70
2808  SHL  ACO,4,0
CC09  LD  BP,AC3,CONST1
D30A  ST  AC3,ACO,TAPCOM  ;SET UP TAPE UART
950C  JSR@  PC,CASOUT  ;SEND BREAK
50FF  LI  ACO,FF
D30A  ST  AC3,ACO,TAPCOM  ;SET UP TAPE UART
5004  LI  ACO,04
9508  JSR@  PC,CASOUT  ;SEND INIT
9507  JSR@  PC,CASOUT  ;SEND INIT
9507  JSR@  PC,CASIN  ;GET REPLY
2008  SHR  ACO,4,0
4307  BOC  BITO,READY  ;OK?
5007  :SELF  LI  ACO,07  ;SELF-TEST ROUTINE
DOFF  ST  BP,ACO,COMBUF
9503  JSR@  PC,COMPAC  ;SEND COMMAND PACKET

B25D  (CASOUT)
B257  (CASIN)
B264  (COMPAC)

5000  :READY  LI  ACO,00
DOFF  ST  BP,ACO,COMBUF  ;CLEAR COMMAND BUFFER
9421  JSR@  BP,MESS
2POD  *C/R  ;PRINT "READY" & RING BELL
2945  RE
20C4  AD
2CDE  Y+  
06AB  C/R+  
03AE  BELL.
941F  JSR@  BP,INTEL  ;INPUT FROM KEYBOARD
F16C  SKNE  PC,ACO,CONSTA  ;S(=SELF-TEST) INPUT?
19ED  JMP  PC,SELF
F16B  SKNE  PC,ACO,CONSTB  ;N(=NEW TAPE) INPUT?
1978  JMP  PC,NEW
F16A  SKNE  PC,ACO,CONSTC  ;W(=WRITE) INPUT?
190E    JMP    PC,WRITE1
F169    SKNE   PC,ACO,CONSTD ;R(=READ) INPUT?
190A    JMP    PC,READ1
F168    SKNE   PC,ACO,CONSTE ;E(=EXIT) INPUT?
996B    JMP@   PC,EDIT
F011    SKNE   BP,ACO,CONST9 ;N/L INPUT?
19EA    :READY2 JMP    PC,READY

9421    :ERROR1 JSR2   BP,MES
1045    SPE                  ;PRINT "ERROR"
2952    RR
27D2    OR
2F2E    ↑
19E4    JMP    PC,READY

5002    :READ1 LI     ACO,02 ;SELECT READ OPCODE
1901    JMP    PC,SFLAG
5003    :WRITE1 LI     ACO,03 ;SELECT WRITE OPCODE
DOFF    :SFLAG   ST     BP,ACO,COMBUF
9421    JSR@   BP,MES
2FCD    ↑C/R                  ;PRINT "START BLOCK NO?"
29D4    ST
20D2    AR
2A20    TSP
214C    BL
27C3    OC
25A0    KSP
274F    NO
1FA0    ?SP
1700    .
9557    JSR@   PC,GBCD       ;INPUT BLOCK NO.
19E9    JMP    PC,ERROR1
DOFE    ST     BP,ACO,BLKN80 ;SET UP BLOCK NO. BUFFER
9421    JSR@   BP,MES
2352    FR                  ;PRINT "FROM?"
27CD    QM
1FA0    ?SP
1700    .
9422    JSR@   BP,GHEX       ;INPUT START ADDRESS
JMP PC, ERROR1
LI ACO, OD
JSR@ BP, OUTEL ; SEND C/R TO PRINTER
ST BP, AC2, STADD ; SET UP START ADDRESS BUFFER
LD BP, ACO, COMBUF
BCC BITO, WRITE2 ; READ OR WRITE?
JSR@ BP, MESS ; PRINT "NO OF WORDS?"

LD BP, ACO, COMBUF
JSR@ BP, MESS ; PRINT "READING"
LD BP, ACO, RECL
JSR@ PC, GSSP ; INPUT WORD COUNT
JSR@ PC, COMPAC ; SEND COMMAND PACKET
JSR@ PC, ENDFAC ; GET END PACKET
BOC REQO, ENDX ; TEST FOR END PACKET OR DATA

RCPY AC2, ACO
JSR@ PC, CASIN ; SEND COMMAND PACKET
RCPY AC1, ACO ; BLOCK BYTE COUNT IN AC1
JSR@ PC, CASIN
ST BP, AC2, BYTE ; TEMP SAVE FIRST BYTE
JSR@ PC, CASIN
RCY AC2, ACO
JSR@ PC, CASIN
AND BP, BYTE ; FORM DATA WORD
BOFD ST@ BP,STADD ;STORE DATA WORD
3700 PFLG CRY
6A00 RADD AC2,ACO
5000 LI AC0,00
7600 RADC AC2,ACO ;FORM SUMCHECK IN AC2
80FD ISZ BP,STADD ;INC STORE ADDRESS POINTER
79FE A1SZ AC1,FE ;DEC BYTE COUNT BY 2
19F3 JMP PC,READ3

951E :ENDA JSR@ PC,CASIN
A919 AND PC,CONSTG
5D00 RCPY AC1,ACO
951B JSR@ PC,CASIN
2810 SHL AC0,8,0
6840 RADD AC0,AC1 ;TAPE SUMCHECK IN AC0
5880 RXOR AC0,AC2 ;COMPARE SUMCHECKS
45AC BOC NREQO,ERROR1 ;EXIT IF ERROR
19E2 JMP PC,READ4

9515 :END JSR@ PC,CASIN
9511 :ENDE JSR@ PC,ENDPAC ;GET END PACKET
BCFA :ENDX LSEX BP,SUCCESSDE ;GET SUCCESS CODE WORD
4507 BOC NREQO,ERROR2 ;SUCCESS CODE = 0 ?
9421 JSR@ BP,MESS ;YES, TRANSFER OK
21CF CO ;PRINT "COMPLETE"
26DO MP
2645 LE
2A45 TE
1700 .
199F :READY1 JMP PC,READY2

950E :ERROR2 JSR@ PC,ERRMSG

0053 (CONSTA)
004E (CONSTB)
0057 (CONSTC)
0052 (CONSTD)
0045 (CONSTE)
0059 (CONSTF)
00FF (CONSTG)
NEW CARTRIDGE ROUTINE

SET UP COMMAND WORD

SET BLOCK NO. TO 511

WIND FORWARD TO END

SET BLOCK NO. TO 0

FULLY REWIND

PRINT "TO?"

INPUT END ADDRESS

PRINT C/R

PRINT "VERIFY (Y/N)?"

INPUT FROM KEYBOARD

VERIFY REQUIRED?
1903    JMP     PC,NOR1
5008    LI      ACO,08
A4FF    OR      BP,COMBUF
DOFF    ST      BP,ACO,COMBUF ;SET VERIFY FLAG
5C80    RCPY    ACO,AC2
3780    SFLG    CRY
90FD    SUBB    BP,STADD
2802    SHL     ACO,1,0 ;CALCULATE RECORD LENGTH
DOFC    ST      BP,ACO,RECLEN
9421    JSR@    BP,MESS
2FOD    tC/R    ;PRINT "FILE 
2349    FI      
2645    LE      
102E    SP.     
COPC    LD      BP,ACO,RECLEN
95D0    JSR@    PC,PBCD ;PRINT FILE LENGTH
9421    JSR@    BP,MESS
1042    SPB     ;PRINT "BYTES"
2CD4    YT      
22D3    ES      
2FOD    tC/R    ;PRINT "PROCEED (Y/N)?"
2852    PR      
27C3    CC      
22C5    EE      
2228    D(      
2CAF    Y/      
2729    N)      
1FA0    ?SP     
1700    .
941F    JSR@    BP,INTEL ;INPUT FROM KEYBOARD
F1B5    SKNE    PC,ACO,CONSTB ;OK TO DUMP?
19B1    JMP     PC,READY1
9421    JSR@    BP,MESS
2FOD    tC/R    ;PRINT "WRITING"
2BD2    WR      
24D4    IT      
24CE    IN      
23AO    CSP     
1700    .
272

1531 JSR  PC,CONPAC  "SEND COMMAND PACKET"
1558 JSR  PC,ENDPAC  "WAIT FOR END PACKET"
4501 BOC  NREQO,WRITE4  "MORE DATA PACKETS?"
199E JMP  PC,ENDX  "NO, END OR ERROR"

C80E :WRITE4  LD  BP,AC2,CONST6
5001 LI  ACO,01
1524 JSR  PC,CASOUT  "SEND FLAG"
COPC LD  BP,ACO,RECLen  "GET RECORD LENGTH"
3780 SFLG  CRY
900D SUBB  BP,CONST5  "SUBTRACT 128"
4D02 BOC  NS1GN,BLOCK  "LESS THAN ONE PACKET?"
C00D LD  BP,ACO,CONST5  "NO, SEND 128"
1901 JMP  PC,BLOCK+1
COPC :BLOCK  LD  BP,ACO,RECLen
D0FB ST  BP,ACO,BYTE  "SAVE RECORD LENGTH"
1513 JSR  PC,CASOUT  "SEND BYTE COUNT"
6A00 RADD  AC2,ACO
2210 ROL  AC2,8,0
A0FD :WLOOP  LD@  BP,ACO,STADD  "GET DATA FROM RAM"
1517 JSR  PC,CASOUT  "SEND LS DATA BYTE"
1540 JSR  PC,SUMCHK  "FORM SUMCHECK"
8CFD ISZ  BP,STADD  "INC STORE POINTER"
2D10 SHR  AC1,8,0
1514 JSR  PC,CASOUT+1  "SEND MS DATA BYTE"
ACFC DSZ  BP,RECLen
ACFC DSZ  BP,RECLen  "DEC RECORD LENGTH BY 2"
5C00 NOP
ACFB DSZ  BP,BYTE
ACFB DSZ  BP,BYTE  "DEC BYTE BY 2"
19F4 JMP  PC,WLOOP
5C80 RCFY  ACO,AC2
1508 JSR  PC,CASOUT  "SEND LS SUMCHECK BYTE"
2D10 SHR  AC1,8,0
150A JSR  PC,CASOUT+1  "SEND MS SUMCHECK BYTE"
COPC LD  BP,ACO,RECLen  "GET RECORD LENGTH"
41A2 BOC  REQO,END1  "ALL DATA SENT?"
19DC JMP  PC,WRITE4
CC09: CASIN  LD  BP, AC3, CONST1 ; TAPE INPUT SUBROUTINE
C30A  LD  AC3, ACO, TAPCOM
2802  SHL  ACO, 1, 0
42FD  BOC  PSIGN, CASIN+1 ; DATA AVAILABLE?
C30B  LD  AC3, ACO, TAPDAT ; INPUT TAPE DATA
8000  RTS

5D00: CASOUT  RCPY  AC1, ACO ; TAPE OUTPUT SUBROUTINE
CC09  LD  BP, AC3, CONST1
C30A  LD  AC3, ACO, TAPCOM
42FE  BOC  PSIGN, CASOUT+2 ; TAPE READY?
5C40  RCPY  ACO, AC1
D70B  ST  AC3, AC1, TAPDAT ; SEND OUTPUT DATA
8000  RTS

520A: COMPAC  LI  AC2, 0A ; COMMAND PACKET SUBROUTINE
5002  LI  ACO, 02
15F6  JSR  PC, CASOUT ; SEND FLAG
2A10  SHL  AC2, 8, 0
6A00  RADD  AC2, ACO
500A  LI  ACO, 0A
15F2  JSR  PC, CASOUT ; SEND BYTE COUNT
COFF  LD  BP, ACO, COMBUF ; GET OP CODE & MODIFIER
A82F  AND  BP, CONST39
15EF  JSR  PC, CASOUT ; SEND OP CODE
240A  ROR  ACO, 5, 0
A4FF  OR  BP, COMBUF
2CO6  SHR  ACO, 3, 0
15EB  JSR  PC, CASOUT ; SEND MODIFIER
2410  ROR  ACO, 8, 0
1513  JSR  PC, SUMCHK ; FORM SUMCHECK
15EB  JSR  PC, CASOUT
15EB  JSR  PC, CASOUT+1
15E7  JSR  PC, CASOUT+1
15E6  JSR  PC, CASOUT+1
C0FC  LD  BP, ACO, RECLEN
15E3  JSR  PC, CASOUT ; SEND LS BYTE COUNT
15OC  JSR  PC, SUMCHK
2D10  SHR  AC1, 8, 0
15E1  JSR  PC, CASOUT ; SEND MS BYTE COUNT
3880 :PECD  SFLG  LINK ;PRINT BCD SUBROUTINE
C5FD  LD  PC,AC1,CONSTH ;AC1 = -1000
1506  JSR  PC,DIGIT
519C  LI  AC1,9C ;AC1 = -100
1504  JSR  PC,DIGIT
51F6  LI  AC1,F6 ;AC1 = -10
1502  JSR  PC,DIGIT
1503  JSR  PC,PRINT
8000  RTS
5200 :DIGIT  LI  AC2,00 ;FORM DECIMAL DIGIT
6840  RADD  ACC,AC1
4B02  BOC  NSIGN TEST
7A01  AISZ  AC2,01
19FC  JMP  PC,DIGIT+1
7101 :TEST  CAI  AC1,01
6840  RADD  ACC,AC1
6E00  RXCH  ACO,AC2
4501  BOC  NBSEQO,PRINT
4803  BOC  LINK,RETURN ;SUPPRESS LEADING ZEROS
3800 :PRINT  PFLG  LINK
7830  AISZ  ACO,30 ;ADD ASCII OFFSET
9420  JSR@  BP,OUTEL ;PRINT DIGIT
6E00 :RETURN  RXCH  ACO,AC2
8000  RTS

03E8  (CONSTJ)
FFFF  (CONSTK)

5200 :GBCD  LI  AC2,00 ;INPUT BCD SUBROUTINE
941F  JSR@  BP,INTEL ;INPUT DIGIT
F011  SKNE  BP,ACO,CONST9 ;IS IT N/L?
1908  JMP  PC,CONV ;YES, START CONVERSION
E010  ADD  BP,ACO,CONST8 ;ADD -58
4202  BOC  PSIGN,ERROR
E011  ADD  BP,ACO,CONST9 ;ADD +10
4201  BOC  PSIGN,FOUND
8000 :ERROR  RTS
2A08 :FOUND  SHL  AC2,4,0
6A00  RADD  AC2,ACO
19F5  JMP  PC,GBCD+1
276

500F :CONV LI ACO,OF
5480 RAND ACO,AC2
DOP9 ST BP,ACO,TEMP
500A LI ACO,OA ;ACO = 10
1508 JSR PC,MULTY
5064 LI ACO,64 ;ACO = 100
1506 JSR PC,MULTY
C1EA LD PC,ACO,CONSTJ ;ACO = 1000
1504 JSR PC,MULTY
500D LI ACO,OD
9420 JSR@ BP,OUTEL ;PRINT C/R
COP9 LD BP,ACO,TEMP
8001 RTS+1
2E08 :MULTY SHR AC2,4,0
6200 PUSH AC2
6C80 RXCH AC2,ACO
A828 AND BP,CONST32
6C80 RXCH AC2,ACO
5100 :MULT LI AC1,00 ;16 X 16 MULTIPLY ROUTINE
5310 LI AC3,10
7000 CAI ACO,CO
6940 :LOOPY RADD AC1,AC1
7400 RADC ACO,AC0
4A02 BOC CRY,TST
6980 RADD AC1,AC2
91D9 SUBB PC,CONSTK
7BFF :TST AISZ AC3,FF
19F9 JMP PC,LOOPY
6600 PULL AC2
E4F9 ADD BP,AC1,TEMP
D4F9 ST BP,AC1,TEMP
8000 RTS

4B09 :EREMSG BOC NSIGN,ERRORX ;ERROR CONDITION SUBROUTINE
9421 JSR@ BP,MESS
2945 RE ;PRINT "RSTRY WARNING"
2A52 TR
2CA0 YSP
2BC1 WA
294E RN
24CE IN
23AE G.
8089 RTS-119
9421 :ERROR X JSR@ BP,MESS
2341 FA ;PRINT "FAIL CODE"
24CC IL
1043 SPC
27C4 OD
22A0 ESP
1700 .
BCFA LSEX BP,SUCCDE ;GET SUCCESS CODE
7001 CAI ACO,01
9501 JSR@ PC,PBCD ;PRINT FAIL CODE NO.
8089 RTS-119
B29C (PBCD)
.TITLE CYCLO~TE MONITOR & LOADER WCM 1979

C93B :LDIN    LD    PC,AC2,CONST1 ;LOAD KEY ENTRY
1A51      JMP    AC2,LOAD
4DFF :LOOP   BOC    JC13,LOOP ;TEST MPU SELECT SWITCH
B13A      STW    PC,CONST3 ;ACO - LED ROW 1
6C40      RXCH   ACO,AC1
B139      STW    PC,CONST4 ;AC1 - LED ROW 2
6C40      RXCH   ACO,AC1
6C80      RXCH   ACO,AC2
B137      STW    PC,CONST5 ;AC2 - LED ROW 3
6C80      RXCH   ACO,AC2
6CC0      RXCH   ACO,AC3
B135      STW    PC,CONST6 ;AC3 - LED ROW 4
6CC0      RXCH   ACO,AC3
1CC0      XCHRS  ACO
B134      STW    PC,CONST8 ;PC - LED ROW 6
1CC0      XCHRS  ACO
6000      PUSH   ACO
6200      PUSH   AC2
0400      CFR    ACO
B12E      STW    PC,CONST7 ;FLAGS - LED ROW 5
A128 :CLOOP  LD@    PC,CONST2 ;PANEL SW - ACO
A930      AND    PC,CONST11
41FD      BOC    REQ0,CLOOP ;KEY PRESSED?
6000      PUSH   ACO
A124 :LOOp1  LD@    PC,CONST2 ;PANEL SW - ACO
B92D      SKAZ   PC,CONST12 ;KEY RELEASED?
19FD      JMP    PC,LOOP1
6400      PULL   ACO
4306      BOC    BITC,RETURN ;STOP/STEP?
4433      BOC    BIT1,LOAD ;LOAD?
4608      BOC    BIT2,READ ;READ?
2C06      SHR    ACO,3,0
4316      BOC    BIT0,JUMP ;JUMP?
4601      BOC    BIT2,RETURN ;MONITOR?
19F1 :LOOpA JMP    PC,CLOOP
6600 :RETURN PULL AC2
6400 PULL ACO
3300 PFLG IE3
1973 JMP PC,RINT

A11C :READ LD2 PC,CONST9 ;WG KEYS - ACO
5E00 RCPY AC2,ACO
C200 LD AC2,ACO,00 ;GET STORE DATA
B118 ST PC,CONST8 ;ACO - LED ROW 6
A111 :LOOP2 LD2 PC,CONST2 ;PANEL SW - ACO
A91A AND PC,CONST12
41FD BCC REQ0,LOOP2 ;KEY RELEASED?
6000 PUSH ACO
A10D :LOOP3 LD2 PC,CONST2 ;KEY PRESSED?
B917 SKAZ PC,CONST13
19FD JMP PC,LOOP3
6400 PULL ACO
4650 BCC BIT2,CLOOP
A10F :WRITE LD2 PC,CONST9 ;WG KEYS - ACO
D200 ST AC2,ACO,00 ;STORE DATA
19DD JMP PC,CLOOP
6600 :JUMP PULL AC2
6400 PULL ACO
A10A LD2 PC,CONST9 ;WG KEYS - ACO
1CC0 XCHRS ACO ;DATA - STACK
19E9 JMP PC,RETURN+2

B000 (CONST1)
D000 (CONST2)
D001 (CONST3)
D002 (CONST4)
D003 (CONST5)
D004 (CONST6)
D005 (CONST7)
D006 (CONST8)
D007 (CONST9)
D008 (CONST10)
003F (CONST11)
001F (CONST12)
0014  (CONST13)
F0C6  (CONST14)
000A  (CONST15)
F0F3  (CONST16)
0010  (CONST17)
007F  (CONST18)
0000  (CONST19)
0004  (CONST20)
0100  (CONST21)

6100  ;LOAD  PUSH AC1  ;BOOTSTRAP LOADER
C9FB  LD  PC,AC2,CONST19
C5FA  LD  PC,AC1,CONST19
D480  ST  BP,AC1,AFLAG  ;CLEAR AFLAG
CIF9  LD  PC,AC0,CONST20
D081  ST  BP,AC0,BCOUNT  ;BCOUNT = 4
C1F6  LD  PC,AC0,CONST19
A1EC  ;START  LD@  PC,CONST10  ;GET PTR STATUS
B9F6  SKAZ  PC,CONST21  ;PTR BUSY?
19FD  JMP  PC,START
A1E1  LD@  PC,CONST2  ;GET PTR DATA
2C10  SHR  ACO,8,0
41FA  BOC  REQ0,START  ;IGNORE BLANKS
A9EE  AND  PC,CONST18  ;MASK 7F
5D00  RCPY  AC1,AC0
500A  LI  ACO,0A
5840  RXOR  ACO,AC1
41F5  BOC  REQ0,START  ;IGNORE L/P
500D  LI  ACO,0D
5840  RXOR  ACO,AC1
41F2  BOC  REQ0,START  ;IGNORE C/R
5014  LI  ACO,14
5840  RXOR  ACO,AC1
412F  BOC  REQ0,HALT  ;HALT CODE?
5020  LI  ACO,20
5840  RXOR  ACO,AC1
41EC  BOC  REQ0,START  ;IGNORE SPACES
5028  LI  ACO,28
5840  RXOR  ACO,AC1
411E  BOC  REQ0,CONST  ;START OF COMMENT?
504C  LI  ACO,4C  
5040  RXOR ACO,AC1  ;ADDRESS MARKER?
4119  BOC REQO,ADDRS  
507F  LI  ACO,7F  
5084  RXOR ACO,AC1  ;IGNORE RUBOUTS
41B3  BOC REQO,START  
5C40  RCFY ACO,AC1  
E1D2  ADD PC,ACO,CONST14  ;SUBTRACT 58
4203  BOC PSIGN,HEX  
E1D1  ADD PC,ACO,CONST15  ;ADD 10
4206  BOC PSIGN,FOUND  
191D  JMP FC,HALT  ;HALT IF ILLEGAL CHAR
E1CF :HEX ADD PC,ACO,CONST16  ;SUBTRACT 13
421B  BOC PSIGN,HALT  ;HALT IF ILLEGAL CHAR
E1CE  ADD PC,ACO,CONST17  ;ADD 16
4B19  BOC NSIGN,HALT  ;HALT IF ILLEGAL CHAR
5C0C  NOP  
2E08 :FOUND SHL AC3,4,0  
6ECC  RADD AC3,ACO  ;FORM WORD IN AC3
AC81  DSZ BP,BCOUNT  ;DEC COUNTER
19D4  JMP PC,START  ;FORMED COMPLETE WORD?
C080  LD BP,ACO,AFLAG  
4102  BOC REQC,DATA  ;DATA OR ADDRESS?
5ECC :ADD RCPY AC2,AC3  
19CB  JMP PC,START-5  
DE00 :DATA ST AC2,AC3,00  ;STORE DATA WORD
7A01  AISZ AC2,01  ;INC ADDRESS POINTER
19C8  JMP PC,START-5  
8CB0 :ADDRS ISZ BP,AFLAG  ;SET ADDRESS FLAG
19CB  JMP PC,START  
A1B7 :COMMT LD@ PC,CONST10  ;PTR BUSY?
E9C1  SKAZ PC,CONST21  ;GET PTR DATA
19FD  JMP PC,COMMT  
A1AC  LD@ PC,CONST2  
2C10  SHR ACO,8,0  
A9BA  AND PC,CONST18  
5D00  RCPY AC1,ACC  
5029  LI ACO,29  
5840  RXOR ACO,AC1
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>45F6</td>
<td>BCC NREQO,COMMT</td>
<td>END OF COMMENT?</td>
</tr>
<tr>
<td>19C0</td>
<td>JMP PC,START</td>
<td></td>
</tr>
<tr>
<td>6500</td>
<td>HALT PULL AC1</td>
<td></td>
</tr>
<tr>
<td>1988</td>
<td>JMP PC,LOOPA</td>
<td></td>
</tr>
<tr>
<td>3380</td>
<td>:RINT SFLC IE3</td>
<td>;ARM L3 INTERRUPTS</td>
</tr>
<tr>
<td>B1AB</td>
<td>STO PC,CONST2</td>
<td>;TRIGGER INT TIMER</td>
</tr>
<tr>
<td>7CC0</td>
<td>RTI</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX II- OP CODE INDEX OF INSTRUCTIONS
ALPHANUMERIC SEQUENCE BY HEXADECIMAL

Read down then roght.

."-,

". I '" I", ,,'- i ;:,~l 1:,~~ I:~(

.~~~~
~

,

."t", ".

I

,

,~ ".~O! ..

Co;, '11''''''~

".

"""!! . . . . ,~ .....

"""0<' .. '011.

I

Eou. .......,..... _.'o<k

'

,.1t

A......"""..
Awll ........t9'.
Sh,h I,ll

Sholl,..

~,:~~~,"Hi'~~;~~~~~~~~~~~ .,. ....
......... '

I ''"' ....

&,'1·1....- ......,
a" 2' 2111,h ......

a,,,2.1·N

·""11 .......

"~

Add '1'9~'" I~ '"'Ii..., Qlu. Ufry; mull '0
.....tI........ carTV

tt,. '"

hId;

ALPHANUMERIC SEQUENCE BY HEXADECIMAL

Read down then 'iiht.

I.

,,
,,

,

,,,
,"
,,I
,,,

,

I III

I

,

I'iii

f:J8:' ,
,,
,"

,
,

,
,
,,
I

,
I

,
,
,

,
,
I

~

'11'''''

,
,
,,
,

.ffte"" .<I<l,... ...,

"l/1l'i

,tac';

,,,,,,1ft

,,,,,,,n,,

O....... n•• on ... ".I.II"', •• ,dd,", ~V l;n,p.,", i.'''"'',o.,1 , .."It· D; ''''".~ioeA;I<I,It .... {XX .......'o./I._I:l(X. !121
S'.<I ind,,,,,.; If0.. ,t,CD in'o I,Mlod"'''':odd,ou' un,onq oll.ClII"" (Xl( •
I:)(X • :121
ANa AC~ .. ,110 '""IO.n 011.""0. IXX • '111'11. "'0...1; "'0 _.. I ;.."""",,. d , ....... 0; XX • !11I

"go"" "'.....

L.,d ,t,CO .... IIo .........a... ; B.I I ollOU'IOn (XX. 't;o"" "'._1" .... nd.d la Aea 1-15; 8,,. D.'''' 10ldtrl fO
lOld ACD .. "h ,.nr"",,, of 10ClUOOO (l(~ • ..,..~ 0./1....,,1: XX • :11I
l.ad ,t,CI ... 1Io , •• "." .110111,,,,, Il(X .'~"""Mwn:; XX, :1l1
h.d ,t,C2 ""Ill eO''''.I1 oIIOU"IfIIXX •
111 • ..,,1; XX • :111
L.,d ACl ""'~ <on',.1I .11 ....""" I~X • '01'11 .. ",....1; XX • :111
SI ... ACO I. loc", •• (xx ....,,,.. , .... nl; xx • !121
~'u ACI'aloullo, (Xl(' .. ;0 .... 1110""'1: XX. :121
S, ••• AC2 '0 10"" •• IXX· ,,,,,,,,.. 111.,,,.1; XX. :121
S, ••• ACl ,. 10""'''' IXX • '''I••It' 1110_I; xx • :111
Add ACO '0 la"," •• IXX • ,,~"' .. ",....1: ~X • :121. , .....1' •• ACO
Add ACI '0 I.""." 4X~ •
'M .. nl; ~X • ~121;
10 ACI
Add AC? '0
I ~ X ...."" .... _1; l (. :121; , ...... ' •• _Cl
Add ACl '0 '.""0" (X X • "1"'" .~o .. nl; XX • ~ 121; , ... 1, '" At]
Camp... ACO ,. I.,. ..,." ,U • '..-"" '"a .... l. lX • :121. ,( .0' <quI'
n'" OO'''"CI"O
Cnm"." AC I ,. I~"".~ IXX • """" II'lOv.M. ~ X • ·11I. ,I •• , '0." '''0 n,_' ,",,,.,~ ..
I Co""o'" ACI "'ocIII.n IX l • "I"~ '~O .."I' X X • '121. ,I 00' 'qu" ,.,. n.. ' ,"""''', ••
C.", ... , ACl •• '.""0. (Xx.
(l • '12/: ,( •• , ....... "'0

''1'1'"

'ox,,,,,,,

,I

"Qi.,.. "' ....);

O",'rnol 1<101
ACO" 1011".1101,11",_ "''''ou; ..... 11 ,. ACO. 0...11_ .... .....,; I<Id,,"" (XX.
XX. 1121
I~'.mlll' COnlt.n 01
1;.l;ip nu, in.".'"'on ,I muI. _ 0; ,...,,',... EA: ....doi'...... O'S' "'0".; XX, !Ul
$.ub ...UtO ..... II."U .. I"'. odd, ...
,t,CO; <=It ,. ACO; ... "'0,.. , mOdl"'.wn; ~~ • !111
Jump ,. ",lit ... " .. ,nd,rot'; p"'h PC 0001.
I..... odd'OU"O ton!.~" Oll ..... ,on (u. tt;i,~ "'_n(; XX. :121
Jump .~d""1; I, .... ~d~". '. "n'"nIlO' l.tollO. I~~ • roqo'''' "'.... nl; l(l( • :111
Ca",p .... ACa .. "h ..... Ift!U .IIOUI":•• (XX, '091 .... ",.....1; XX, :121; •• ;~ .1I1;M!rUet,o••1 ACa > (EA(
LOld ,nd".et; 10ld ACO w"h ,.nllnn a' I,,,", odd, ... ; loIeI" ... ,e.'tnll olloe_bOn IU,
"'ow-nJ; XX • !1l1
OR ACa .. 1II eO"'onl1 01, ...." •• (XX. Ifgo1'" slIoWftI; XX • :111; ,.. ~II.e ACO
ANa ACO "'''h
ollonl;cn (XX. ''1'.111 "'.""'(; Xl( • !121; , ...... 10 Aea

..

,

,,

~

Add l(X '0 'f9<!'rr:.k,p ."'"",ue',o. ,I mul.' "'0: X•• !111
RI'u,. If ... '."""P'; add XX I •• ap ol.'o<k 1110 !>lo<' IftUII i. PC; XX. !111;If,'EN fl~
Rt'u," "0" sub, .......;ldd Xl( ,. f,P.' melt ,.0 pi... ",.iI ,. PC: XX" :111

"'''t,

,,,.It

'''0

I

'"1''''' "'._1.

n~., ,o""'",on

"C~ I.,. a.l; XX • !In

