The distributed simulation of highly automated batch manufacturing systems

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

Additional Information:

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

Metadata Record: https://dspace.lboro.ac.uk/2134/10823

Publisher: © Nigel Shires

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/
<table>
<thead>
<tr>
<th>VOL. NO.</th>
<th>CLASS MARK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>date due</th>
<th>Loan copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>19 MAY 1992</td>
<td>12 DEC 1997</td>
</tr>
<tr>
<td>07 OCT 1994</td>
<td>- 9 JAN 1998</td>
</tr>
<tr>
<td>27 JUN 1997</td>
<td>15 JAN 1999</td>
</tr>
<tr>
<td></td>
<td>14 JAN 2000</td>
</tr>
</tbody>
</table>
THE DISTRIBUTED SIMULATION OF HIGHLY AUTOMATED

BATCH MANUFACTURING SYSTEMS

by

Nigel Shires

A Doctoral Thesis
Submitted in partial fulfilment of the requirements
for the award of
Doctor of Philosophy of the Loughborough University of Technology
April 1988

Copyright by Nigel Shires 1988
Abstract

This thesis examines the use of distributed discrete-event simulation techniques as part of an aid to the design of highly automated batch manufacturing systems. The methodology and objectives of the design of highly automated batch manufacturing systems are described and an assessment is made of the use of modelling and simulation as part of the method. Criteria are developed for a simulator used during the detailed design stage.

The different approaches taken by existing simulation systems to building and configuring simulation models and their use of particular simulation techniques are described. Limitations on simulation models due to the sequential processing of event-lists and activity scans are identified in a review of the problems of simulation that current existing distributed simulators have been designed to answer. The advantages of concurrent and distributed computing and in particular, a tightly-coupled multi-microprocessor computing engine for executing the normally batch-processed computing tasks of simulation are identified.

A novel approach to the distribution of the computational tasks in a distributed simulation system is described and the operation of a simulator built using this approach to simulate the operation of highly automated batch manufacturing systems is also described.
The question of whether such a distributed simulator of highly automated batch manufacturing systems satisfies the criteria is examined on the basis of an analysis of the operation of the simulator. It is shown that a number of advantages in the areas of level of detail, configuration, parallel processing and speed of execution can be achieved through the use of distributed computing and multi-processing techniques for simulation during the detail design stage of highly automated batch manufacturing systems.
## CONTENTS

### CHAPTER 1
**INTRODUCTION**

### CHAPTER 2
**LITERATURE SURVEY**

| 2.1 | Introduction | 5 |
| 2.2 | Manufacturing System Design | 5 |
| 2.3 | Simulation for manufacturing system design | 9 |
| 2.3.1 | Computer Graphic Models | 9 |
| 2.3.2 | Physical Models | 10 |
| 2.3.3 | Analytical Models | 11 |
| 2.3.4 | Discrete event simulation models | 12 |
| 2.4 | Simulation Techniques | 15 |
| 2.5 | Distributed Simulation Systems | 19 |
| 2.6 | Multi-computer Processing (Multiprocessing) Systems | 21 |
| 2.7 | Conclusion of chapter | 24 |

### CHAPTER 3
**MANUFACTURING SYSTEMS**

| 3.1 | Introduction | 26 |
| 3.2 | Types of Manufacturing System | 27 |
| 3.3 | Performance Measures | 33 |
| 3.3.1 | FMS performance measures | 36 |
| 3.4 | Stages of manufacturing system design | 38 |
| 3.5 | Analytical Models | 42 |
| 3.6 | Simulation | 44 |
| 3.7 | Conclusion of chapter | 45 |

### CHAPTER 4
**SIMULATION OF MANUFACTURING SYSTEMS**

<p>| 4.1 | Introduction | 46 |
| 4.2 | Configuration | 46 |
| 4.3 | Data Requirements | 48 |
| 4.3.1 | Data Input | 50 |
| 4.3.2 | Data Management | 50 |
| 4.4 | Simulation Techniques | 51 |
| 4.4.1 | The Event Approach | 52 |
| 4.4.2 | The Activity Approach | 52 |
| 4.4.3 | The Process-interaction Approach | 53 |
| 4.4.4 | The Three-phase Approach | 54 |
| 4.5 | Performance | 54 |
| 4.5.1 | Detailed Simulation | 55 |
| 4.5.2 | Performance requirements for detailed simulation | 57 |
| 4.6 | Conclusion of chapter | 58 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2.5</td>
<td>Loading/Unloading Strategy.</td>
<td>134</td>
</tr>
<tr>
<td>9.3</td>
<td>System Control Rules/Technique.</td>
<td>135</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Pallet Storage Buffer Unload Strategy.</td>
<td>140</td>
</tr>
<tr>
<td>9.4</td>
<td>Deadlock.</td>
<td>142</td>
</tr>
<tr>
<td>9.5</td>
<td>Results and simulation output.</td>
<td>146</td>
</tr>
<tr>
<td>9.5.1</td>
<td>Derivation of Utilisations.</td>
<td>147</td>
</tr>
<tr>
<td>9.5.2</td>
<td>Animated Output.</td>
<td>148</td>
</tr>
<tr>
<td>9.6</td>
<td>Hardware Operation.</td>
<td>150</td>
</tr>
<tr>
<td>9.6.1</td>
<td>Start-up State.</td>
<td>150</td>
</tr>
<tr>
<td>9.6.2</td>
<td>Loading the software.</td>
<td>151</td>
</tr>
<tr>
<td>9.6.3</td>
<td>Executing the software.</td>
<td>151</td>
</tr>
<tr>
<td>10.1</td>
<td>Introduction.</td>
<td>160</td>
</tr>
<tr>
<td>10.2</td>
<td>Description of System</td>
<td>160</td>
</tr>
<tr>
<td>10.3</td>
<td>Configuration of the Simulator.</td>
<td>163</td>
</tr>
<tr>
<td>10.3.1</td>
<td>General Layout Configuration.</td>
<td>163</td>
</tr>
<tr>
<td>10.3.2</td>
<td>AGV data and Animation route data configuration.</td>
<td>165</td>
</tr>
<tr>
<td>10.3.3</td>
<td>Layout for graphical animation.</td>
<td>168</td>
</tr>
<tr>
<td>10.3.4</td>
<td>Configuration Process or Technique.</td>
<td>169</td>
</tr>
<tr>
<td>10.4</td>
<td>Loading and executing the software.</td>
<td>172</td>
</tr>
<tr>
<td>10.4.1</td>
<td>Executing the software.</td>
<td>172</td>
</tr>
<tr>
<td>10.4.2</td>
<td>Characteristics of the data used in the case study</td>
<td>174</td>
</tr>
<tr>
<td>10.4.3</td>
<td>Time increment of the simulation.</td>
<td>174</td>
</tr>
<tr>
<td>10.5</td>
<td>Case Study - Simulation Output.</td>
<td>176</td>
</tr>
<tr>
<td>10.5.1</td>
<td>Validation of the Operation of the Simulation.</td>
<td>176</td>
</tr>
<tr>
<td>11.1</td>
<td>Introduction.</td>
<td>179</td>
</tr>
<tr>
<td>11.2</td>
<td>Results in the area of software technique.</td>
<td>179</td>
</tr>
<tr>
<td>11.2.1</td>
<td>Functionality of the modules.</td>
<td>180</td>
</tr>
<tr>
<td>11.2.2</td>
<td>Level of detail.</td>
<td>180</td>
</tr>
<tr>
<td>11.2.3</td>
<td>Simulation Executive (Events-list Processing).</td>
<td>184</td>
</tr>
<tr>
<td>11.3</td>
<td>Result of the study of hardware techniques.</td>
<td>184</td>
</tr>
<tr>
<td>11.4</td>
<td>The User Interface.</td>
<td>185</td>
</tr>
<tr>
<td>11.4.1</td>
<td>Configuration of a particular simulation model.</td>
<td>186</td>
</tr>
<tr>
<td>11.4.2</td>
<td>Configuration of the modules.</td>
<td>186</td>
</tr>
<tr>
<td>11.4.3</td>
<td>Start-up and running of the simulator.</td>
<td>187</td>
</tr>
<tr>
<td>11.4.4</td>
<td>Collection of statistics.</td>
<td>187</td>
</tr>
<tr>
<td>11.4.5</td>
<td>Validation and Verification.</td>
<td>188</td>
</tr>
<tr>
<td>11.5</td>
<td>The speed of execution of the simulation model.</td>
<td>188</td>
</tr>
<tr>
<td>11.5.1</td>
<td>The number of cycles used in the simulation experiments.</td>
<td>190</td>
</tr>
<tr>
<td>11.5.2</td>
<td>Number of events in the simulation experiments.</td>
<td>190</td>
</tr>
<tr>
<td>11.5.3</td>
<td>The simulation experiments.</td>
<td>192</td>
</tr>
<tr>
<td>11.5.3.1</td>
<td>Configuration of the experiments - software.</td>
<td>192</td>
</tr>
<tr>
<td>11.5.3.2</td>
<td>Configuration of the experiments - hardware.</td>
<td>195</td>
</tr>
<tr>
<td>11.5.3.3</td>
<td>Simulator speed measurements.</td>
<td>195</td>
</tr>
<tr>
<td>11.6</td>
<td>The influence of the synchronisation code.</td>
<td>201</td>
</tr>
<tr>
<td>11.7</td>
<td>Discussion of Results.</td>
<td>201</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>4</td>
</tr>
<tr>
<td>2. The range of application of manufacturing systems</td>
<td>28</td>
</tr>
<tr>
<td>3. Transfer Line</td>
<td>28</td>
</tr>
<tr>
<td>4. Flexible Manufacturing Cell (Rotational Components)</td>
<td>29</td>
</tr>
<tr>
<td>5. Flexible Manufacturing System</td>
<td>31</td>
</tr>
<tr>
<td>6. Unmanned Station</td>
<td>31</td>
</tr>
<tr>
<td>7. Hierarchical Control System</td>
<td>34</td>
</tr>
<tr>
<td>8. Some steps of an FMS design procedure</td>
<td>40</td>
</tr>
<tr>
<td>9. A distributed simulation system</td>
<td>68</td>
</tr>
<tr>
<td>10. Block Diagram of a hardware reliability simulator</td>
<td>74</td>
</tr>
<tr>
<td>11. Proposed parallel computer for simulation</td>
<td>74</td>
</tr>
<tr>
<td>12. Manufacturing System Design</td>
<td>90</td>
</tr>
<tr>
<td>13. Manufacturing System Emulation</td>
<td>92</td>
</tr>
<tr>
<td>14. Flow Chart of Design Aid</td>
<td>94</td>
</tr>
<tr>
<td>15. Flow Chart of Design Aid - 2</td>
<td>95</td>
</tr>
<tr>
<td>16. Hardware - Intel SYS310 Computer</td>
<td>99</td>
</tr>
<tr>
<td>17. Software Modules</td>
<td>107</td>
</tr>
<tr>
<td>18. Structure of a PASCAL-86 Non-Main Module</td>
<td>114</td>
</tr>
<tr>
<td>19. Buffer Activities - 1</td>
<td>120</td>
</tr>
<tr>
<td>20. Buffer Activities - 2</td>
<td>121</td>
</tr>
<tr>
<td>21. Pallet Storage Buffer Load/Unload Strategy</td>
<td>122</td>
</tr>
<tr>
<td>22. Machining Activities - 1</td>
<td>125</td>
</tr>
<tr>
<td>23. Machining Activities - 2</td>
<td>126</td>
</tr>
</tbody>
</table>
### List of Figures (continued)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.</td>
<td>AGV Activities - 1</td>
<td>129</td>
</tr>
<tr>
<td>25.</td>
<td>AGV Activities - 2</td>
<td>130</td>
</tr>
<tr>
<td>26.</td>
<td>AGV Movements at Linear Buffer</td>
<td>138</td>
</tr>
<tr>
<td>27.</td>
<td>Deadlock</td>
<td>143</td>
</tr>
<tr>
<td>28.</td>
<td>Multiprocessor System : Start-up State</td>
<td>152</td>
</tr>
<tr>
<td>29.</td>
<td>Multiprocessor System : State after loading simulation software</td>
<td>153</td>
</tr>
<tr>
<td>30.</td>
<td>Synchronisation code of a multi-processor simulation</td>
<td>156</td>
</tr>
<tr>
<td>31.</td>
<td>Multiprocessor System : State when running simulation software</td>
<td>158</td>
</tr>
<tr>
<td>32.</td>
<td>Case Study Layout</td>
<td>162</td>
</tr>
<tr>
<td>33.</td>
<td>Case Study Layout - Machine Numbering</td>
<td>162</td>
</tr>
<tr>
<td>34.</td>
<td>Case Study Layout - Buffer Numbering</td>
<td>164</td>
</tr>
<tr>
<td>35.</td>
<td>Case Study Layout - Station Numbering</td>
<td>164</td>
</tr>
<tr>
<td>36.</td>
<td>Case Study Layout - AGV Station Numbering</td>
<td>166</td>
</tr>
<tr>
<td>37.</td>
<td>Case Study Layout - Route Numbering</td>
<td>166</td>
</tr>
<tr>
<td>38.</td>
<td>Case Study Layout - Buffer Route Numbering</td>
<td>167</td>
</tr>
<tr>
<td>39.</td>
<td>Simulation configuration directory tree structure</td>
<td>171</td>
</tr>
<tr>
<td>40.</td>
<td>Comparison between multi-processor and serial simulation</td>
<td>202</td>
</tr>
<tr>
<td>41.</td>
<td>Configuration of the simulator</td>
<td>235</td>
</tr>
<tr>
<td>42.</td>
<td>Case study layout for configuration example</td>
<td>237</td>
</tr>
<tr>
<td>43.</td>
<td>Route numbering for configuration example</td>
<td>237</td>
</tr>
<tr>
<td>44.</td>
<td>Graph paper layout of configuration example</td>
<td>238</td>
</tr>
<tr>
<td>45.</td>
<td>Interactive use of program 'genroute'</td>
<td>239-241</td>
</tr>
</tbody>
</table>
### List of Figures (continued)

<table>
<thead>
<tr>
<th>Figure Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>46. Interactive use of programs 'cgis', 'genfort06' and listing of 'fort06.dat'</td>
<td>242-243</td>
</tr>
<tr>
<td>47. Interactive use of program 'gen09'</td>
<td>244</td>
</tr>
<tr>
<td>48. Interactive use of program 'gentcon'</td>
<td>245</td>
</tr>
<tr>
<td>49. Interactive use of program 'genpf'</td>
<td>246</td>
</tr>
<tr>
<td>Table Title</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1. Examples of current distributed simulators</td>
<td>64</td>
</tr>
<tr>
<td>2. Characteristics of the case study data</td>
<td>175</td>
</tr>
<tr>
<td>3. Configurations of the simulation system</td>
<td>193</td>
</tr>
<tr>
<td>4. Configurations of the simulation system</td>
<td>194</td>
</tr>
<tr>
<td>5. Simulation speed measurement</td>
<td>196</td>
</tr>
<tr>
<td>6. Comparison of simulation speed measurement</td>
<td>197</td>
</tr>
</tbody>
</table>
Acknowledgements.

The author wishes to acknowledge the advice and support of Dr. E.A. Roberts, Prof. R. Bell, Mr. S.T. Newman, Mr. G. Charles, and many other colleagues at the Manufacturing Engineering Dept. of Loughborough University of Technology.

The author also acknowledges the guidance given to the FMS design aid project by the representatives of the collaborative companies involved: Mr. R. Freeman of NGL Ltd., Mr. A. Eldred and Mr. T. Stratton of Baker-Perkins Ltd. and Mr. R. Duddin.

Acknowledgment is also due to the Asea Brown Boveri Research Centre, Baden for the facilities provided for the production of this thesis and to Mr. J. Bastos, Mrs. D. Scheuber and other colleagues at the Research Centre.

The work presented here is original, except where reference is made to the work of others, and was carried out by the author as part of a multi-person project team involved in research into design aids for highly automated batch manufacturing systems. No part of the work presented has been submitted to any other University or Institute for the purpose of obtaining academic degrees or qualifications.
In recent years there have been radical developments in manufacturing systems suitable for the manufacture of small batches of parts. With the introduction of numerically-controlled (NC) machines, machining systems evolved from collections of manually-operated, independently working machines to groups of NC machines controlled by a supervisory computer (Direct Numerical Control, DNC). The recent introduction of automatic workpiece transfer equipment, tool handling devices, and additional levels of control has made the evolution proceed further, to manufacturing systems capable of the highly automated efficient manufacture of workpieces in small batches. This evolution has been a process involving increasing degrees of complexity at each stage.

The increasingly competitive and dynamic nature of the manufacturing environment in recent years means that a manufacturer requires a faster response in the quotation for the supply of new products and improved ways of re-organising machining systems to manufacture most efficiently in order to produce at least cost. These requirements mean that the cost and time required to design new machining and manufacturing systems must be lowered in order to improve the
attraction of investing in more efficient systems, and hence to improve the capability of survival in this competitive environment. This has led to an increasing requirement for sophistication in the computer-aids which are used during the design process of manufacturing systems.

Both of these influences, viz. the increasing complexity of the machining systems and the improvement required in the cost, duration and accuracy of the manufacturing system design process, have led to the requirement for a coherent computer aid which supports the complete design process, and supports the user to easily make decisions and choices about a proposed system. Such a computer-aid requires modelling facilities in order to predict the performance of a proposal.

Two modelling methods which are used at different stages of the design process are analytical models (used during the initial planning stage) and discrete-event simulation models (used at the initial planning stage through to the stage of very detailed planning).

A discrete-event simulation model which is used at the detail-planning stage must, however, be consistent with the overall objectives of improving the design process, in that it must provide a fast response and include sufficient detail for modelling at this stage of design. Two ways of improving a computer simulation system, so that these requirements can be met, are distributed computing and multi-processing.
Distributed computing is concerned with distributing the parts of a computer system in such a way that the advantages of distribution can be achieved (functionality, modularity, etc.). Multi-processing is concerned with applying more than one computer processor to a computer program in order to improve the execution speed of the program. Parallel processing is a particular technique of multi-processing where parts of the program which are processed by separate computer processors may be executed simultaneously in parallel.

The main subject of this thesis is the study of the application of these techniques to the discrete-event simulation of highly automated batch manufacturing systems at the detail-design stage, in order to determine whether these techniques will be applicable in this area in the future (figure 1).

The work reported in this thesis has been carried out by the author as part of a multi-person project team involved in research into design aids for highly automated batch manufacturing systems. The character of the work has been strongly influenced by the needs of the industry as expressed by the views of the industrial collaborators. Interactions with colleagues and industry showed that some research into the aspects of parallel processing was required which could be used for improving the support given by a design aid.
Requirements from Industry

| Increasing Complexity of Manufacturing Systems | Faster response and lower cost of manufacturing system design |

Requirement for a design aid

A design aid requires modelling facilities - two types:

- Analytical Models (used at the pre-detail design stage)
- Simulation models (used throughout the design process)

Simulation at the detail design stage must be consistent with initial requirements (provide a fast response, be able to model complex systems). How can this be achieved?

Techniques of distributed computing

Techniques of multi-processing

Techniques to improve performance of simulation systems

Object of work: Will these techniques be able to improve the simulation of highly automated batch manufacturing systems?

Introduction

Figure 1
CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

The object of this literature survey is to review recent work in the areas of manufacturing system design; manufacturing systems simulation; simulation techniques; distributed simulation; and multi-computer processing systems. The review of literature in the areas of manufacturing system design, planning and simulation is directed mainly to material describing flexible manufacturing systems (FMS) using modern automation techniques.

2.2 Manufacturing System Design

Suri and Whitney /SURI84/ describe manufacturing system design for system modification, creation or expansion as an activity which is carried out with a long term time horizon at the upper management level. This activity is often carried out alongside long-term changes in part-mix.
The actual activity of manufacturing system design is described by researchers at the Charles Stark Draper Laboratory /CHAR84/ as consisting of four main steps. These are:

- selecting parts;
- selecting machines;
- designing configurations; and
- evaluating and judging different configurations.

Bilalis /BILA83/ reviews literature which expands these four steps and describes them in more detail. Among those reviewed are Barash et al. /BARA75/ who describe the process in a similar manner with the exception of expanding the last stage of evaluation to include a separate stage of simulation. Another is Eversheim /EVER79/, who describes seven steps of manufacturing system design as:

- Analysis of machining requirements;
- choice of system structure;
- determination of machine requirements;
- determination of the degree of automation;
- design of the transport system;
- conception of the organisational control; and
- justification of the economic operation of the system.

The problem of FMS design is described by Browne, Chan and Rathmill /BROW85/. Three levels or phases of activity during system design which apply to flexible manufacturing systems are identified: these are planning, design and implementation. Within the planning phase,
two basic questions should be answered, which are: 'what components (range and families) are to be manufactured and how?', and 'what quantities of components are required and in what batch sizes?'. The design phase has two parts. These are selection of the type of FMS, and detailed system design. The third phase of implementation consists of software design, hardware design and implementation.

The first of the tasks described in /CHAR84/, that of 'selecting the parts to be produced by a manufacturing system', can be achieved by using technological and geometrical criteria: for example, a part family may require drilled holes as part of their manufacture. Steinhilper /STEI84/ contends that part families suitable for Flexible Manufacturing systems cannot be established with purely geometrical or technological classification systems but should be selected by a combination of technological, geometrical and organisational criteria. An example of a part family chosen in this way is manufactured in the FMS facility of Normalair-Garrett at Crewkerne and is described by Wills et al. /WILL83/ where the emphasis is to produce a kit of parts at regular intervals, each 'kit' making one complete assembly.

The next tasks, those of 'selecting machines' and 'designing configurations of hardware' can be such a large task that not all of the candidate solutions can be examined. Hannam /HANN85/ describes the influences of the choice of particular items of hardware. The types of hardware described are: Machine tools; Pallets and fixtures; Tooling; Work handling devices; and Storage. Tooling, particularly, is identified as an area of choice, the result of which can restrict the number of parts in a system and force certain machines to be dedicated to particular tasks.
The task of 'designing the configuration of the manufacturing system' is described in /CHAR84a/. This task involves choosing specific machines from vendors each with its own particular limitations. The combination of a variety of these choices leads to a number of different configurations, each with attributes and limitations. The capacity, balance and cost of the system should also be known at this point.

There are a number of computer aids available for the process of manufacturing system design. Caputo /CAPU83/ describes the use of Computer-Aided-Design (CAD) for part analysis and manufacturing system layout. Computer aids for the selection of machine tools and process simulation are described by Warnecke /WARN82/; Warnecke et al. /WARN84/ also describe a range of computer aids, for present condition and weak point analysis; for material flow analysis; and for simulation-supported selection and optimization.

Such computer aids are essential for the final task of designing manufacturing systems, that of 'evaluation and justification'. Evaluation of a proposed system can only take place following the selection of suitable criteria. Warnecke and Scharf /WARN73/ describe a range of criteria for manufacturing systems depending on at which level the performance of the manufacturing system is to be assessed. If performance is to be viewed as that of the whole plant or factory, criteria may include profit rate, return on investment, productivity, delivery term and variety of products. If performance is to be viewed at the manufacturing system level, criteria may include flexibility, output, cost, utilization, capacity, availability etc. Performance may also be viewed at the product level: the criteria here may
include material costs, piecepart manufacture time and cost, quality, throughput time, etc. Ranky /RANK83/ describes some FMS evaluation criteria as capacity; accuracy; reliability; flexibility; quality; processing time and efficiency. Some criteria for the evaluation of the FMS design process itself are also included, such as time, documentation, training, maintainability and acceptance.

2.3 Simulation for manufacturing system design

The activity of manufacturing system design includes the task of evaluating different configurations of the manufacturing system on the basis of a set of criteria. A particular aid to evaluating the performance of a manufacturing system is simulation.

The types of simulators which are used typically during the process of manufacturing system design include:

- Computer Graphic Models;
- Physical models;
- Analytical models (Mathematical and Queueing models); and
- Discrete event simulation models.

2.3.1 Computer Graphic Models.

Bonney et al. /BONN85/ describe the evaluation and use of a computer graphic simulator (GRASP). The graphical robot simulator is a computer-aided design system for modelling and evaluating industrial robot work cells and flexible manufacturing systems. It incorporates a three-dimensional solid modeller. The simulation of the movement of robots, machines and conveyors aids the designer to detect possible
collisions between robots and pallets, fixtures and other hardware in the robot workplace.

Another program which can be used to define, program and simulate a robotic workcell environment is described in /CADD83/. This system (called 'Designer') is based on a graphics system called 'Robographix', both from Computervision. The simulation allows the user to examine factors such as alternative workcells, locations, joints and motions. A similar program from McAuto (called 'PLACE') is described by Graiser /GRAI83/.

2.3.2 Physical Models.

Physical models of manufacturing systems are usually built using technical modelling kits. Diesch and Malstrom /DIES85/ describe a physical model of an FMS. When such a model is controlled by a computer, software intended for eventual use in the actual system can be created and tested on the physical simulator prior to the construction of the actual system. The primary disadvantage of a physical simulator is its high cost (both of the kit and of the design and construction of the model).

A physical model used by machine-tool builders Scharmann to simulate an FMS at the Caterpillar plant in Gosselies, Belgium is described by Percival /PERC83/. The model was built using a Fishertechnik modelling kit and cost around DM 40,000 (1983). The model was constructed in fine detail with the individual elements of the simulated machine tools moving under full control of the DNC computer. The advantage of pre-testing the control software on the model meant that commissioning the software took only two weeks. However it is
reported in the same reference that more recently Scharmann have used micro-computer simulation and graphics in preference to a physical model.

The use of physical models to generate representative system data, validate mathematical and computer simulation models, develop and test system software and study the system is described by Young /YOUN84/. The physical model is used to fill a gap which exists in the design approach to manufacturing systems which occurs when conflicts that occur between physical devices cannot be easily modelled by descriptive computer simulation models.

2.3.3 Analytical Models.

A well-documented queueing model for the analytical performance evaluation of manufacturing systems is CAN-Q, described by Solberg /SOLB81/, /SOLB82/. This model provides a wide range of performance measures with acceptable accuracy from minimum data requirements. The model is highly aggregated and is unsuitable for the purpose of answering detailed questions of design but provides the best results when used to support calculations regarding sizing and balancing of the proposed system during the design phase. CAN-Q uses a normalisation constant approach which allows detailed marginal probability distributions describing the performance of the modelled system to be calculated.

Reiser and Lavenberg /REIS78/ contend that for practical purposes these marginal distributions contain too much detail. In order to calculate much simpler performance measures such as mean queue lengths, mean waiting times, utilisations and throughputs, mean value
analysis can be used. Mean value analysis provides more efficient algorithms than those used by CAN-Q if only the mean measures of performance are required.

The use of mean value analysis for modelling closed networks of queues is described by Suri and Hildebrant /SURI84a/. The model, called MVAQ, provides predictions of FMS performance which are considered reasonable for 'ball-park' or 'first-cut' planning and control decisions and is easier to use than simulation models (data input requirements being small). MVAQ can also be used during an interactive session at a computer terminal because of the short response time. However, the model aggregates and approximates the occurrence of activities such as the movement of parts and the machining of particular operations, and it is reported in the same reference that typical predictions can be inaccurate by up to 20%.

MVAQ also extends the functionality of CAN-Q with additional features such as the modelling of multiple part classes (a fixed number of pallets, each of which can be dedicated to certain part types), the requirement of less computer memory and the maintenance of efficiency of calculation and numerical stability when the number of parts and machines in a system becomes very large.

2.3.4 Discrete event simulation models.

Hartley /HART84/ describes when discrete event simulation models can be used to the greatest effect, which is at a greater level of detail than when mathematical models should be used. This is also supported by Browne, Chan and Rathmill /BROW85/, where during the detailed system design stage, the analytical approach (using queueing models...
and mathematical models) is considered in general not to be suitable because too much aggregation and too many simplifying assumptions are required. Questions relating to the transport system, machine loading and scheduling cannot be realistically addressed. Digital computer simulation is likely to be superior to analytical methods at this stage of the design procedure. 'Ball-park' answers may be provided by analytical models but the complex nature of FMS, coupled with the need to maximize its performance, means that it can only be effectively designed by iterative computer simulation methods.

The application of both types of models for the design of an automated circuit card manufacturing system is described by Dietrich and March (DIET85). The major advantage of the analytical model is that results can be achieved quickly and inexpensively. During the design phase of a project when many candidate designs are under consideration, exact data is not available and rough estimates are required immediately. Simplified analytical models can provide substantial time savings during this stage of design and provide an insight into the design of the simulation model. They can also significantly reduce the range of parameters over which the simulation model is run which is an advantage because the simulation model is described as being expensive and time-consuming to design, build, test and run. However, the simplifying assumptions made in the analytical model will not allow the representation of all aspects of the manufacturing system.

There are discrete event simulation models specifically for manufacturing systems simulation. An example of this type of model is MAST (MAnufacturing System design Tool) (CITR84). The use of MAST for a particular simulation is described by Lenz (LENZ85). Typical
output statistics which are available include: a production summary report; part performance report; a station utilizations report; a report on pallet numbers and types in the system; and a queue length report for each station in the system.

A simulation model designed for a manufacturing system producing rotational, rather than prismatic, parts is described by Bilalis and Bell in /BELL82/. A variety of loading and control strategies were implemented within the simulation models. It was concluded from the study that the performance of the particular system modelled was highly dependent on the loading and control strategies used.

Brennan and Davies /BREN85/ describe the use of a simulation model as part of a formal evaluation procedure for management. The model was written in the GASP-IV simulation language and is applied to a job-shop environment of a machine tool builder. An interactive environment was provided for the user of the model to change variables such as resources, product types, production variables and product mix.

Rathmill and Chan describe a simulation model written using the GPSS simulation language for the 600 group FMS /RATH83/. This particular investigation was designed to relate the total production time or throughput to variation in the duration of an automatic inspection cycle within the modelled system.

A comparison of different computerized design tools for flexible manufacturing is described by Valcada et al. /VALC84/. The main conclusions reached by the authors are that different software tools may be used at different stages in the FMS design process and that it
is impossible to have both high accuracy and rapidity of use of the software tool. Four simulation systems are described and compared.

Another review of simulation programs for FMS design are described by Mills and Tavalage /MILL85/. Four systems are described (GCMS, MAP/1, GEMS and SIMAN) and a generalised FMS simulator called PATHSIM is described for use as a design aid.

2.4 Simulation Techniques

Changes in systems may take place continuously with time or at discrete time intervals. Pidd /PIDD84/ describes the simulation systems and languages which are designed to model systems that change at discrete time intervals as discrete event simulation languages and systems. The systems to be modelled may also be stochastic or deterministic. A system in which the behaviour of most, if not all of the elements in the system can be entirely predicted is more suited to the use of deterministic techniques.

Discrete event simulation languages usually embody one of the four main approaches to simulation. These approaches are described by Carrie /CARR85/, and Roberts and Shires /ROBE85/ as the event approach; the activity approach; the process interaction approach; and the three-phase approach. Pegden /PEGD84/ described the particular features of the event approach and the activity approach. The event approach models a system so that an instant in time when the state of the system changes is called an event time and the logic associated with the change is called an event. The activity approach models an activity within a system by using two related events; the first event models the changes at the start of the activity and the
schedules the second event to occur at the end of the activity.

The three-phase approach was introduced and described in detail by Tocher /TOCH63/. The first phase consists of a time scan to determine the next event due to begin; the second phase consists of activities occurring due to the advancement of simulated time to that of the next event due to begin; and the third phase consists of activities occurring due to changes following the second phase.

The three approaches are also described and reviewed by Hooper /HOOP86/. Algorithms in 'pseudo-code' are given for each strategy and the characteristics of each approach are described. The conclusion is reached that the strategy used in a simulation model profoundly affects the modelling/simulation task because it dictates the point of view of the system to be modelled, and it may affect the amount of modelling effort and computer time required.

In addition to the standard computing languages with which simulation models can be built based on one or other of these approaches, other specialised languages are available which embody one of these approaches. They allow the expression of standard simulation routines in a straightforward manner and provide mechanisms for time handling, statistic collection etc.

A number of simulation languages which embody the process-interaction approach to simulation are reviewed by Banks and Carson /BANK85/. The languages GPSS/H /HENR83/, SIMAN /PEGD84/, SIMSCRIPT /RUS800/, and SLAM /PRIT79/, are reviewed and the facilities offered by each are compared. Cellier /CELL83/ carries out a wider review of fifteen simulation languages including languages for continuous, discrete and
combined systems simulation. Features such as numerical behaviour, program execution, data handling, input/output and status of implementation are examined and compared.

Some simulation languages can be combined with specific computer hardware and/or other software to enable easier use and better presentation of output, for example. This combination of a simulation language with other aids to its use can be called a simulation system. Examples of simulation systems are HOCUS /PEIN00/, and SEE-WHY /ISTEO0/. SEE-WHY is a combination of a set of FORTRAN subroutines and graphics software to be run on a specific computer colour terminal: it can also be combined with EXPRESS /SCHA85/, a software extension to aid its use. Another simulation system is OPTIK /INS100/, which incorporates visual-interactive graphics and a database in the system.

Some of the software extensions to the basic simulation systems are specifically known as program generators which aid the user to generate code in the simulation language. Mathewson /MATH84/ reviews program generators applied to model building. A particular program generator called DRAFT is introduced which produces FORTRAN code that is intended to use a set of FORTRAN subroutines providing mechanisms for simulation with the SIMON simulation language/system. Another example of this type of system is described by Clementson in /CLEM82/. The program generator is called CAPS, which produces code for the ECSL system. Haddock and Davis /HADD85/ describe a program generator for manufacturing cell design and control. This particular software aid produces code for the SIMAN simulation system.
Extensions to standard computing languages can be made to facilitate simulation. In particular, the process-interaction approach has been developed in connection with the PASCAL language. An extension of PASCAL to enable quasi-parallel programming to be supported, and hence process-interaction simulation by the use of coroutines is described Kriz /KRIZ80/. The parallel activities of the simulated system are modelled by processes which can be running, suspended or blocked, but there can never actually be more than one running process at a time (hence the term 'quasi-parallel programming'). Raczynski /RACZ86/ reports a similar concept of extensions of PASCAL where, instead of the use of coroutines, a process/event structure is used so that the execution of a process can be suspended but not the execution of an event. The system is called 'PASION', and is in the form of a pre-processor of a modified PASCAL program which produces a standard PASCAL program to be compiled in the normal way.

In these process-oriented simulation systems, a number of simulation processes appear to execute in parallel. In fact, only one such process is executing at a given time, and that process continues to execute until it chooses to stop, at which time the simulation system schedules another process for execution. This type of process-oriented simulation scheme utilises one master simulation clock. An implementation of the simulation of manufacturing cells using the ADA language is described by Antonelli et al. /ANTO86/. This implementation allows the tasks or processes to take advantage of any underlying multiprocessor hardware that is offered by the computer system and ADA compiler by providing each task with a local simulation clock in addition to the global clock which is synchronised when processes interact.
2.5 Distributed Simulation Systems.

Distributed simulation is the term applied to the process of distributing the simulation of a physical process over a network of computer processors. Current research into distributed simulation is described by Reynolds (REYN85) as being concerned with alleviating the time and memory constraints often encountered in large scale simulations.

Techniques of distributed simulation are being applied to the solution of differential equations describing dynamical systems and to discrete event simulation. A proposal for a simulation system using standard minicomputers for low-cost multiprocessing was described by Korn (KORN72). This proposal was designed primarily for the simulation of dynamical systems modelled in terms of differential and difference equations. It envisaged the use of standard, unmodified minicomputers connected via a common bus called UNIBUS to provide faster and cheaper simulation. A similar proposal for low-cost parallel computation for optimization and simulation is described by Singh and Allidina (SING85). This proposal is to enable the construction of specific computer architectures using low-cost standard single processor computer boards to solve high-order dynamical models.

Other examples of distributed simulation systems include those described by Wyatt (WYAT85) and Cellier and Rimvall (CELL85). Wyatt describes a multitasked implementation of a microprocessor-based discrete event simulation. A PASCAL-based simulation language is used, called SIMPAS (BRYA81) which provides constructs for simulation primitives. Tasks in the system are allocated by their simulation support function. One task includes event routines of the
simulation; another task includes a collection of statistics routines; another task was allocated to random number generation. The system was implemented on a single computer multitasking system which was designed to model a true multi-computer processing system. Cellier and Rimvall describe a similar concept of structuring simulation systems. A complex simulation program is decomposed into a series of simpler tasks which are basically independent of each other. These tools are interfaced to each other through a database. Typical tasks would be graphics and statistical routines.

DESFOR, a FORTRAN-based discrete event simulation package is described by Bruno and Canuto /BRUN82/. DESFOR is based on the process-interaction approach to simulation and is designed to use concurrent programming techniques. The simulation program is decomposed into quasi-parallel processes or coroutines. The execution of any single process is linked to the behaviour of its coroutines. Standard components of production systems such as workstations and storage areas are modelled by particular linguistic constructs of DESFOR. Deans and Mann /DEAN83/ describe the development of a computing instrument which simulates the reliability aspect of a system. A particular component of a system is simulated by the generation of random variables with a particular statistical distribution. Each random number generator can proceed simultaneously with other generators. The generators 'interact' in accordance with a mathematical model describing the overall system.

The benefits of distributed simulation depend on many variables. Berry and Jefferson /BERR85/ describe an investigation of the lower bound on the duration of execution of an object-oriented simulation...
carried out in parallel. A critical path algorithm is used to analyse how much 'speed-up' can be obtained for a particular simulation. Different decompositions of the same problem can be compared by analysing how much concurrency there is in each decomposition. The maximum speed-ups obtained in three experiments were 1.48 and 3.55 with 10 parallel processes and 1.67 with 3 parallel processes. These figures are upper bounds on the speed-up, and it is noted that no simulation mechanism can be expected actually to achieve these figures.

The relationship between the potential parallelism of a simulation and the processors employed has been studied by Livny /LIVN85/. The results of the study showed that systems exhibiting a high degree of concurrent activity did not necessarily entail the use of a simulator with a high degree of inherent parallelism.

Other software which is required for distributed simulation is that which maps an abstract simulation model onto a distributed simulator or multi-processor architecture. Such software is described by Concepcion /CONC85/. A particular mapping algorithm is described which is to be implemented on a proposed hardware configuration. A hardware configuration for a real-time multiprocessor simulator is described by Blech and Arpasi /BLEC85/. The hardware includes 'off-the shelf' microcomputer boards and minimal customized interfacing, and partitioning algorithms have been evaluated on the multiprocessor computer.

2.6 Multi-computer Processing (Multiprocessing) Systems.

A multiprocessor system consists of three main items: the
application software; the system software; and the computer hardware. Each of these items can contribute to the four major characteristics of multiprocessor systems which demonstrate their advantage over single processor systems. These characteristics are described by Woodward /WOOD81/ as throughput; flexibility; availability; and reliability.

The application software is usually constructed by using a high-level language. Syre /SYRE82/ describes the attributes of an ideal high-level language for multiprocessor systems. The attributes include the implicit expression of parallelism; abstraction from hardware considerations (such as memory organisation); ease-of-use and the inclusion of powerful constructs. However, these ideals can limit the effectiveness of a multiprocessor system for its application. Newman /NEWM82/ compares the long development time required for programs which must deal with the hardware considerations against the much shorter development time required for programs which use a high level language that conceals the hardware. The maximum speed of the system is, however, not normally obtainable when such a high level language is used.

The extensions to ordinary PL/M and PASCAL compilers that are required to convert them into concurrent high level languages are described by Bowen and Buhr /BOWE80/. It is noted that the small number of concurrent high level languages that do exist do not support multiple processors, and are mainly intended to be used with a single processing unit. Ardo and Philipson /ARDO84/ describe an implementation of a PASCAL-based parallel language specifically for a multiprocessor computer. The language is based on a standard
implementation of PASCAL with primitives added to support parallel programming.

Woodward /WOOD81a/ describes three basic organisations used for the design of operating systems for multiprocessors. They are the master-slave organisation; an organisation which maintains a separate executive for each processor; and an organisation which treats all processors anonymously or symmetrically. Master-slave organisation requires that all operating system routines are executed in the same processor (the master); a separate executive for each processor means that each one can operate independently; and anonymous treatment of all processors means that they are treated as any other resource, the tasks being shared out asynchronously between them. An operating system for INTEL 8086-based computers is INTEL RMX86/INTE82/. This is a real-time multitasking executive which provides routines for task control and synchronisation (task suspension, creation, deletion, semaphores, message passing etc.). The operating system is intended primarily for uni-processor use although the family of Intel's single board computers (for which the operating system is intended) does use a common bus called Multibus which does support multiprocessor applications.

Fielland and Rogers /FIEL85/ describe the use of a different bus suitable for multiprocessing, called the SB8000 system bus. The bus is described as one of the five principle areas which need to be carefully optimised to take full advantage of multiprocessor computers. The other four areas are the processor sub-system; the system memory; the I/O mechanisms and the operating system. A computer system whose architecture stresses parallel computing at two
major levels is the Denelcor HEP computer system described by Grit and McGraw /GRIT85/. The overall machine structure can support up to 16 separate processing units connected to shared memory, which is the first level of concurrency. Each processing unit can also simultaneously execute multiple instructions by exploiting an 8-stage instruction pipeline, which is the second level of concurrency.

2.7 Conclusion of chapter.

Recent literature in the areas of manufacturing system design; manufacturing systems simulation; simulation techniques; distributed simulation; and multi-computer processing systems has been reviewed in this chapter. Distributed simulation using multi-processing methods has been applied in the past mainly to the simulation of dynamical systems. There appears to be potential in also applying distributed simulation to the detailed simulation of manufacturing systems because of the user and computing requirements of detailed simulation of large and complex manufacturing system designs (rapidity, detail, support for simulation configuration and graphic output) and the potential benefits of distributed simulation (alleviating time and memory constraints, modularity). There also appears to be potential in applying multi-processing to the simulation of manufacturing systems because of the potential benefits of increased computer throughput, flexibility, and availability.

There have been some attempts to provide support for multi-processing (parallel processing) in standard computer languages. However, the use of a standard high level simulation or normal computer language is unlikely to really achieve the benefits of parallel processing because
it is high level and does not take full advantage of the hardware. In addition, a system which exhibits a high degree of concurrent activity does not necessarily entail the use of a simulator with a high degree of parallelism.

The problem areas to be addressed are thus: the provision of a distributed simulation system which offers support for generating distributed simulation programs in the area of highly automated batch manufacturing systems; and whether the techniques of distributed simulation and multi-processing offer advantages for the detailed simulation of highly automated batch manufacturing systems.
CHAPTER 3
MANUFACTURING SYSTEMS

3.1 Introduction

The purpose of this chapter is to introduce the reader to manufacturing systems, to the areas of performance for which different types of manufacturing systems are designed, and in particular performance measures which affect the design of flexible manufacturing systems. The performance measures which are relevant to FMS require sophisticated techniques of analysis to be used when designing FMS. One such technique is simulation, and the use of simulation during the process of manufacturing system design will be described in this chapter.

A manufacturing system 'is a unified assemblage of hardwares, which include workers, production facilities (including tools, jigs and fixtures), material handling equipment and other supplementary devices. This is supported by the software, which is production information referring to methods and technology'. The purpose of a manufacturing system is to convert 'raw materials into finished products, aiming at maximum productivity and efficiency' /HITO79/.
3.2 **Types of Manufacturing System.**

Manufacturing systems may be classified into five general types. These are: the dedicated system (i.e. a transfer line); the flexible manufacturing cell; the flexible manufacturing system; the unmanned station; and the job shop, i.e. a collection of individual machines. The range of application of each of these systems in terms of volume and variety is shown in figure 2.

The transfer line (figure 3) is designed to produce a high volume of manufactured parts at the lowest cost and is usually dedicated to producing a particular manufactured part for a very long period of time relative to the production time of a single part. Hence the variety of parts produced by this type of system is small and the quantities produced high. Examples include car manufacture and the manufacture of mass-produced goods such as domestic electrical appliances (television sets, washing machines, etc.). The cost involved in changing production from one type of part to another is usually high and only justifiable because of the large volume to be produced /BILA83a/.

Automated systems for batch manufacturing are described by the terms 'flexible manufacturing cell', 'flexible manufacturing system', and 'unmanned station'. Flexible manufacturing cells offer the least flexibility of the three types and are designed to produce a small range of manufactured parts. A typical automated flexible manufacturing cell would consist of a central server, such as a robot, loading and unloading parts and tools to and from a small number of local machines (figure 4).
Dedicated Systems

F.M.C.

F.M.S.

U.M.S.

Individual Machines

VOLUME

VARIETY

The Range of Application of Manufacturing Systems.

Figure 2

F.M.C. = Flexible Manufacturing Cell
F.M.S. = Flexible Manufacturing System
U.M.S. = Unmanned Machining Station

Machining Stations

Transfer Line

Figure 3 (After NUME85/)

- 28 -
Flexible Manufacturing Cell
(Rotational Components) Figure 4 (After /NUME85/)
The capabilities of the robot determine the configuration of the cell in terms of its layout and the complexity of the pattern of operation of the cell /NUME85/.

Flexible manufacturing systems include a range of processing capability and routing capability and are designed to produce varieties of parts in the range of 10 - 500 product variants (although some FMS's are reported to produce only 2 variants or as high as 1500 product variants) /EUCE86/. The degree of automation varies, but there are typically automated material handling devices (robots, automatic guided vehicles) and CNC machines into which the process instructions (the NC program) can be loaded from a central controlling computer. There is usually some capacity for parts storage at start, finish and intermediate stages of production. Control of the system is carried out through the central computer which can interface to other information systems in the factory. The limits on the flexibility of the system are usually tool quantities and supply mechanisms (i.e. tool management), material handling techniques, the types and number of pallets, and variable routing and organisational control (i.e. the real-time control capability).

An example of the layout of an FMS which consists of a number of inter-linked cells is shown in figure 5.

Unmanned stations are single units at which provisions exist for operating during (typically) one or two shifts without attention from an operator (figure 6).

This requires a suitable quantity of tooling, process monitoring (e.g. for tool breakages), and part storage and automated loading/unloading. Unmanned stations are designed to produce a fairly
Flexible Manufacturing System

Figure 5 (After WEBB86)

Unmanned Station

Figure 6
large variety of manufactured parts but in fairly low volumes.

A manufacturing system which is designed to produce the greatest variety of manufactured parts is the job shop. The machines in a job-shop are usually arranged in a functional layout (milling machines together, turning machines together etc.). The primary limitation of a job shop is due to the queues of work which form prior to each machine or work station. These queues represent waiting time between operations on a part which requires a number of operations on different machines. The waiting time in each queue is usually quoted to be one week, and hence the 'throughput time' or the time taken by a part which requires a number of operations to get through the whole shop can be long.

There has recently been a shift away from the installation of large, complete flexible manufacturing systems, due to the volatility of the manufacturing environment together with the large financial investment which is required in modern manufacturing equipment and staff (computers, CNC machines, networking, control software etc.). This volatility arises because of the uncertainty in manufacturing, which itself is due to a range of causes, among them unconfirmed future orders for products, forecasts of sales, changes in the financial environment (tax structures, special government programmes to introduce particular technologies), technological developments in products (requiring design work) and technological developments in manufacturing processes (requiring machine trials, process quality acceptance etc.).

These uncertainties mean that large, one-off installations of
automated systems for batch manufacturing (i.e. flexible manufacturing systems) are less attractive investments than a number of smaller individual investments in manufacturing cells which can be individually introduced and installed. One of the advances in technology which has made this possible is the increasing power of computing devices and the corresponding decrease in cost. This advance together with advances in computer networking means that automation, the distribution of control, and the linking (integration) of individual manufacturing cells has become possible and cost-effective.

This multi-cell operation (i.e. machines operating in a number of parallel cells) is consistent with step-wise investment and additionally, is consistent with the requirements of an integrated manufacturing control system, a particular example of which is a hierarchical control system /ANST86/. The organisation of this example is shown in figure 7.

3.3 Performance Measures.

Performance measures or evaluation criteria of a manufacturing system vary in emphasis depending on the type of manufacturing system to be evaluated. Typical examples of performance measures are: flexibility, output, investment, utilization, capacity, availability, quality (precision), system life, redundancy, compatibility, enlargeability, running costs. Costs (investment cost and running costs) are a fundamental performance measure which apply to all the different types of manufacturing systems. The relevance of the other measures varies depending on which type of manufacturing system is being evaluated.

- 33 -
BRITISH AEROSPACE - HIERARCHICAL CONTROL SYSTEM - Small Machined Parts FMS

Figure 7 (After /ANST86/)
A Transfer line would be evaluated primarily in terms of investment cost, running cost, output, availability (i.e. down time), capacity, and system life. A transfer line is designed to produce a specific part to a specification at a certain throughput rate or capacity. The utilisation of the individual manufacturing units is usually designed to be the same, or 'balanced'.

Similarly, the performance measures for an FM Cell are primarily throughput and availability but variety starts to enter into the criteria for the cell. 'Variety' means the number of different types of parts that the manufacturing cell has been designed to produce: the limitation on variety being the types of processes and suitability of the material handling devices.

The Unmanned station loses 'throughput' from its list of primary objectives, and gains 'utilisation', since an adequate amount of work and tools must be supplied to the station to keep it utilised. Another objective of the unmanned station is the length of time that the station can remain working unmanned (usually a third shift etc.). The factors that influence this are swarf removal, in-process gauging of work and tools (offsets must be able to be changed) and reliability of the machine.

The main criteria of a job shop are utilisation and throughput. Reliability or 'up time' is not so important since there are usually duplicate machines on which work can be processed. Utilisation is usually high if queues of work are maintained before (or at) each station but in this case throughput can be slow (as opposed to volume of output which can still be high). Maintaining a fast throughput time of items and achieving high utilisations of the machines is one
of the most difficult scheduling problems of the job shop.

3.3.1 FMS performance measures

The particular objective of a flexible manufacturing system is defined by Steinhilper in /STEI84/ as the ability of the manufacturing system to manage different manufacturing tasks. This flexibility can be divided into two types, short term flexibility and long term flexibility, which are also defined in the same reference as:

'Short term flexibility is expressed in the capability for fully automated resetting for new parts within the limits of foreseen and preprogrammed manufacturing tasks'; and

'Long term flexibility is related to the availability of workpiece-independent machining and material flow components which do not need to be replaced in the case of unforeseen workpiece changes'.

The objective of achieving fully automated resetting for new parts within the limits of foreseen and preprogrammed manufacturing task, which is also an objective of highly automated batch manufacturing systems, means that set-up time on each machine becomes relatively short and hence high productivity and high machine utilisation are possible without the high capital cost of the dedicated line. The performance requirements of an FMS to make this investment worthwhile are therefore:

- high capital equipment utilisation. Pre-fixturing of parts on pallets and efficient scheduling by the
controlling computer are necessary to achieve high utilisation.

- ability to maintain production. Graceful degradation following machine breakdown is required (bypass of failed machines, redundant machinery capability).

- high product quality. A reduction in rework and its associated scheduling problems is required to maintain an orderly and well-utilised system.

- reduced work-in-progress (and lead time). Efficient computer scheduling of parts batched into and within an FMS is required to reduce the quantity of work-in-progress and the number of pallets and fixtures required.

Particular influences on the performance of a highly automated batch manufacturing system such as an FMS are:

- the number of different routes which can be used to produce each part. Each route must be tested to ensure product quality can be achieved when that route is used. If the number of different routes is high, then a restriction is usually put on the system to limit the number of different routes. One way of achieving this is to restrict the number of pallets which can accept a certain part type and also to restrict certain pallets to certain machines. These restrictions can, however,
have side-effects on tooling capacities, utilisation of machines, production rates and the ratio of the production of parts.

- the sophistication of control hardware. An example of this is that transporter control rules must be sophisticated enough to resolve blocking of routes when congestion of the system arises /LENZ85a/.

The primary measures of performance of an FMS are the utilisation achieved of the manufacturing units, together with the amount of inventory and throughput. This is in contrast to the transfer line and FM Cell where the system, designed to produce a high number of similar items, is highly dependent on one operation supplying another (i.e. balanced), so that the utilisation of each manufacturing unit is the same and is in fact the utilisation of the whole system. This is also in contrast to the job shop, where inventory is usually kept at a high level in order to maintain the high utilisation of the manufacturing units, since the utilisation of each manufacturing unit is related to the level of inventory /LENZ86/.

3.4 Stages of manufacturing system design.

The most basic issue in the entire manufacturing system design and evaluation process is to decide which parts can be produced on which machines to maximise the cost saving compared with alternative production methods. Qualitative factors can also influence this decision but the primary concern is usually economic /CHAR84b/.
As previously indicated, the three basic steps in the design of a manufacturing system are: selecting parts and machines; design alternative configurations; and evaluating the alternative configurations (figure 8).

Either parts or machines may be selected first i.e. a family of parts may be chosen as the family to be manufactured using a new manufacturing system or an existing set of machines can be loaded with alternative families of parts.

If a family of parts is established first, then a set of machines which include a suitable range of processes must be chosen. Approximate capacities are used to establish the quantities of each type of machine and process required. Analytical modelling (using queuing models, for example) can be of great value during this stage to determine the capacity requirements of the proposed system.

If an existing set of machines is to be loaded with a new range of parts then the parts can be selected by examining how completely each part can be manufactured within the range of existing processes i.e. whether a part must leave the system to be processed specially elsewhere and then re-enter the system. A family of parts can thus be found whose manufacturing requirements are primarily located in the existing system.

A combination of both approaches for the design of FMS is described in /CHAR84c/ where a range of candidate parts and part families are established which have 'FMS compatible' attributes. These attributes are: desired machining cube; material; form; types of processes; tolerances; production quantity; machining time; and current number
Design Manufacturing System

Select Parts and Machines

Design Alternative Configurations

Evaluate Configurations

Choose Configuration

Some Steps of an FMS design procedure

Figure 8 (After /CHAR84/)

- 40 -
of fixturings.

The current cost of manufacture of each part is found and an estimated 'FMS manufacturing cost' is calculated for each part. Specific machines are then chosen and a set of parts which produce the maximum savings on the chosen machines are defined (from the candidate set). A potential system payback and return-on-investment can then be calculated. Different mixes of machines and parts may then be analysed until the most cost-effective solution is reached.

The design of a configuration of a manufacturing system consists of two steps: estimating the work content of the selected parts, and then designing several configurations of the equipment. Estimating the work content of the selected parts involves detailed process planning, tooling and fixturing requirements estimation. This data is then applied to specific combinations of machines, auxiliary equipment and material handling equipment. A range of candidate manufacturing system designs are then available.

The evaluation of the candidate manufacturing systems consists of the following steps: the construction of an evaluation matrix showing all the factors considered important to the evaluation of each candidate configuration; the development of operational strategies - batching, balancing, scheduling and dispatching rules; simulation of the operation of a particular configuration to provide performance measures for economic analysis; and economic analysis (estimating return-on-investment and payback period).
The economic analysis using throughput data etc., generated by the simulation model together with the evaluation matrix will indicate the particular configuration which should be chosen. Typical criteria used for the evaluation matrix include cost; throughput; availability; flexibility; precision; tool management; redundancy; and surge capacity.

Following the development of operational strategies the system performance measures can be obtained through the use of simulation. Queuing models or discrete events simulation system are used to model the manufacturing system.

3.5 Analytical Models.

Analytical modelling (using queuing theory, for example) is used at two points in these stages of manufacturing system design. The first stage is to determine the capacity requirements of the proposed system with a particular type or set of machine types. The result of applying modelling at this stage is to determine quantities of machines so that an initial cost of the proposed system can be calculated.

An example of this is described in /SOLB82/, where the objective was to design a manufacturing system where the size, complexity and processing requirements of five dissimilar groups of parts to be produced were known. Variations in parts mix could occur, but were unknown. The number or quantity of machines was found by, initially, calculating the predicted output of the smallest possible system which could process the mix of parts. The bottleneck in this system could thereby be identified, and then the quantity of that machine
incremented by one. The predicted output of the 'new' system was then calculated, and the 'new' bottleneck machine duplicated. This cycle was repeated until the overall mean utilisation of the machines was maximised.

The analytical model used to calculate the mean performance measures of the proposed manufacturing system was CAN-Q. Using this model, the effect of variations in other system parameters could be explored; for example, the effect of additional transport resources could be measured and 'trade-offs' could be made between production rate, inventory and flow time. The end result of using the model was the number of machines and transporters which were required to satisfy the various performance requirements (utilisation, production rate etc.) and hence an initial cost for the proposed system could be calculated.

The second point at which analytical models can be used to obtain system performance measures is when different candidate manufacturing systems are being evaluated. However, some details in these types of models are at present treated inadequately. Examples of these details are: intermediate or buffer storage, use of shared resources (tools and fixtures) and scheduling and breakdowns. This is because these types of models take a steady-state point of view of the manufacturing system, where the outputs of one component become the input of another so that the interaction between components is well defined. This provides a view at 'optimal' steady state conditions. This simple relationship between components is, however, not always consistent with the philosophy of FMS and it is important, during the detail design stage, to study inter-component relationships to obtain realistic performance measures /LENZ85a/.
3.6 Simulation.

The point at which discrete event simulation is particularly appropriate is at the detail design stage, when different candidate manufacturing systems are being evaluated, having developed some operational strategies. The main objective is to study the operation of the FMS to identify how much congestion will result from integrating all the components of the proposed system together in a dynamic way. Simulation is most effective when the proposed system has been previously 'sized' for capacity and throughput so that a starting point for evaluation exists. An example of the detail which can best be studied with discrete event simulation and is necessary to study at this stage of design is the individual motion of pallets in the transportation system (when Automatic Guided Vehicles are used, for example) so that the interference of vehicles and algorithms for traffic control can be studied. The data and strategies which must be included in a model of the proposed system include part data, machine processing time data, system data and batching, balancing, scheduling and dispatch strategies.

Generalised simulation packages can be used, or a simulation model may be individually developed. Pre-written, generalised packages usually deal with a particular class of manufacturing system linked with a specific type of material handling system (e.g. conveyor, tow-line carts, or carts on rails). The simulation model should include the ability to model the effects of failures. However, building "too detailed a model can be both expensive and [the model] complicated to use" /CHAR84e/. 

- 44 -
3.7 Conclusion of chapter.

This chapter has briefly introduced the different types of manufacturing systems and described the criteria against which the performance of these manufacturing systems may be measured. A typical design procedure for a flexible manufacturing system has been described in which discrete event simulation is necessary to evaluate the performance of the proposed systems during the detail design stage. Simulation is necessary because of the complex nature of highly automated batch manufacturing systems (and in particular FMS), lying somewhere in application between the job shop and the transfer line, and also because of the complex effects of integrating automated equipment. The following chapter will describe discrete event simulation techniques in more detail.
CHAPTER 4

SIMULATION OF MANUFACTURING SYSTEMS

4.1 Introduction

In the preceding chapter, the application of discrete event simulation to the detail design phase of highly automated batch manufacturing systems was described. The purpose of this chapter is to describe in more detail the techniques of simulation which are applicable to the modelling of manufacturing systems.

A computer simulation system consists of a model and some associated programs which are interfaces to the user. These interfaces enable the user to collect results, input data for the simulation model and to configure a simulation model for a particular application, i.e. to simulate a particular candidate manufacturing system at the detail design phase.

4.2 Configuration

Simulation models may be constructed in two ways. The first is by configuring a 'pre-written' model which simulates a certain class of problem. The second is by constructing a new model using a computer language. The computer language may provide pre-defined constructs
for simulation (time advance, sets, trace variables etc.) in which case it would be called a simulation language, or it may be an ordinary high level computer language.

Some simulation languages are also designed to address a particular class of problem (e.g. ACSL, which is used for modelling and analyzing continuous systems /CATA86/). Some simulation languages also provide a 'pre-processor' which may use an interactive method to generate a program written in the simulation language.

Examples of these pre-processors are 'DRAFT' which produces FORTRAN code for the simulation language/system 'SIMON' /MATH84/. 'CAPS' (Computer Aided Programming of Simulation) produces code for the 'ECSL' simulation language/system /CLEM82/. Another is 'EXPRESS' /SCHA85/ which provides a front-end pre-processor for the 'SEE-WHY' simulation system. The use of these pre-processors can reduce what can be a lengthy procedure of building models of complex systems. However, as noted by Clementson /CLEM82/, 'the interactive program (CAPS) is not capable of accepting every detail of every simulation model but should be capable of building at least the skeleton of all simulation programs'. The few activities which contain special or unique features must still be added to the program by using the standard file editing facilities of the computer system being used. Thus the analyst who builds the model must be capable of understanding the code generated by the pre-processor and of enhancing this code to take into account the complexities inherent in a particular system.

Models which simulate a certain class of problem are usually data-driven, in that by specifying in a data section the quantity and configuration of a particular entity or entities, the model is then
set up to simulate a particular system. These models can reduce the
time required to build a model since only the data section has to be
constructed (this would have to be done in any case when using a
simulation language). The main disadvantage of these pre-configured
types of models is that it can be very difficult to add special
features to the model (particular control rules, etc.) that are not
already provided.

4.3 Data Requirements

There is a variety of input data required for the discrete event
simulation of highly automated batch manufacturing systems. Ranky
/RANK83a/ lists some:

- Manufacturing time distributions (the time each part
spends being processed at each work station)
- Inspection time distributions, washing time
distributions etc.;
- Pallet Transportation times;
- Fixturing time distributions; and
- Unloading time distributions.

The time distributions can be statistical distributions or actual
machining data. The amount and accuracy of data required increases as
the system is simulated in more and more detail.

Manufacturing time distributions (or parts data) in the case of an
ECSL model of a small manufacturing system /BILA83b/ were obtained
from parts list standard route card information and machine processing
times for NC machining. This data included: batch type identity;
batch size; required number of chuckings on a machine tool; first choice operation sequence; second choice operation sequence; machining operation times; the number of parts carried on a pallet; the type of pallet required; the priority assigned to the batch; and the time of release into the system.

This type of data describes inputs into the modelled system which vary from day to day i.e. the parts which are to be machined by the manufacturing system. Other more permanent data relates to the configuration of the system and its resources i.e. data relating to machine tools - tool magazine capacity; reliability; maximum unattended working time, etc. There is, in addition, permanent data relating to the materials handling system: i.e. the load and unload mechanisms for the machines (robots, automatic pallet changers); data relating to the storage systems (capacity, speed of locating a pallet etc.); data relating to manual labour in the system (availability for work, technical ability etc.); and data relating to pallets (the type of part that each pallet holds, its capacity etc.).

Other data of this type relates to the transport system: speed of transporter; capacity of transporter; restriction on routes; etc. This data does not change as the model operates unless new resources (machines, tools etc.) are added to the modelled system.

Additional data may be required for the presentation of output following an experiment using the model: an example of this is data relating to the spatial relationship of the elements of the system if graphical animation of the simulated operation of the system is required.
In a simulation model used at the detail design stage of highly automated batch manufacturing systems, a large amount of data is used. This data must be well managed and organised to ensure that the data can be efficiently input into the model, efficiently used by the model, and is capable of being changed without complete reprogramming of the simulation model.

4.3.1 Data Input

The ease and style of data input depends on the simulation software to be used and on the hardware available. In a model written in the ECSL simulation language, the data for a particular simulation model and run of the model is a list of data items in a text file immediately following the 'statement' text of the model. The model is compiled together with its particular data and then executed. This requires a minimum of hardware (i.e. an alphanumeric terminal). The data cannot usually be changed as the model is executing (i.e. the model is non-interactive), and a change in data would mean recompiling the model. However the same data can easily be used in more than one model of a similar type just by copying the data block to the text file of the new model.

A model built using the SEE-WHY simulation system /ISTEOO/ relies on data files but interactive changes to data can be made by stopping the execution of the model (by interrupting) and changing, for example, the number of parts in the system, or adding an operation.

4.3.2 Data Management

The large amount of data required for detailed simulation models can best be managed using a database system to provide fast access to
files or records of data. A database system provides an interface between the simulation model and the physical arrangement of the data and the data handling techniques for the management of data (searching for particular items, etc.). One example of a simulation system which uses this method or technique is the OPTIK suite of software /INS100/. The three major items in this suite are a relational database, an interactive graphics package, and simulation software providing the basic constructs for entity creation, queue handling and an executive for time advancement.

The SIMAN simulation language can be used to build a model of a system and a particular experiment is then carried out using the model. The data for the particular experiment is entered into an 'experimental frame' using an experimental frame processor which checks the data for correct syntax. This data is held in a separate text file from the model and is associated to a particular model by the use of a 'linker' program /PEGD84a/.

4.4 Simulation Techniques

Simulation systems which simulate a particular class of problem and which are data-driven have an underlying technique or approach to simulation embedded within them which cannot be influenced by the user of the simulation system. Simulation languages also employ different approaches to simulation, and the user of a simulation language can choose to use a particular approach to simulation by using a particular language. Each approach to simulation has its particular characteristics summarised in /HOOP86/. The four main approaches to discrete event simulation are:
(i) the event approach;
(ii) the activity approach;
(iii) the process-interaction approach; and
(iv) the three-phase approach;

Each simulation language based on one of these four approaches includes an executive which performs a sequential search and a number of tests depending on which modelling approach has been used /PIDD84b/.

4.4.1 The Event Approach

The event approach models a system in the following way:

An instant in time where the state of the system changes is called an event time. The logic associated with this change is called an event. Time does not advance within an event and the system behaviour is simulated by state changes that occur as events happen. The event scheduling approach has the characteristics of efficient execution for models with relatively independent components; it is very flexible but considerable model development effort is usually required.

The simulation executive which is related to an event-based approach searches through an 'event-list' to determine which events are due to begin. This search is known as a 'time scan'.

4.4.2 The Activity Approach

The activity approach to simulation models an activity which occurs over time by using two related events. The first event models the changes that occur at the start of the activity, and then schedules
the second event to occur at the end of the activity' /PEGD84/. An example of this approach is a machining activity. The first event models the start of the machining cycle (machine busy, part being processed) and the second event models the end of the machining cycle (machine becomes free, part returns into queue for next machine).

The activity approach has the characteristics of efficient execution for models with highly dependent components (i.e. when many conditions are required to permit an activity to start); considerable work is done by the simulation executive; and condition testing is localised and not scattered throughout the model.

An activity based executive carries out a 'time scan' which is similar to that carried out by an event-based executive, and, in addition, a repeated activity scan involving tests at the head of each activity to determine whether that activity can start.

4.4.3 The Process-interaction Approach

The process interaction approach to simulation involves the construction of a process for each temporary entity within a model /PIDD84a/. A process is defined as the sum total of activities and events that the entity passes through whilst it is within the system. It is a combination of the event and activity approaches to modelling.

The process-interaction approach has the characteristics that the simulation executive provides a lot of support; model development and modification can be straightforward; the model representation is 'close to the problem'; but complexities can be masked from the modeller and it can be inflexible and be inefficient in execution time.
A process based executive involves a 'time scan' and a more complex 'current events' scan, where each process is attempted and executed until the process cannot continue. This point in the life of the process is stored and the process and time scan begins again.

4.4.4 The Three-phase Approach

The three phase approach to simulation /TOCH63/ divides the activities into two types viz. bound and conditional activities. A bound activity is executed directly by the executive program whenever its scheduled time is reached. The execution of a conditional activity is dependent on the satisfaction of specific conditions within the simulation. The repeated activity scan of the executive involves only the conditional activities. This technique reduces the amount of conditional testing carried out by the executive.

The effect of integration of equipment in a highly automated batch manufacturing system, such as a flexible manufacturing system, means that the activity approach to simulation is particularly suitable. This is because of the localisation of condition testing using the executive as in the three-phase approach. However, if a lot of condition testing must be carried out this executive can be inefficient and must carry out a lot of work.

4.5 Performance

There are several criteria by which one may wish to assess the performance of a simulation language or system. Two main parameters used to assess the performance of generalised simulators for manufacturing systems are described by Mills and Tavalage /MILL85/. Firstly, the 'level of generality': this relates to the range of
systems that the generalised model is capable of handling. The second parameter is 'ease-of-use', which relates to speed of operation, the need for specialist computing knowledge, and the simplicity of model building.

Cellier /CELL83/ shows that the measures of performance by which the features and performance offered by a variety of simulation languages and packages can be compared are:

- expressiveness of the language;
- numerical behaviour;
- structural features (application program development facilities, program validation and verification, program execution, data handling, input/output);
- status of implementation;
- portability; and
- documentation.

The weight given to each of these criteria will vary depending on the environment in which the simulation package or language is to be run and what it is to be used for.

4.5.1 Detailed Simulation

There are many levels of detail at which a simulation model can be built. For example, modelling a manufacturing system in order to investigate the overall balancing of the system (viz. the number of machines required to produce a number of manufactured parts in a given time period) can be achieved using a simulation model or an analytical model.
A model for this purpose may well contain assumptions: examples of assumptions in this case may be that each machine has appropriate tooling available; similarly demand patterns, processing times etc. may be sampled from distributions. Alternatively, a simulation model with a high level of detail would be used to investigate the effect of complex interacting parameters such as part release rules, control algorithms and tool supply. In such a highly detailed model for this purpose the minimum of assumptions are made and a minimum of detail is excluded.

A model that contains a higher level of detail does not usually require reconstruction to investigate alternative variables because the variables are usually already included in the model, whereas with a less detailed model reconstruction is usually required to investigate different sets of variables. The maximum level of detail in a simulation model is achieved when all the variables that it is possible to measure are included for a particular computational vehicle.

Considerable effort is also required to build a simulation model to achieve detailed modelling of a system, and as the complexity of the model increases, the response time of the simulation model deteriorates (the time to build the model and to execute it). The level of detail that it is possible to include in a simulation model thus depends on the computational vehicle to be used, the effort to be expended whilst building the model and the response time required. The effort expended whilst building the model can usually be reduced by the use of aids to configuration.
4.5.2 Performance requirements for detailed simulation.

For detail design of a proposed manufacturing system several features are of primary importance. Firstly, the level of detail that it is possible to include within the model - this is related to the space available to store data and program source during compilation and also related to the ability of the analyst to deal with large and unwieldy programs.

Secondly, speed of model execution - the long execution times often required for detailed models do not encourage the testing of many variables within a model purely because of constraints on the analyst's time. The shorter the response time is (from start to completion of a run) and the nearer it is to an 'interactive' response rather than a batch-process type response (tens of minutes to hours), the easier it becomes to test the effects of more variables.

Thirdly, a model that is relatively easy to build (the use of standard simulation approaches, a clear and modular structure, a database for data storage and handling etc.) becomes particularly important for detail design work because of the length of time taken to develop a range of configurations of the model and to validate and verify the model. The facilities available for model verification and validation also become particularly important in a complex model because the interactions between events becomes difficult to predict and unforseen effects of these interactions often occur. The ability to verify whether the model is operating correctly under these circumstances is obviously important.
4.6 Conclusion of chapter.

This chapter has described how simulation systems and languages can be configured to model a particular situation or manufacturing system. It is difficult to configure a pre-written simulation model for a particular detailed circumstance or application because of the problems of introducing particular control rules into a pre-written model. It can also be difficult to use a 'front-end' to a simulation language which generates code in the language because of the necessity, in a detailed model, of modifying detailed sections of the code so produced.

Examples of the data required in a simulation model of a manufacturing system and the way in which a particular set of data can be associated with a particular model have been described. The amount of data associated with a detailed simulation model requires a good data structure and management.

The most common approaches to simulation have been reviewed. The technique of simulation using the activity approach is particularly appropriate to the simulation of highly automated batch manufacturing systems because of the inter-dependence of the components in such a manufacturing system.

The particular performance measures of a detailed simulation model have been described, which are: the level of detail of the model; the speed of model execution; the ease-of-use (the ease of building and configuring a model); and what facilities are provided to verify and validate a suitable model.
The following chapter will describe a particular method of achieving this type of simulation performance using a discrete event simulation model. The method is that of distributed simulation, which is concerned with taking the simulation of a physical process and distributing it over a number of logical processes and/or a number of computer processors /REYN85/.
CHAPTER 5

DISTRIBUTED SIMULATION SYSTEMS.

5.1 Introduction

In the preceding chapter, the criteria which are used to evaluate the performance of a discrete event simulation model for the detailed simulation of highly automated batch manufacturing systems were described. The main criteria are: the speed of execution of the model; the level of detail that it is possible to include within the model; the ease-of-use of the simulation model; and the facilities available for model verification and validation.

The purpose of this chapter is to review the technique of distributed simulation and to examine how it may be applied to the detailed simulation of highly automated batch manufacturing systems in order to satisfy these criteria.

Distributed simulation is concerned with taking the simulation of a physical process and distributing it over a number of logical processes and/or a number of computer processors /REYN85/. Two primary goals of existing distributed simulation systems are to increase the efficiency of simulation and to sub-divide a large
The goal of increasing the efficiency of simulation (and therefore satisfying the criteria of increasing the speed of execution of the model) can be achieved by producing algorithms to process the 'events-list' in a parallel manner.

The executive of a discrete event simulation model which uses the event or activity approach to simulation, searches through an 'events-list' to determine which events and activities are due to begin. This 'events-list' is required to effect synchronisation between the natural order of occurrence of events. A sequential algorithm is required to do manipulations on the events-list and although considerable effort has been expended to reduce the cost of these manipulations, a definitive solution is still awaited /GHOS85/.

The efficiency of general-purpose simulation languages is limited by the efficiency of the algorithm to process or manipulate the events-list, and one primary concern of distributed simulation techniques is to implement such an algorithm in a parallel manner. Algorithms for distributed simulation are aimed at breaking away from the sequential nature of processing events so that parallelism inherent to systems can be captured: this should increase the efficiency of the list-processing algorithm and hence increase the computational speed and therefore the speed of execution of the simulation model.

The goal in distributed simulation of sub-dividing the simulation program into autonomous modules may be achieved by defining modules by functionality or in the form of sub-models or elements of the total system to be used. The nature of the modules defines the level of
detail which is included within the model. Sub-division of a large simulation model in this way can enable individual elements of a system to be more easily modelled and hence an increase in the level of detail within the model can be achieved.

Each sub-model, representing a set of particular elements of the system, must be correctly linked to others in order to produce a larger model. The requirements for the correct linking of these sub-models are that the interface to the module must be designed in a way that is acceptable to all possible users, and the interface should be such that the user does not need to know how the sub-model works /CLEM85/.

The motivation for developing modular or distributed simulation modules in this way is described as 'enabling a considerable reduction in development time for large simulation models'. Another motivation described by Clementson /CLEM85/ for developing modular simulation is to allow simulation to gain speed by the use of distributed computing. This could lead to real-time simulation being a step along the route to the development of an FMS control system. The off-loading of graphics functions (for animated display, etc.) is described as a function which can be moved to a separate computer to enable an increase in execution speed of the model.

5.2 Examples of current distributed simulators

The purpose of this section is to review some existing distributed simulation systems and to examine how the criteria described above are met by existing distributed simulation systems.
Most of the simulation systems to be examined here are not specifically designed for use with manufacturing system problems. This means that entities found in manufacturing systems are not pre-defined. The simulators are directed to general use, or for continuous system modelling, or for other specific problems. A summary of the distributed simulators which are reviewed is presented in table 1.

The simulation software designed for use with distributed multiprocessor computing systems falls into three main categories: that which is implemented on a multitasking uni-processor computer system; that implemented on a true multiprocessor system; and that which is proposed.

5.3 Multitasking Simulation Systems.

A multitasking uni-processor computer system is where the single central processor divides its time between a number of different tasks. These tasks may be time-shared i.e. a fixed amount of CPU time is allocated to each task or controlled by some sort of monitor which monitors the progress of each task and suspends a task when required. However, the execution of each task proceeds in a sequential manner and the control of such a system is sequential (although the execution of a number of tasks may appear to the user to occur simultaneously). This type of system is unable to exploit the parallelism inherent in many simulations to improve efficiency since it is essentially a sequential processing system.
1. Multi-tasking

<table>
<thead>
<tr>
<th>Name of author</th>
<th>Name of simulator</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bruno, G. &amp; Canuto, E.</td>
<td>DESFOR</td>
<td>/BRUN82/</td>
</tr>
<tr>
<td>Livny, M.</td>
<td>DISS</td>
<td>/LIVN85/</td>
</tr>
<tr>
<td>Wyatt, D.L.</td>
<td>SIMPAS</td>
<td>/NYAT85/</td>
</tr>
</tbody>
</table>

Features extensions to FORTRAN to allow quasi-parallel programming
Features discrete event simulation based on SIMSCRIPT II.5
Features tasks allocated via simulation support function

2. Multiprocessor

<table>
<thead>
<tr>
<th>Name of author</th>
<th>Name of simulator</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barel, M.</td>
<td>DESC</td>
<td>/BARE85/</td>
</tr>
<tr>
<td>Blech, R.A. &amp; Arpasl, D.J.</td>
<td>RTMPS</td>
<td>/BLEC85/</td>
</tr>
<tr>
<td>Deans, N.D. &amp; Mann, D.P.</td>
<td>----</td>
<td>/DEAN83/</td>
</tr>
<tr>
<td>Korn, G.A.</td>
<td>DARE</td>
<td>/KORN72/</td>
</tr>
<tr>
<td>Reed, D.A.</td>
<td>----</td>
<td>/REED85/</td>
</tr>
<tr>
<td>Roehdar, W. et al</td>
<td>----</td>
<td>/ROEH82/</td>
</tr>
</tbody>
</table>

Features tasks allocated via simulation support function
Features a real-time aircraft simulator
Features tasks allocated by function (not discrete event simulation)
Features solution of differential equations
Features simulation on a CRAY-XMP - a case study of VLSI circuit simulation
Features a flexible multiprocessor system for simulation purposes

3. Proposed

<table>
<thead>
<tr>
<th>Name of author</th>
<th>Name of simulator</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cellier, F.E. &amp; Rimvall, M.</td>
<td>----</td>
<td>/CELL85/</td>
</tr>
<tr>
<td>Concepcion, A.I.</td>
<td>DEVS</td>
<td>/CONC85/</td>
</tr>
<tr>
<td>Ghosh, J.B.</td>
<td>----</td>
<td>/GHOS85/</td>
</tr>
<tr>
<td>Singh, M.G. &amp; Alidina, A.Y.</td>
<td>----</td>
<td>/SING85/</td>
</tr>
</tbody>
</table>

Features tasks allocated via simulation support function
Features a formal specification for multiprocessor simulation
Features a proposal for the asynchronous control of simulation
Features the simulation of high order differential equations

Examples of current distributed simulators

Table 1
Distributed simulation systems which are implemented on a multi-tasking uni-processor computer system are mainly intended to achieve the goal of sub-dividing the simulation model into modules. This is because a multi-tasking processing system is essentially a sequential processing system and hence the actual parallelism of the modules cannot be exploited. This type of simulation system has been used to satisfy the criteria of increasing the level of detail within the model, and increasing the ease-of-use of the model.

An example of a simulation system which falls into this category of multitasking systems is DISS/LIVN85/ which stands for a 'Distributed System Simulation modelling and simulation methodology' and includes the DISS simulation language which is a high level language (which is a macro-extension of the SimscriptII.5 simulation language). This simulation system was built to investigate how the goal of sub-dividing a problem into modules should be achieved, by measuring the inherent parallelism of a particular configuration of processors and modules.

The scheduling mechanism for DISS is meant to aspire to minimize the execution time of the simulation by maximizing the concurrent execution of events on different processors. However, the DISS simulation language has been designed for a uni-processor environment and thus only one process can be active at a given time. To enable the execution of events on different processors the centralized scheduling mechanism of DISS would have to be replaced by a distributed one.
DISS is used in this particular study to measure the inherent parallelism of different types of simulations to enable the calculation of an upper bound on to the effectiveness of a distributed execution of the simulation. The types of simulators which have been studied by the use of DISS include a Distributed Database simulator; a broadcast load balancing simulator; a store and forward communication network; and a routing and connecting simulator. The result of measuring the inherent parallelism (IP) of these different simulators shows that increasing the number of processors beyond half the number of processes has only a marginal effect on the inherent parallelism unless each process is dedicated to a particular processor. However, the value of the IP was strongly dependent on the values assigned to the input parameters.

Another multitasking simulation system is DESFOR/BRUN82/ which is a FORTRAN based package oriented to the simulation of discontinuous systems. It uses the process-interaction approach to simulation, and it is a collection of FORTRAN sub-routines together with two assembly routines. The extensions to FORTRAN allow quasi-parallel programming (the decomposition of the simulation program into quasi-parallel processes or coroutines); additional FORTRAN routines for random number generation and statistical data collection; and predefined constructs for modelling the interactions among processes (such as competition for limited resources etc.);

This is also a simulation system which has been designed with the goal of sub-dividing the simulation model into modules (such as separate modules for random number generation and statistical data collection). In this way the level of detail within the modules is
increased and the ease-of-use of the system is also increased (because routines for specific purposes are collected in particular modules). DESFOR is designed for use with production systems and standard components of these systems are modelled, such as workstations and storage areas.

Another multitasking simulation system is based on SIMPAS /WYAT85/, which has been designed to sub-divide the simulation problem into six modules, or tasks. These six distinct modules or tasks, which are the output of the SIMPAS preprocessor are: a supervisor; the event routines; a filing system; data input/output; statistics collection; and random number generation (figure 9).

SIMPAS is a commercially available PASCAL-based simulation language. It is event-orientated and is implemented as a preprocessor which accepts as input an extended version of PASCAL and produces as output a standard PASCAL program. SIMPAS has facilities to define entities, declare events routines, manage event execution and queues, generate random numbers and collect statistics. SIMPAS (the preprocessor) is itself written in standard PASCAL, and requires modification to enable a multi-tasking version to be implemented.

The tasks are allocated by simulation support function as opposed to being allocated via model function. All the data is declared as global, with no data local to any given program and the supervisory task must be manually changed to insert the program names or identification when the five other tasks have been compiled. The modified preprocessor was designed to be used on a multi-microprocessor network but was actually implemented on a uni-processor system (a Texas 990/12 DX 10) to allow the
A Distributed Simulation System with Tasks allocated by Simulation Support Function.

Figure 9 (After WYAT85/ )
preprocessor to be developed without the debugging problems one would find in a multi-micro network. However, care was taken during development of the preprocessor to ensure that it did not take particular advantage of the single processor architecture so that the multitasked system would be a true emulation of a multi-micro system.

A further example of a distributed simulation where the modules are divided or defined by function is described by Cellier and Rimvall /CELL85/ where the subtasks of the simulation (those designed to run in parallel) are, typically, the event routines in one module; a filing system in another; data i/o in another; statistics collection in another; and a random number generator in another. Cellier and Rimvall envisage moving all graphics generation into one task, which is interfaced to the others via a data area in a data base. The motivation in this case was to structure the simulation system so that the individual program modules remained manageably small and to allow several people to interact with different portions of the software system in a basically independent manner.

These simulation systems which are implemented on multi-tasking computer systems are designed to sub-divide the simulation problem into modules. This can have the effect of increasing the level of detail within the modules and of increasing the ease-of-use of the system by clearly locating statistic generation routines, for example, within one module or task.

5.4 Multi-processor Simulation Systems.

There are some distributed simulators actually implemented on multi-microprocessor computers. Among these are DESC /BARE85/ which
is a Discrete Event Simulation Computer. The objective of this simulation system is to improve simulation performance through an exploitation of parallelism and hence is concerned with sub-dividing the simulation problem into modules. It is also concerned with algorithms for list-processing to improve the speed of execution of the simulation model.

This computer consists of several processing elements dedicated to certain tasks. A front-end general purpose computer using a Z8000 processor with the UNIX operating system serves as the system manager of the DE5C, and provides programming facilities for the user; compilation of simulation programs; archival storage; links to other computers; and maintenance and development tools. The processors which are linked to this system management processor are a Simulation Executive Unit; a List Processor Unit; a Random number Generator; a Statistical analysis unit; and a Graphic display unit. The simulation tasks are therefore distributed according to their simulation support function and are implemented in this way to be applicable to a large number of simulation packages, such as SIMULA, SIMSCRIPT II.5 and GPSS which use the process interaction approach to simulation.

Another distributed simulator is described by Reed /REED85/ which is an application of an algorithm for distributed simulation developed by Chandy and Misra on a two-processor Cray-XMP computer. This simulator is mainly concerned with the criteria of speed of execution of the model because 'evaluating a detailed discrete event simulation model of a large scale system can be computationally taxing [and] parallel simulation provides hope of reducing computation time to tractable
levels'.

The Cray-XMP computer, however, is a vector processor and it has been proved by Chandak /CHAN83/ that discrete event simulation models cannot always be vectorized: in particular, a model of a network of queues containing feedback is not vectorizable. A distributed simulation model using the Chandy-Misra algorithm was written in FORTRAN in order to assess the performance of the algorithm but the attempt was unsuccessful particularly because the Cray Fortran compiler is unable to vectorize any loops containing 'If' statements and most of the loops in the model contained 'If' statements. This means that the code generated would run in a sequential manner hence there was no evidence of speed-up due to parallel execution of the simulation model. This problem, however, is a limitation of the compiler on the particular computing machine.

Other multiprocessor distributed simulation systems are designed to solve a particular class of problem, namely the solution of high order differential equations. An example of this type of simulator is that reported by Roehdar, Ameling and Lange at Aachen /ROEH82/. This simulator is also concerned with achieving shorter simulation times at a better price/performance ratio and is therefore concerned with the criteria of speed of execution.

The computing vehicle is called an M5PS (Modular-Multi- Microprocessor System) which is an array of subsystems (each subsystem being one microprocessor system) which have up to 8 high speed data channels each of which can be connected to adjacent subsystems. The M5PS may be used for SIMD (single instruction - multiple data) operation, or MIMD (multiple instruction - multiple
data) operation, or a combination of both (see chapter 6.6 for clarification of SIMD and MIMD). The flexibility of interconnection between the processor systems in hardware has shifted many problems, however, into software and to realise the operating system problems have had to be solved which are usually only apparent on large mainframe computers.

A multi-microprocessor system designed to simulate a system with emphasis on reliability aspects is a hardware reliability simulator developed at Aberdeen and reported by Deans and Mann /DEAN83/. This simulator is concerned with the criteria of speed of execution because analysis of the type of systems simulated 'carried out on a digital computer with a von-Neumann architecture necessitates long computational times'. The simulator is also concerned with the criteria of level-of-detail because the 'investigation of systems whose behaviour cannot be accurately modelled by exponential distributions .... presents difficulty.'

The technique relies upon the generation of random variables with known statistical distributions to describe particular properties of the components which make up the system. These properties are allowed to 'interact' in accordance with a mathematical model describing the overall system. The simulator is divided into four main modules and, in addition, one module for each component in the system. All the modules are implemented on separate microprocessors connected by a common bus, to a host computer and a microprocessor based data transfer handler and bus arbiter. The four main modules are a repair maintenance policy module; a network specification module; a statistic gathering module; and a module for simulation control
Each component module represents a particular aspect of the system to be modelled which, in its simplest form, may represent a generator or a relay. It is also possible to represent more 'abstract' system aspects such as computer software or human behaviour in a component module. The simulation is set up by an interactive definition of the components and the system. The parameters which are required for each component are: the time quantisation values to be used; whether an age replacement policy is to be used and if so, the age replacement time; whether a block replacement policy is to be used and if so, its replacement time; probabilities of start-up failure; start-up delays; the repair and failure policy; and the initial age of the component. The topology of the success tree using an interactive graphics terminal and the points in the network at data are specified. The simulation is then executed for a predetermined time.

Other simulators which are designed to be executed on multiprocessor computers are those which simulate the operation of, for example, aircraft in real-time, which are called flight simulators and are used for training purposes etc. Other simulators of aircraft propulsion systems are used to evaluate new control systems designs. These applications of real-time simulations require simulators that are portable and cost effective and hence are concerned with the criteria of level-of-detail and speed of execution. Distributed simulators using multi-processor computers can offer sufficient computational throughput while having cost and portable benefits over larger main frame computers. One such multiprocessor simulator for these aerospace-type applications is described by Blech and Arpasi /BLEC85/ which uses Motorola VM02 microcomputer boards connected via a
The image displays two diagrams related to hardware reliability simulation and parallel computing.

**Block Diagram of Hardware Reliability Simulator**

- **VDU**
- **Micro-processor based data handler plus bus controller**
- **Host Computer**
- **Components**
- **Control**
- **Time increment bus**
- **Simulation control**
- **Statistical gathering module**
- **Network specification module**
- **Repair maintenance policy module**
- **Address / Control**
- **Common Bus**

**Figure 10 (After /DEAN83/)**

**Proposed Parallel Computer for Simulation**

- **Co-ordinator**
- **Co-ordination iterations**
- **Subprocessors solving independent sub-problems**

**Figure 11 (After /SING85/)**
dual-bus to avoid congestion of the bus due to frequent interprocessor communication.

5.5 Proposed distributed simulation systems.

Other proposals for distributed simulation systems are described by, for example, Cellier and Rimvall /CELL85/. They describe a concept of structuring simulation systems so that individual program modules remain manageable small. This concept is designed in response to the large size of compiler programs required to offer any reasonable number of features in a simulation software system. The system proposed is concerned with the criteria of level of detail and also ease-of-use of the simulation modules.

The simulation task is intended to be decomposed into a series of simpler tasks each of which are basically independent of each other and each of which is kept sufficiently small to make it implementable, maintainable and updatable, the interface between the modules being through a database (or information base). The types of modules being considered include graphics input/output, data analysis modules etc. i.e. distribution of tasks via simulation support function.

Another proposal for simulation and optimisation of dynamic equations is described by Singh and Allidina /SING85/. This proposed simulator is concerned with 'improving simulation speed using parallel computation' and is therefore concerned with the criteria of speed of execution (figure 11).

The proposed system is designed to be used for problems where the essential features are that: most of the computational effort is
concentrated in the sub-problems, with the coordinator doing relatively little work; the amount of data transferred between the coordinator and sub-problems is very small compared to the amount of computing required to solve the subproblems; the execution time for each sub-problem is approximately equal, ensuring that processors are not idle; and that the decomposition coordination schemes converge to a solution of the overall problem.

A computer architecture to carry out simulation in a distributed manner is described by Concepcion /CONC85/. This proposal is concerned with sub-dividing the simulation problem into modules. The design of the computer is proposed to have advantages over other distributed simulators, which are: a flexible architecture; simplicity in design and organisation; exploitation of parallelism and concurrent execution resemblance to the hierarchical structure of a model; and reduction in contention for control of resources due to the multi-bus architecture.

5.6 Conclusion of Chapter.

It can be concluded from the descriptions of distributed simulation systems which have been reviewed in this chapter that multitasking distributed simulation systems have been used to investigate how a large simulation problem can be sub-divided into modules to enable the criteria of increasing the level of detail within the simulation model to be satisfied. The multiprocessing simulation systems reviewed have been concerned also with the criteria of speed of execution and increasing the level of detail by sub-dividing the problem into modules.
However, the distribution of the modules has occurred mainly by simulation support function i.e. random number generation, statistical data generation, graphic output etc., because of the nature of application of the simulators reviewed here (either general purpose, for simulating high-order differential equations, or one specific case of hardware reliability). It is apparent that a distributed simulator for manufacturing systems has not been developed (within the limits of the review conducted for this thesis). It is the intention in this thesis to show how distributed simulation which is concerned with all of the four main criteria described in the introduction to this chapter can be applied to a specific case of simulation i.e. that of highly automated batch manufacturing systems and to show that the specificity of the case enables advantages in modularisation to be exploited.
6.1 Introduction

The previous chapter reviewed existing distributed simulation systems with a view to the systems satisfying the criteria of increasing the level of detail in the simulation, increasing the speed of execution of the simulation, the ease-of-use of the system, and the techniques available for model validation and verification. The two principal types of distributed simulators were implemented on multi-tasking computer systems and multi-computer processor systems.

It is the purpose of this chapter to introduce multi-computer processing systems and to describe the advantages inherent in this type of computer system over a sequential processing system (such as a multi-tasking computer system).

6.2 Advances in computer hardware.

There has been, since the advent of the first electronic computers, a constant activity concerned with improving the performance and lowering the cost of computing machinery. Early attempts to lower the
The cost of computing installations consisted of attempts to keep them busy particularly by the use of batch processing \cite{BOWE80}. However, even while one job would be running the central processor would be idling during periods when external operations were required (for example: communication with slow peripherals or its human operators). This situation was remedied by the introduction of interrupt-driven hardware by which means the central processor was able to continue operations after requesting an external activity because it would be 'interrupted' when the task was completed \cite{HARL85}.

Operating systems were structured to take advantage of interrupt-driven hardware so that several user 'processes' would be executed during the same time period, interleaved in space and time in response to peripheral performance. 'Time-shared' operating systems developed from this for the purpose of managing a number of interactive terminal users for whom the illusion was created that the computer was dedicated to each one's need alone, while the central processor would be kept busy.

The decentralization of external activities so that the central processor could continue working on a different process while the external activity was taking place has led to the distribution of functions within a computer system. This distribution of processing power often leads to a lower system cost with enhanced performance.

The main influence on performance improvement since the original computer architecture concept was proposed by J. Von Neuman has been the advances in performances of semiconductor components. However, J. Von Neumann was critical of his own design of the sequential computer
architecture when he declared it to be obsolete, and that 'Large and efficient natural automata are likely to be highly parallel' /FORS85a/.

6.3 **Advantages of concurrent and distributed systems.**

There are many advantages to be expected from distributing processing power in a computer system. Firstly, an improvement in throughput, or speed of execution of a program related to the cost of the computer system. For particular applications higher throughput can be obtained by using a multiprocessor system instead of a uni-processor system of the same cost /BOWE80a/. Current technology is also rapidly approaching the absolute limit imposed by the physics of electronic devices on the speed at which instructions can be executed /SILC85/. Optical technology is, however, improving this speed although a multiprocessor computer system based on optical technology could still improve the throughput of a uniprocessor system but this will remain unanswered until devices are commercially available. Secondly, reliability can be improved because redundancy is easier to create in distributed systems. Graceful degradation can be achieved by enabling other processing units to take over the tasks of one that has failed.

Thirdly, improvement of the availability or response of the computer system. Tasks which require the greatest processing power can be executed on the most powerful processing units enabling others to be free to take on tasks suited to their level of performance. Fourth, the cost modularity or flexibility of a computer system can be an advantage where relatively low-cost processing units can be added to enhance the power of the computer system if greater speed or capacity
is required. Fifth, resources can be shared. Hardware resources (disk storage etc.) and programs, databases and computational power can be shared (rather than duplicated) in a distributed system, and lastly, ease of software design. Software at a high level can be easier to design, implement, debug and maintain if it is functionally partitioned and executed on dedicated processors.

The particular advantages of distributed computer systems applied to the simulation of complex highly automated manufacturing system are that, firstly, because software can be functionally partitioned, each module can incorporate a high level of detail which if incorporated into a single program, would cause the program to be long, convoluted and unwieldy. Secondly, an increase in throughput for a similar cost for the simulation system will enable a reduction in the delay during the design and analysis of a manufacturing system due to the time taken by an analyst using the simulation system to explore the effects of many different variables.

However, parallel computing machines have often only been successful when applied to particular problems and have often proved to be difficult to program, and to show diminishing returns in terms of efficiency. (i.e. if the time taken to execute a program on a uni-processor computer is 't', the time taken to execute the comparable program on a comparable multiprocessor computer with 'N' processors working at 100% efficiency would be 't/N'. This means with two processors the program would run in half the time, with three processors in a third of the time etc. However, in practice, the reduction in time is only 80% of that achieved at maximum efficiency for each additional processing unit, and hence the addition of
processors shows a diminishing return /FORS85/).

6.4 Concurrent language and systems software.

Several languages with mechanisms for parallel programming have been developed e.g. Concurrent Pascal, Modula, Chill and Ada /ARD084/. These languages, designed to express parallelism, were intended as originally designed to be used on ordinary computers with only one central processing unit. However, because of the small number of multiprocessor computers around, each computer tends to have a language adapted or developed particularly for it in order to handle the actual parallelism in the hardware.

Pattenden /PATT85/ describes extensions and developments of FORTRAN, PASCAL, BASIC and ALGOL to enable parallel processing and three properties which most parallel languages have to a greater or lesser extent. These are: the capability of defining arrays on which to act in parallel, the capability of selecting a subset of that array for partial processing, and the availability of parallel instructions with global control and with local control. However, it is noted that most of the languages surveyed did not support the simultaneous execution of several dissimilar tasks.

The system software organization used to manage parallelism depends on the reason for configuring the system and the interface that is required for the application user /NEWM82/.

6.5 Multiprocessor hardware

The basic model of a multiprocessor system is of a number of processor
units connected to memory and input-output devices, and the way in which this connection is achieved gives rise to the different organisations of multiprocessor systems /WOOD81b/. Three basic ways in which these connections are achieved are: time-shared or common bus; crossbar switch matrix; and multiport memories.

A time-shared or common bus organisation is where all the system components are connected by a common communication path (the bus). One unit wishing to communicate with another first ascertains that the bus is free and then places the address of the unit with the communication itself on the bus. All units which wish to receive communications monitor the bus to see if their address has been transmitted. Contention over the use of the bus is usually handled by a bus arbitration unit. A crossbar switch matrix organisation requires a different access path from each processor to each unit required to receive communication (memory, input/output modules, etc). The amount of circuitry required to cope with the potential contention at each interconnection in the crossbar is large but the main advantage of this type of system is that potentially, data transfer to and from each module can be made simultaneously.

A multiport memory organisation means that the memory becomes the interface by which communication is achieved. Each processor unit is able to access some common memory and may also have private memory, inaccessible to other processors. Pre-assigned priorities are usually given to the processors to reduce the contentions that may arise. The common memory may be physically located either centrally, on a separate memory card, or locally at any of the processor units. Private memory is usually physically located locally at the processor.
6.6 **Organisation of multiprocessor systems**

It is possible to classify parallel processing systems according to the means by which coordination of the parallel paths is achieved: those relying on hardware functions for coordination; and those relying on software functions for coordination.

A system which consists of several processing units which simultaneously execute the same instruction is classified as a **Single-Instruction-Multiple-data system (SIMD)**. These systems consist of many small arithmetic processors attached to a host computer. The processors are coordinated by being clocked by a common timing signal, so that while parallel operations are taking place, all the arithmetic processors are performing their steps at the same rate and the host processor is waiting at the hardware level for their operation to cease. Coordination at the software level is not required for these systems. Examples of these systems are image processors and array processors.

A system which consists of several separate processing units each capable of executing separate instructions independently is called a **Multiple-Instruction-Multiple-Data system (MIMD)**. Software coordination is required with MIMD systems because the 'unit of parallelism' is at a higher level than that which the hardware operates, i.e. the 'unit of parallelism' in an MIMD system can be complete sections of program (or many instructions). Some hardware primitives may be used but software coordination is required whereas with a SIMD system the hardware could take care of coordination since
each processor executed the same instruction, taking exactly the same time to execute, so that completion of each parallel path would happen at a known time. In an MIMD system the time taken to complete each parallel path may consist of a number of instructions, each path taking a variable amount of time to execute and hence software coordination must be built in to control the process. An example of software coordination using the function 'Lockset' is given in chapter 8.

6.7 Operating system organisation.

Three basic organisations used in the design of operating systems for microprocessors are: the master-slave organisation; an organisation which maintains a separate executive for each processor; and an organisation which treats all processors anonymously or symmetrically.

A master-slave organisation requires that the operating system routines are always executed in the same processor (the master). If a slave processor requires an operating system service, then a request must be made to the master processor and the slave must wait until the master has both become free and carried out the service.

An organisation in which a separate executive is maintained in each processor gives several potentially independant operating systems, however, cooperation and synchronisation must occur when shared resources are required. The supervisory code may be placed in shared memory (in which case only one copy need reside in memory) or it may be placed in the local memory of each system.
The third type of organisation, in which all the processors are treated anonymously or symmetrically, tasks are shared out asynchronously between the processors, and the status of master (the 'mastership') 'floats' between the processors. Each shared resource has only one master, but which processor will have the status of master is decided by the synchronisation which is required prior to the resource being accessed.

Parallelism within hardware normally would be concealed from a user of a parallel system by the system software where ease of implementation, overall system throughput, or overall reliability were criteria for performance since in these a user would not need to know or to be able to exploit the use of the individual processing elements which would be managed by the system software. This is typically the case in MIMD systems. SIMD systems however are generally designed for high execution speed on particular classes of problems (e.g. array or vector processing) The systems software in this case usually ignores the parallel architecture, passing it on to the application programmer to gain the maximum benefit from the parallel hardware with no system software overhead. However, the burden of programming is also passed on and this can lead to considerably longer development time for programs.

Programming a multiprocessor system explicitly is a great deal more difficult than programming a uniprocessor to carry out the same tasks since synchronisation between paths must be considered and even in some SIMD machines compilers may be provided which can translate vector or array operations, such as in APL, FORTRAN or PASCAL extensions, directly into the appropriate hardware operations.
It is, however, usually thought counterproductive to conceal the parallelism from the application programmer if the reason for configuring the system is to provide higher execution speeds within a single program because it may be difficult for the system software to use the available parallelism effectively for solving a particular class of problems (rather than the MIMD machine designed for processing many different types of application).

6.8 Conclusion of Chapter.

This chapter has described some of the organisation and advantages of multi-computer processing systems. In order to satisfy the criteria which have been developed for a simulator of highly automated batch manufacturing systems (in particular the criteria of increasing the speed of execution), a multi-computer processing system is required with a MIMD organisation.

The following chapters will describe the design of a distributed simulation system using such a computing vehicle with the objective of determining whether these criteria can actually be satisfied by a simulator of highly automated batch manufacturing systems.
CHAPTER 7

A DESIGN AID FOR HIGHLY AUTOMATED BATCH MANUFACTURING SYSTEMS.

7.1 Introduction

This chapter is intended to introduce the reader to the complete project of which the work reported in this thesis is a part. A major objective of the project is to produce a computer-aid to the design of highly automated batch manufacturing systems. The project was initially a collaborative research project involving Loughborough University of Technology, Baker-Perkins Ltd. of Peterborough, and Normalair-Garrett Ltd. of Yeovil/BELL83/, and was later expanded to include additional industrial collaborators.

7.2 The Design Aid.

The overall purpose of the design aid is to aid a designer in a number of interlinked tasks. The first task is to collect and collate the necessary data with which a proposed manufacturing system can be specified. The second task is to aid the designer in using an analytical modelling technique to examine whether the proposed system has the ability, on average, to produce the intended production rate. Thirdly, the designer is aided to specify the extra data regarding
spatial layout of the proposed system and more detailed data regarding part storage buffer capacities, speeds etc., required for detailed simulation which is then automatically configured for the simulation of the particular proposed manufacturing system.

Areas of work within the design aid include /BELL86/: 

(i) interactive manufacturing system data input and manipulation using a database management system; 
(ii) menu-driven software for the interactive systematic specification of manufacturing systems; 
(iii) automatic configuration of data for mathematical modelling and detailed simulation from (i) and (ii). 
(iv) analytical queueing modelling for the initial evaluation of manufacturing systems; 
(v) interactive graphical layout of a manufacturing system to determine the floor plan; 
(vi) software for the entry of layout data on transport routes and pallet movement. This data is necessary for the emulation phase and for providing animated graphical output; 
(vii) detailed simulation of a manufacturing system to provide detailed modelling and determination of dynamic values of system parameters; 
(viii) development of a novel approach to simulation by using multi-processing on a dedicated parallel processing micro-computer, to underpin the design system.

Areas (i) to (iv) comprise an initial evaluation stage, which by using
Manufacturing System Design

Figure 12 (After /ROBE85/)

- 90 -
an interactive 'softkey' approach as successfully used in MDI systems, collects and collates system data to specify a manufacturing system, and uses an analytical model to obtain approximate performance statistics.

Detailed modelling of manufacturing systems, which in some descriptions of this project is called emulation, embraces all areas except (iv). Detailed simulation requires a plant layout with detailed information on transport and workpiece handling. The simulation is automatically configured from the system specified in (ii), and executed using parallel processing methods to obtain dynamic system output.

These two phases of manufacturing system design are shown in figure 12. The first phase, called 'design evaluation' may require iterations until the manufacturing system design is shown to be capable of the average steady-state performance required. The second phase of detailed modelling, called 'emulation', results in an assessment of the expected performance of the manufacturing system design under dynamic conditions. Following an emulation of the manufacturing system a re-design may be required and subsequent re-evaluation, until the design is shown to be capable of the performance required both under average conditions and under short-term dynamic conditions.

The terms 'simulation' and 'emulation' tend to be used synonymously to described the modelling of systems in order to assess or predict the performance of original systems. The term emulation is used in this description of the research project to describe a model which imitates a system in regard to all the variables that it is possible to measure.
Figure 13
while using a particular computational vehicle. The distributed simulator described in this thesis is designed to approach this idealised performance characteristic of emulation by including a high level of detail in terms of the entities included within the model and in the specification of the activities. Some of the inputs and outputs of the emulation phase are shown in figure 13.

A flow chart which shows a route through the design aid is shown in figure 14 and figure 15. The initial task on this route is to set interactively the system specification by the input of data relating to parts, machines, load/unload buffers and material handling equipment. This data can then be used for approximate modelling evaluation or emulation. If modelling is selected, then the data is configured into a form suitable for use with an analytical model and results can be obtained. This work will be reported in more detail within a forthcoming complementary thesis /NEWM88/. If emulation is selected or the results of the analytical modelling are satisfactory, then the tasks shown in the part of the flow chart in figure 15 must be carried out. These tasks involve the configuration of emulation data and the addition of the necessary additional data (the static spatial layout, AGV route data, additional data for animation, a proximity data file). The emulation or detailed simulation is then executed and the results may be animated or analysed. The additional data which is required for this task is described more fully in chapter 10 and appendix I.
Figure 14 (After /NEWM87a/)
From Analytical Modelling

Configure Emulation Data

Create Scaled Layout of static FMS elements with DOGS CAD system

Create IGES file with DOGS system

IGES File Conversion Program to system graphics format

Generate AGV and Pallet route data

Add Rotary buffers, AGVs and Pallet data

Add segment numbers to layout file

Add buffer data information

Create proximity data file

Parallel Processing Emulation Run

Output Module

Results OK?

no

yes

Draw and Animate System

Results OK?

no

yes

Interactively re-build Initial System Specification

Stop

Flow Chart of Design Aid - 2

Figure 15 (After /NEWM87a/)
CHAPTER 8

A DISTRIBUTED SIMULATOR - 1 (OVERVIEW)

8.1 Introduction

This chapter is intended to describe the general principles of design of the distributed simulator which has been developed in order to investigate the proposition that a distributed simulation system for highly automated batch manufacturing systems can be advantageous for the purpose of manufacturing systems design.

A software and hardware overview of the distributed simulation system is given: in particular, the modular design is introduced and the entities of simulation which reside in the modules are described. The activities that these entities are engaged in, and the simulation executive are introduced. The hardware overview describes the computational vehicle on which the software is mounted.

The software itself is described in more detail in the next chapter (chapter 9).

8.2 Hardware Overview.

Some of the ways in which multi-computer processing systems are
organised have been described in chapter 6. Three basic methods of connection between processors were described, which were: time-shared or common bus; crossbar switch matrix; and multiport memories.

The two principal hardware organisations considered for use with the distributed simulator were the common bus organisation and the multiport memory organisation. An example of the common bus organisation is where a number of computers (micro, mini or mainframe) are connected together via a serial link such as a local area network (LAN). Data transfer in this case is slow compared with the time required to execute instructions by each processor.

A common bus organisation may also use the communication facility of the multiport memory-type organisation. An example of this organisation is to connect the processors together along a common bus through which communication between the multiport memories of the processors occurs in a parallel manner. The time taken for data transfer in this case is comparable with the instruction execution time of each processor.

An example of a computer system which is capable of being expanded to form a multi-computer processing system is the Intel System 310 computer. This is the computer on which the distributed simulation system described in this thesis was implemented.

8.2.1 The INTEL SYSTEM 310 Computer.

The Intel System 310 computer consists of a cardcage, power supply, disks, and disk controller, and a backplane which is Intel's
'Multibus'. 'Multibus' is an industry standard bus which supports the use of Intel's single board computers. Intel have a continued commitment to developing Multibus-compatible single board computers using newer, faster CPU's (for example, the 80186 and the 80286 CPU's). There is space in the SYS310 cardcage to allow additional Intel single board computers to be integrated with the basic SYS310 (figure 16).

A single board computer consists of a CPU, some form of input/output, and some local memory. On Intel's single board computers, this local memory can be configured as private memory or dual port memory (available to all computers connected to the same Multibus). Other dual-port memory (available to all single board computers) can exist on a separate memory card connected to the Multibus. The access time to local memory on-board is shorter than the time required to access off-board memory, so it is desirable that program code and data that is frequently used is located in memory which is quickest for access i.e. local on-board memory.

Data transfer between processors or single board computers in the Intel SYS310 is achieved by dual port RAM reads and writes, i.e. data transfer between multiport memories in a parallel manner along a common bus. (Dual port RAM is computer memory that can be directly accessed by more than one CPU although not simultaneously). There is no time overhead associated with this technique provided there is no conflict between separate computer CPU's attempting to access the same memory location. In a system where there is a limited amount of message passing compared with the number of instructions executed or time spent executing instructions by each processor the use of dual
<table>
<thead>
<tr>
<th>Hardware - Intel SYS310 Computer</th>
<th>Figure 16</th>
</tr>
</thead>
</table>

### Basic INTEL SYS310 Computer
- Single Board Computer
  - SBC 86/30

### INTEL MULTIBUS I
- Disk Controller
  - SBC 215G
- 512K RAM
  - SBC 012C

### Additional Single Board Computers
- Single Board Computer
  - SBC 86/12A

### Multiprocessing Slave Computers
port RAM can reduce the overhead inherent in the use of standard networks.

Some arbitration is required to resolve contention for access to memory locations between processors. This arbitration is provided on the Multibus in two ways: firstly, a priority resolution network is provided. This can be serial or parallel and in a SYS310 computer a parallel priority arbitration network is provided on the backplane of the Multibus. This is implemented in a way such that the position of a single board computer within the cardcage determines its priority within the system. The second way that arbitration is provided by the Multibus is by the use of control signals or lines which indicate that a particular processor has gained control of the bus and is using the bus for access to i/o devices or memory. Other processors or 'bus masters' (Multibus boards capable of gaining control of the bus) are not allowed to use the bus for the same purpose until the control of the bus is relinquished by the particular bus master using it. This contention by processors for the resources such as memory causes a time overhead and must be minimised for maximum efficiency in a multiprocessor system.

The resources of the computer system (such as disk storage, serial i/o) can be configured for multiprocessor access or for access by one processor only (this processor would be considered the master of the whole system). If the resources are configured for access by every processor, a copy of the operating system which contains the code to enable access to the resources must reside on each single board computer. If the resources can only be accessed by the master single board computer, then it is the only computer which requires a copy of
the operating system which includes code to use the resources, and
other single board computers (or processors) can be provided with
either a minimum operating system to carry out functions like task
suspension, etc., or be provided with no operating system at all and
to rely on the explicit use of hardware if required (i.e. code for
the use of hardware is provided by the programmer and not by the
operating system).

The distributed simulator uses only one copy of the operating system
which is used by the master processor. This is the only processor
which can use the resources such as disk storage. The other single
board computer in the experimental two-processor system is provided
with code which explicitly uses the resources available to it i.e.
the RS232 port situated on the computer board.

8.3 Software Overview.

8.3.1 Objects which are simulated in the modules.

The objects or permanent entities which are considered to be part of a
highly automated batch manufacturing system are:

- the machines (metal cutting and inspection machines);
- the part storage and part transfer buffers (i.e. pallet exchange buffers);
- the material handling systems i.e. automatic guided vehicles (AGVs);
- the tools used for cutting;
- the pallets which carry the parts; and
- the parts to be processed.

Each of these entities has appropriate characteristics which are parameters of the entity. Examples of these parameters are:

- for a machine: tool storage capacity, speed of tool exchange, and pallet load/unload time;

- for a pallet storage buffer: pallet capacity, pallet load/unload time, speed of pallet movement on the buffer, and direction of movement (for rotary buffer storage devices);

- for an AGV: there is a parameter of the speed at which it travels along a route;

- for the tools used for cutting: their maximum tool life and their tool life remaining;

- for the pallets which carry the parts: there are parameters of the number of different parts that they may carry and of the number and types of parts that are actually loaded onto the pallet (for a particular simulation run); and

- the parts to be processed have parameters of the total number of operations required (an operation being defined as one visit to one machine) and, for each
operation a number of tools are required: one for each 'sub-operation', each of which has a processing time. There is also a setting up time associated with each operation.

The buffers and machines in the simulated system are also considered to be 'stations'. This uniform type for both different types of objects is necessary in order that the AGV control system can refer to both types of objects in a uniform manner (a pallet can be delivered to a machine or a buffer - the operation that the AGV carries out is the same in each case).

8.3.2 Activities of the entities.

Each of the classes of permanent entities in the system (machines, material handling system components, pallet storage and transfer buffers) have activities which relate to their operation.

The activities of the machines are the following:

- loading;
- setting up;
- tool changing;
- cutting;
- completing the operation; and
- unloading.

The activities of 'tool changing' and 'cutting' may be repeated for as many different cutting sub-operations as are required a particular
operation. If more than one part type is present on a pallet then a 'changing operation' activity must take place.

The activities of pallet storage/loading and unloading buffers are:

- being requested to load a pallet;
- being requested to unload a pallet;
- being requested to load a machine;
- being requested to unload a machine;
- moving a specific pallet or space in a buffer to the load position (a load move);
- moving a specific pallet or space to the machine position (a machine move); and
- completing a move (transferring the pallet).

A material handling system consisting of automatic guided vehicles (AGVs) has the following activities:

- allocating AGVs to stations in the manufacturing system that request the service of an AGV;
- commencing AGV service;
- waiting for a related activity;
- travelling;
- waiting for a clear path; and
- completing the move.

The permanent entities in the system which store a variety of pallets i.e. pallet storage buffers and load/unload buffers also take part in a decision-making activity. The contents of each storage buffer are constantly examined and checked to see if any of the pallets on the
buffer are destined to be sent elsewhere within the system.

This decision-making activity consists of the following steps: calculation of the next pallet to be released from the buffer on the basis of some rule (for example, on the basis of priority or on shortest processing time etc.); checking to determine whether the pallet chosen is to be released to a machine which the buffer serves or is to be released into the manufacturing system via the material handling system; and updating the status of the pallets which have been loaded onto buffers ready for transfer elsewhere.

The preceding description of the objects or permanent entities which are considered to be part of a highly automated batch manufacturing system, and their activities, leads to the definition of the software modules described in the next section.

8.3.3 Modular design.

It has been described in chapter 5 that the software distribution in existing distributed simulation systems is achieved by sub-dividing the problem into modules. The modules have largely been distributed with regard to their simulation support function i.e. one module allocated to random number generation, another module allocated to the collection of statistics etc. The simulation system to be described here is designed to represent a certain class of models i.e. the simulation of highly automated batch manufacturing systems, and therefore the modules do not have to be generally useful modules but can be modules which are useful in order to simulate this class of problem.
Each of the different classes of permanent entities within a simulated highly automated batch manufacturing system is involved in a similar sequence of activities, although a specific activity may vary depending on the particular type of permanent entity within a class. For example, linear part storage buffers hold particular pallets in particular positions on the buffer: rotary buffers move a requested pallet to a load/unload position (the move may be in a clockwise or anticlockwise direction). Therefore rotary buffers must take part in the activity of 'moving the pallets on the buffer' or 'rotating the buffer', which a linear part storage buffer does not.

Each of these sequences of activities (or processes) together with a set of interface records (which are used to pass parameters between the modules) can be described as a module which can operate autonomously since none of the activities within the module are required by other entities within the system. This leads to the definition of five modules which can operate simultaneously. These are:

- a module representing machines;
- a module representing part storage buffers;
- a module representing the material handling system;
- a module calculating or identifying pallets to be released into the system; and
- a module generating output data.

Each module can also privately read and use the data which describes the entities which it represents (this data is not required by the other modules in the system) (see figure 17).

- 106 -
A sixth module is required: this is the control module which contains the simulation executive.

8.3.4 Simulation Executive

Each class of permanent entities in the system (for example, all automatic guided vehicles) can be represented to other parts of the system by a brief description or record in a computer data structure. This record, which can have a limited number of fields, is the interface between a module which controls the activity of the particular type of permanent entities and a control module, or executive. Examples of the type of fields present in the record are:

- a description;
- a number representing the state that the particular entity is in or the activity that the entity is currently involved in;
- a number representing the remaining time that the entity will be involved in the activity or will be in the particular state; and
- other fields representing useful control information such as which pallet is involved in the activity (for machines); which destination on automatic guided vehicle is travelling towards or currently stationed at (for agvs), etc.

The effect of changes in the states of the permanent entities within the system is achieved by the action of the control module or executive which changes the state of a related entity in its interface.
records.

An example of this action occurs when an automatic guided vehicle arrives at a particular destination. The record which represents the automatic guided vehicle will, as time passes, contain a sequence of numbers in the field describing the state of the automatic guided vehicle which show that the vehicle is travelling, docking at its destination, and then waiting.

The simulation executive is designed to detect this particular sequence of changes in state of the AGV and, providing that the entity at the destination at which the AGV has docked (e.g. a pallet storage buffer) is not involved in an activity as the AGV arrives, then the entity will have its state changed by the executive so that the correct sequence of actions takes place. If the receiving station or entity was a pallet load/unload buffer, for example, the state field of the buffer in its interface record would be changed by the executive so that in the following time the buffer would go through the necessary activity to unload a pallet onto the AGV, or conversely to move an empty position to the docking station and load a pallet onto the buffer, thereby releasing the AGV which would become available for other actions.

8.3.4.1 Simulation Cycle.

Each program module representing a particular class of entities cycles once, updating the state of each entity in that class due to the changes that have taken place since the last advance in time. The updates of the states of the entities appear in the interface record as described in the example above. The executive control program
module then cycles to check for a change in the state of any entity and hence subsequently changes the state of inter-related entity in the appropriate module. This control or executive cycle takes place after all the simulation modules have completed their independent cycles and before the next time advance. Hence the complete cycle of simulation is represented by one cycle of each module and one cycle of the executive. The simulation executive, which is similar to an unrepeated activity scan, searches a fixed number of the interface records for events due to happen (such as the AGV arriving at the buffer in the example above).

A unit time advance mechanism is used at present within this technique, so that after every simulation cycle the time is advanced by one unit. This mechanism has the disadvantage, when compared with a discrete time advance mechanism, that for an equivalent accuracy the number of time advancing steps is increased while questioning the possibility of every activity or task at each time advance /TOCH63/.

The interface records representing the permanent entities in the system are searched and tested in two phases. The first phase tests for the completion of events or activities which have previously been scheduled to start. The second phase tests if decisions have been made regarding the selection of pallets to leave the storage buffers so that additional new activities or events can be started. The series of tests in the first phase (for the completion of existing events or activities) are carried out in the following order:

- firstly, the machines in the system are checked to determine if any have completed the machining operation.
on a pallet;

- secondly, the records representing the buffers are searched for the following events:

  - pallets loaded onto a buffer;
  - pallets to be unloaded onto an agv;
  - pallets to be loaded onto a machine; and
  - pallets to be loaded from an agv.

- thirdly, the records which represent the automatic guided vehicles are searched to determine if any agvs have arrived at their destinations.

The reason for this particular order is that the actual completion of some activities and the starting of the next related activity is dependent on a particular state of an entity. For example, the unloading of a machine is dependent on the part storage buffer onto which the pallet is to be unloaded being free. This activity has the highest priority so that any free buffers at the start of the executive cycle are claimed for the activity 'unloading pallet from machine' first, before being claimed for any other activity. The checks on the buffer operations and agv operations are carried out next in this sequence: there is no particular reason for the checks on the buffer operations to be carried out before the check on agv operations.
The series of tests in the second phase (to determine if a decision has been made regarding the release of pallets from the part storage buffers) is carried out in the order of pallets to be sent to elsewhere in the system first, then pallets to be loaded onto a machine which the part storage buffer serves. A pallet can only be sent to elsewhere in the system if the associated entities are free i.e. there is room on the receiving part storage buffer and transport is available. If a pallet is available to be sent elsewhere in the system but the associated entities are not free, or there is no pallet to be sent to elsewhere, then the buffer may be involved in loading a pallet onto the machine which it serves.

This sequence means that a higher priority is attached to sending pallets to elsewhere in the system than the priority attached to loading pallets onto the machine which a buffer serves. This strategy aims at keeping the whole system utilised by pushing pallets through the system at the possible expense of the utilisation of individual machines. However, there is no reason why this priority could not be reversed in the simulation software.

8.3.5 Software language.

The language chosen for the development of the distributed simulation was PASCAL. The principle reasons for this choice were that:

(i) PASCAL has an inherent structure which, if used correctly, imposes upon the programmer a logical and economical method of building software; and

(ii) PASCAL has a strong typing of variables. This capacity
to strictly define the types and ranges of variables aids a high-level programmer because conflicts of types in parameter passing (such as in calls to subroutines or procedures) and variables exceeding a pre-set range are quickly and easily identified, leading to efficient debugging of software.

Pascal also has a self-documenting capability when appropriate variable names and procedures are chosen (the names are not usually limited to six characters as in other languages, and integers and real variables are not predefined by being limited to their names commencing with certain letters of the alphabet).

Intel's implementation of Pascal for its computer systems using the RMX86 operating system is called PASCAL86. A PASCAL86 program can be partitioned into smaller units, called modules, for separate development and compilation. There must be one main module and a number of non-main modules. Each module (either main or non-main) contains the following items (see figure 18):

- the module heading;
- the interface specification;
- the private heading;
- the private declaration/definition part; and
- a 'period' to end the module.

The objects of a module, which include labels, constants, types, variables, procedures and functions, may be either private or public. Private objects are those that only the particular program module may
<table>
<thead>
<tr>
<th>Module Heading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Specification</td>
</tr>
<tr>
<td>Private Heading</td>
</tr>
<tr>
<td>Declaration/Definition Part</td>
</tr>
<tr>
<td>Constant Definitions</td>
</tr>
<tr>
<td>Type Definitions</td>
</tr>
<tr>
<td>Variable Definitions</td>
</tr>
<tr>
<td>Procedure and Function Declarations</td>
</tr>
<tr>
<td>. (Period)</td>
</tr>
</tbody>
</table>

Structure of a PASCAL-86 Non-Main Module

Figure 18
reference while public objects are objects that other designated modules of the program may freely reference and are those declared in the interface specification of a module. The interface specification provides the only means for modules to communicate with each other.

This modular compilation facility together with the ability to declare public and private objects is exactly the structure required to build a modular program suitable for multiprocessing because the communication between the modules is limited to the variables declared in the interface specification. These variables translate into those which must be located in common memory i.e. memory accessible by all the processors in a multiprocessor system. Variables which are private to a module can be located in the local memory of a particular processor.

8.3.6 Operating System.

The operating system chosen for the development of the distributed simulation system was INTEL RMX86, which is a real-time multi-tasking executive. It provides constructs for multi-tasking (for example, task creation; task suspension; task deletion; semaphores) and is configurable to whatever size and complexity is required for a particular application. A minimum configuration of RMX86 may be required to run slave tasks in a multiprocessor environment (where each processor has its own copy of the operating system) while the master processor in a multiprocessor environment may require a complete version of RMX86 to enable it to carry out tasks that other processors do not, such as disk i/o etc.
Tasks are the basic elements of applications built on the RMX86 operating system. Each task is an entity capable of executing CPU instructions and issuing system calls in order to perform a function. The characteristics of a task are the register values, a priority between 0 and 255 and the resources associated with a task. Each task in an rmx86 system is scheduled for operation by the 'Nucleus' of rmx86. The five states in which a task may be placed are: 'ready to run' ; 'running' ; 'suspended' ; 'asleep' ; or 'asleep and suspended'. A task may, once it is running, suspend itself /RMX86/.

Each software module required to run independently can be created as a task and then control of that task is implemented with the controls described above (sleeping, suspension). Other controls are provided by rmx86 for real-time event synchronization in multi-tasking applications. The controls are regions and semaphores, for example.

Regions can be used to restrict access to critical sections of code and data, and are typically used to protect data structures from being simultaneously updated by multiple tasks. Semaphores are used to provide mutual exclusion between tasks. They contain abstract 'units' that are sent between tasks, and can be used to implement the co-operative sharing of resources.

Intel also provide support for multiprocessing with standard interfaces provided by the MMX800 Multibus Message exchange package, which enables the rmx86 operating system to support a loosely coupled multi-processing environment. Tasks on one processor may communicate with tasks running on other processors even if they operate under different operating systems.
8.3.7 Other facilities for control.

Intel's language PLM/86 also provides functions which can be usefully used by PASCAL86 programs for the control of shared data areas and synchronization. One such facility is the built-in PLM function 'Lockset'.

Lockset has two parameters: a pointer to a variable, and a value, which is the value to which the variable should be set by the function Lockset. The number returned by the function is the original value of the variable.

A convention is established that no processor may access shared memory locations unless lockset returns the value of zero (0) for the original value of the variable, and that lockset always changes the value of the variable to one (1). A programming construct is used so that a program will not continue beyond a certain point unless 'Lockset' obtains a value of zero for the returned value of the variable. When a program which has passed this point has completed accessing any shared data area it sets the value of the variable back to zero.

A program which needs to access a shared data area uses the function 'Lockset'. If the value returned by Lockset is 'one', then the variable has previously been set to 'one' by another program. The program requiring access to the shared data is made to wait in a loop, testing Lockset. Meanwhile, the other program should complete accessing the shared data area and will then set the variable to zero. Lockset will then return a value of 'zero' and the program requiring access to the shared data area will continue.
CHAPTER 9

A DISTRIBUTED SIMULATOR - 2 (OPERATION)

9.1 Introduction

The purpose of this chapter is to describe the operation of the software and the hardware of the distributed simulator in more detail than the overview in the previous chapter.

9.2 Operation of the modules.

The activities and the operations, which are carried out within the modules which simulate each particular class of entities, will be described in the following sections.

9.2.1 Simulation of Buffer Storage Operation.

The activities of pallet storage/loading and unloading buffers are:

- being requested to load a pallet;
- being requested to unload a pallet;
- being requested to load a machine;
- being requested to unload a machine;
- moving a specific pallet or space in a buffer to the load position (a load move);
- moving a specific pallet or space to the machine position (a machine move); and
- completing a move (transferring the pallet).

The sequence in which buffers are involved in these activities is shown in figure 19. The structure of the major individual activities (which is described below) is shown in a diagrammatic form in figure 20.

9.2.1.1 Loading pallet Activity.

The activity of being requested to load a pallet from an AGV consists of finding a free space on the buffer; moving that free space to the load/unload position; and transferring the pallet. Initially, the state of the buffer is changed from the state of '2' (a request to load a pallet has been received) to a state of '7' (moving free space to load position). A procedure called 'stationrequest' is called, which finds the position of the nearest free space, (in this example) then calculates the time it will take the buffer to move to this free space based on the speed of the buffer as described in the data file 'bf'. The strategy used to find the nearest free space depends on the type of pallet storage buffer: for example, if the pallet buffer is rotary and can only move clockwise, then the nearest free space on the buffer is found starting in an anti-clockwise direction (see figure 21). If the pallet buffer is rotary and can move clockwise or anticlockwise then the nearest free space is found starting in either direction.
Buffer Activities - 1

- load pallet from AGV
- move pallet to load/unload position
- load pallet to machine
- unload pallet from machine
- moving buffer
- moving space to load/unload position
- moving pallet to load/unload position
- moving pallet to machine position
- moving space to machine position

complete move (end buffer activity)

Figure 19
## Process the activities of each buffer

### Major activity types (states of the buffer)

<table>
<thead>
<tr>
<th>'Load pallet from AGV'</th>
<th>'Move pallet to load/unload position'</th>
<th>'Load pallet to machine'</th>
<th>'Unload pallet from machine'</th>
<th>'Moving buffer'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find a free space on the buffer</td>
<td>Find the particular pallet on the buffer</td>
<td>Find the particular pallet on the buffer</td>
<td>Find a free space on the buffer</td>
<td>Subtract time increment from duration of the move</td>
</tr>
<tr>
<td>Calculate the time duration to move the space to the load/unload position</td>
<td>Calculate the time duration to move the pallet to the load/unload position</td>
<td>Calculate the time duration to move the pallet to the machine position</td>
<td>Calculate the time duration to move the space to the machine position</td>
<td>remaining time = 0 ?</td>
</tr>
<tr>
<td>Put buffer into state 'moving space to load/unload position' (moving buffer)</td>
<td>Put buffer into state 'moving pallet to load/unload position' (moving buffer)</td>
<td>Put buffer into state 'moving pallet to machine position' (moving buffer)</td>
<td>Put buffer into state 'moving space to machine position' (moving buffer)</td>
<td>no</td>
</tr>
<tr>
<td>continue</td>
<td>complete move depending on type of move (moving pallet or space, to load/unload position or to machine position)</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>final buffer state = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Repeat until all buffers involved in activities have been processed

Buffer Activities - 2
Pallet Storage Buffer Load / Unload Strategy

Figure 21
The time that it takes the buffer to move a free space to the load position is not only dependent on the speed of movement of the buffer but also on the type of buffer (again, whether it can move clockwise, anticlockwise etc.). Following the calculation of this time duration the pallets on the buffer are moved (i.e. the buffer moves) relative to the fixed 'ground' starting position of the buffer if an integer incremental amount of movement has been reached in the elapsed time. If the buffer has also completed its move at this stage, i.e. the remaining time is 0, then an operation completion stage is executed: the pallet or free space is indicated to have arrived at the load position and the buffer is put into an intermediate state before becoming free in order to indicate to the control software or module that it has completed a particular operation.

9.2.1.2 Other buffer Activities.

The other buffer activities of moving a pallet or free space to the machine position ('load machine' and 'unload machine') and of moving a pallet to the load/unload position ('unload request') follow a similar pattern: i.e. finding the position of the nearest free space or the particular pallet requested (in the cases of 'unload request' and 'load machine'); calculating the time taken to move the pallet or free space to the machine or load/unload position; moving the pallets around the buffer if an integer incremental move has been achieved in the elapsed time; and if possible, concluding a move by indicating that the pallet or free space has arrived at the load/unload or machine position.
9.2.2 Simulation of Machining Activity.

The activities of the machines in the system are:

- loading;
- setting up;
- tool changing;
- cutting;
- completing the operation; and
- unloading.

The sequence in which machines are involved in these activities is shown in figure 22. The structure of the major individual activities (which is described below) is shown in a diagrammatic form in figure 23.

The machine is requested to load a part from a buffer by putting it into the state '4'. This finds the part number (the first part on the pallet), and the quantity of that part type; and the number of the operation required on the part. If the part number and operation is the same as that for which the machine is already set up then the operation is begun; if the part number and operation is different from the already set-up part then the machine is put into the state of setting up for a duration as specified in the operation file for the operation of that part. Following the set up the number of the part and operation for which the machine is set-up is stored and the operation is begun.
Machining Activities - 1

Figure 22

- start machining activity -
  - 'loading' -
    - 'setting up' -
      - 'tool changing' -
        - 'cutting' -
          - 'completing operation' -
            - 'unloading' -
              - end machining activity -
## Process the activities of each machine

<table>
<thead>
<tr>
<th>Major activity types (states of the machine)</th>
</tr>
</thead>
<tbody>
<tr>
<td>'loading'</td>
</tr>
<tr>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>load pallet from buffer</td>
</tr>
<tr>
<td>got first part number and quantity</td>
</tr>
<tr>
<td>is the part the same as the last part machined?</td>
</tr>
<tr>
<td>no</td>
</tr>
<tr>
<td>yes</td>
</tr>
<tr>
<td>no</td>
</tr>
<tr>
<td>yes</td>
</tr>
</tbody>
</table>

Repeat until all machines involved in activities have been processed.
This occurs by calling the procedure 'opstart', which finds the total number of operations to be carried out on the part and the number of sub-ops (or distinct 'tool' operations) to be carried out. The first activity is then to change the tool (it is assumed that the machine commences the cycle with no tool in the spindle). This activity initially stores the current tool number in the pocket of the tool magazine which is in the tool exchange position (if there is a tool to be stored) together with recording the tool life left for that tool. Then the tool magazine is searched for the tool that is requested for the operation and the tool with the highest remaining tool life is selected (if any of the tool type are present). If a tool is found then the machine is put into the state '6' (toolchanging) for a duration dependent on the distance of the tool from the changing position to the pocket where the tool is located.

Following the tool change, the machine will go into the activity of cutting (state '7'). This simply puts the machine into the state '7' for the duration of the cutting operation. If, during the cutting activity, it is found that the tool has used all its useful cutting life, then if the cutting operation is completed the tool is changed for the next tool required, otherwise a duplicate tool is searched for in the tool magazine. When a cutting operation is completed a change of operation takes place. This change of operation determines whether all the cutting operations have been completed on the part. If they haven't, then the next tool is found and the machine is put into the activity of cutting again. If all of the cutting operations have been completed on the part, then a check is made to determine whether all of the parts on the pallet have been completed. If there are more parts on the pallet then the machine may have to be reset for a
different part type, so the complete cycle of the machining activity is commenced: otherwise the machine is put into the state of 'ending operation', from which is will go into the state of 'idle' or 'free'.

9.2.3 Simulation of Automatic Guided Vehicle Activity.

The activities of the material handling system consisting of automatic guided vehicles are:

- commencing agv service;
- waiting for a related activity;
- travelling;
- waiting for a clear path; and
- completing the move.

The module which simulates the AGVs also contains an activity or procedure to allocate AGV's to stations that request them; and to find possible routes for the AGV's.

The sequence in which AGVs are involved in these activities is shown in figure 24. The structure of the major individual activities (which is described below) is shown in a diagrammatic form in figure 25.

The first function of the module is to find a station which is requesting service by an AGV. Each station has a priority number in a data file 'rq' and this function of the module finds the station with the highest priority at the time which requires service.
AGV activity

start AGV activity

commence AGV service

waiting for related activity

travelling

change route

complete move

waiting for clear route

end AGV activity

AGV Activities - 1

Figure 24
**Process the requests of stations for service**

Find a station that is requesting service

Allocate AGV to the station
(put AGV in state 'commencing travel' (state = 4))

repeat until there are no more stations that require service or there are no more free AGVs

---

**Process the activities of each AGV**

<table>
<thead>
<tr>
<th>Major activity types (states of the AGV):</th>
<th>'request for agv service' state = 4</th>
<th>'travelling' state = 2</th>
<th>'completing move' state = 5</th>
<th>'waiting for related activity' state = 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is AGV at final destination?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no</td>
<td></td>
<td>Subtract time increment from duration of activity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>find next route</td>
<td></td>
<td>time remaining = 0 ?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is a route free?</td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGV state = 'waiting for clear path'</td>
<td></td>
<td>complete move state = 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state = 3</td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGV state = 'travelling' state = 2</td>
<td></td>
<td>continue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGV state = 'completing move' state = 5</td>
<td></td>
<td>another route?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGV state = 'waiting for related activity' state = 6</td>
<td>complete move state = 5</td>
<td>calculate new time duration</td>
<td>do nothing</td>
<td></td>
</tr>
</tbody>
</table>

repeat until all AGVs involved in activities have been processed

---

**AGV activities - 2**

Figure 25
This facility can be used to express a preference for service when more stations are requesting service than there are AGV's. If there is no preference on the basis of priority then the stations are chosen by the rule first come-first served (FCFS).

If a station is found that requires service, then an AGV is allocated to the station (if there are any AGV's available). In addition to the priority indicated in the data file 'rq', there is also an indication for each station of the basis on which the AGV should be allocated to it (this is an indication of the type of service that the station requires). This can be the nearest free AGV; redirecting a particular AGV to the station; or redirecting a particular AGV with a time delay (used for the occurrence of 'deadlock' - see section 9.4). In the case of a station being found that requires service and requiring an AGV to be allocated to it using the rule of 'nearest free AGV' then the nearest AGV which is free is found using the proximity of the stations indicated in a data file 'pf'. In the cases of the AGV being redirected, the number of a particular AGV is known from the request data file 'rq'. (For a description of the data files, see appendix I). In all these cases once an AGV has been found then the AGV is put into the state of 'commencing travel' and the destination of the AGV is initialised with the number of the requesting station together with other related initialisations.

This cycle of finding a station which requires servicing and then allocating an AGV to it is repeated until either there are no more stations that require servicing or there are no free AGV's.
The second function of this simulation module is to find an AGV which is involved in an activity. Each AGV, when sent to service a station, takes the priority of that station as its own priority. The AGV's which are involved in activities are processed on the basis of priority (if no difference in priority then on the basis of FCFS). Each AGV may be involved in the activities of travelling; free; waiting for a clear path; commencing travel; or waiting for a related activity to take place.

If an AGV has been allocated to a station which was requesting service during the first function of this simulation module, the AGV is in the state of 'commencing travel' (state = 4). The activity related to this state firstly checks whether the AGV is already at its final destination, and if so the AGV move is completed (no move has actually taken place) but the AGV is put into the state that indicates that it has arrived at its destination. If the AGV is not at its final destination then the next route is found. The procedure to find the next route initially determines all the routes that are possible from the station at which the AGV is waiting. Then, for all the possible routes, each route is checked to see if it is blocked by an AGV already on the route or will be blocked by another AGV on its way to that route. If an AGV is blocking a route and the AGV is free then the AGV which is blocking the route is sent to its 'home station'.

If the route is free or a number of routes are free, the route that is closest to the ultimate destination of the AGV is chosen based on the proximity data in the file 'pf' (see appendix I). The time that the AGV will spend on the chosen route (dependent on the speed of the
AGV is calculated and the state of the AGV is set to 'travelling' (state = 2). If no route was possible, the AGV is set to the state of 3 which is 'waiting for a clear path'. (This state or activity checks, at all subsequent time steps, the possibility of a clear route until a route is found).

The activity of 'travelling' (state = 2) subtracts the time elapsed from the duration that the AGV should be on the route, and when the remaining time is zero a check is made to see if the AGV has arrived at its ultimate destination. If not, the next route is found and 'travelling' is continued on the new route with a duration related to the speed of the AGV and the length of the new route.

The 'activity' of 'waiting for a related activity to take place' is a dummy state. The state of the AGV is set to '6' and in this state the AGV is not free, so it cannot be used for any other purpose (or sent home, for example), and the AGV remains in this state until its state is changed by the controlling executive. This reserves AGV's for particular purposes used when breaking deadlocks (see section 9.4).

This second function of the AGV simulation module is repeated (the cycle of finding the next AGV involved on an activity and carrying out the various procedures depending on the state of the AGV) until all the AGV's have been checked. This completes the simulation of AGV activity.

9.2.4 Simulation of Decision-making Activity.

Each buffer or load/unload device which stores a pallet can be involved in a decision making activity. If the state value in a
record (which conceptually represents the decision-making state of each buffer) is set to '1' then a decision is required from the buffer. If a decision is made by the buffer the state value in this record is set to '2' indicating to the control strategy that a decision has been made. If the state value is '2' then no action or calculation is carried out by the decision module. The following strategy is carried out by the decision module if the state value (for any buffer) in this decision record is set to '1' (decision required).

9.2.5 Loading/Unloading Strategy.

Each pallet storage buffer in the simulated system can be assigned a particular load/unload strategy to decide between candidate parts or pallets for loading to a machine or unloading from the buffer. In the case study described in the following chapter, each pallet storage buffer was assigned the same rule, based on priority. (In the case of equal priority, then the rule was FCFS ('First come - first served')). This was implemented in the following manner: for each buffer on which a pallet or pallets could be stored, a list of pallets held by the buffer was examined (this list is actually the data file 'bdf' - see Appendix I). This list is examined once for each possible destination that a pallet can go to in the system. During each of these examinations, the number of the pallet with the highest priority (or if equal priority the pallet chosen first) for the destination of interest during the particular examination is placed in an array of destination numbers coupled with pallet numbers. A different rule could also be used, such as shortest (least) processing time (SPT) or any heuristic. The purpose of this first
selection process is to provide the numbers of the pallets chosen with respect to the particular rule in use which are waiting on the buffer ready to be released for each possible destination.

This array of pallet numbers (one pallet number for each possible destination) is then examined to choose from the candidate pallets one which is destined for a machine that the buffer might serve and one which is destined to be sent elsewhere in the system.

The purpose of providing a second stage in the selection process is to provide an opportunity for a rule to be used based on the dynamic state of the system. The relative importance of each destination normally changes dynamically and hence at this stage it is possible to provide the pallet number that should be released from the buffer for the most important destination at the time that the decision is made.

In this case the rule of FCFS (first come - first served) was again used and the pallet number destined for the first machine (or if no pallet, subsequent machines in ascending order) would be chosen. Two pallet numbers would be provided for each buffer on which pallets could be stored in this way: the first pallet number would be the pallet intended for a machine that the buffer serves, and the second pallet number would be the pallet chosen to be sent elsewhere in the system.

9.3 System Control Rules/Technique.

A brief description of the simulation executive which provides the operating rules for the simulated system was given in the preceding chapter. However, a more specific description will be given here.
The first activity of the control strategy is to check whether any of
the machines have completed their operations on a pallet by comparing
the last state of the machine with its current state. Each machine is
examined in turn and if its last state was '5' or equivalent to
completing the operation and its current state is '3' or equivalent to
idle then a search is carried out to find out which pallet storage
buffer or load/unload buffer serves the machine. If the storage
buffer is free and waiting to carry out an operation then it is put
into the state (or signalled) to begin the activity or operation of
unloading a pallet from the machine. If the pallet storage buffer or
load/unload device is not free then the machine is maintained in the
state of 'ready to unload a completed pallet' or state '5'.

The second activity of the control strategy is to check the activities
of all the pallet storage buffers and load/unload buffers to determine
whether any of them have completed their activities or operations.

The first part of this second activity of the control strategy is to
check whether any pallets have been unloaded from the machines. If a
pallet has been unloaded from a machine then the number of the pallet
is put into a list of pallets held on the buffer. This check takes
place for all the buffers.

The second part of this second activity of the control strategy is to
check a series of possible operations that may have been completed by
the buffers for each buffer in turn.

The first of this series is to check whether the buffer is ready to
load a pallet to an AGV (which will have been previously requested to
travel to the buffer). If the AGV is ready and waiting at the
load/unload position of the buffer then its state will be '0', which is checked.

If the AGV is there and the AGV does not have to carry out a further move to arrive at the pallet load/unload position (as it would be in the case of pallet storage buffers with a number of load/unload positions (as shown in figure 26), in which case the additional move is carried out first), then the following actions are carried out:

firstly, if the pallet is to travel to a machine the number of the buffer which serves the machine is found; then if the buffer is free (in a state of '1') the pallet is deleted from the list of pallets on the unloading buffer; the pallet number is linked to the AGV data list; and the receiving buffer requests the particular AGV carrying the pallet to travel to it. If any of these actions have not been possible then the buffer unloading the pallet remains in a state 'ready to unload the pallet' in order to try next time (the next time that the simulation executive or control program is executed).

The second of these series of checks on operations is to check whether a buffer is ready to load a pallet to a machine, i.e. that a pallet has been moved into position to be loaded on to the machine. If it has, then the pallet is deleted from the list of pallets on the buffer; the pallet number is transferred to the machine data; and the machine is put into the state of 'loading pallet' (if the machine was free).

The third in this series of checks on operations is to check whether a pallet storage buffer is ready to collect a pallet from an AGV which is being sent to it (or is travelling to it). Firstly, the AGV which has completed its journey and is waiting at the buffer is
Additional move required to pick up pallet at position 2

AGV Track

Logical Connection of AGV Track to Pallet Storage Buffer at position 4

AGV Movements at Linear Buffer

Figure 26
found; if the AGV is at the actual load position of the buffer (the exception is a long linear buffer) then the pallet is transferred, i.e. the number of the pallet is added to the list of pallets stored on the buffer and the AGV is made to be free. If the AGV is not at the actual load position of the buffer then its position is adjusted until it is. This completes the series of checks in the second part of the second activity of the control strategy.

The third activity of the control strategy is concerned with the actions (or activities) of the AGV's in the system. For each AGV in the system, its state is checked to find out if the 'state = 0' which would mean that the AGV has arrived at its destination. If the station at which the AGV has arrived is free (state = 0) or (state = 1) then the station (which would be a pallet storage or pallet load/unload buffer) is put into the state where it will carry out the activity of 'loading pallet from AGV' during subsequent time cycles.

The fourth activity of the control strategy is concerned with pallets which are to be unloaded from the pallet storage buffers or load/unload buffers in the system. As described in section 8.2.5, the loading/unloading decision strategy provides, for each buffer, the numbers of pallets to be unloaded and the numbers of pallets to be loaded onto the machines. During this fourth activity of the control strategy the buffers are checked to determine if they have a pallet to be unloaded and if they are available or free in order that the pallet can be unloaded. If this is so, then the process of sending a pallet through the system is commenced (to be discussed in more detail later).
Every buffer is checked for unloading, and then the fifth activity of the control strategy takes place. This is similar to the fourth activity except that in this case all the buffers are checked to see if they have a pallet to be loaded onto a machine which the buffer serves. If the buffer has such a pallet and is free then the process of loading the pallet onto the machine is commenced. This process is, in fact, the same process as that of sending the pallet through the system which will now be described.

9.3.1 Pallet Storage Buffer Unload Strategy.

This strategy or algorithm attempts to set the events in motion which will send a pallet from one storage buffer or load/unload buffer to elsewhere in the system. The algorithm is used in response to the fourth or fifth activity of the control strategy detecting that a pallet can be unloaded from a load/unload or storage buffer (i.e. that a pallet is available and the buffer is free). The first function of the algorithm is to determine whether the intended route of the pallet is to a machine or a storage buffer: if it is to a machine then the number of the load/unload buffer that serves the machine is found, and if that number is the same as the number of the buffer for which the algorithm is being invoked then, following a check to determine if the machine is free, the buffer is put into the state where during the subsequent time advancement the pallet will be moved to the load/unload position and when it arrives, the second part of the second activity of the control strategy will take care of the event.
If the intended route of the pallet was to a storage buffer or a buffer which serves a machine that is not the same one as the one that the algorithm was invoked for, then a series of checks are carried out. Firstly, the number of spare or empty positions on the intended buffer are checked to make sure that there is room for the pallet to occupy; the number of free AGV's is counted; thirdly, if the buffer which is intended to receive the pallet is free (state = 1) and an AGV is free and the buffer is not involved indirectly in another activity then a flag is set which indicates that it is possible to unload the pallet.

The next check determines whether any other buffer is currently involved in an activity which will subsequently require the services of the buffer for which the checks are being made. If this is true, then the flag is set back to 'false'.

If it is found impossible to unload the pallet at this point, i.e. the flag is found to be set to 'false', then a check is made which tests for 'deadlock' (see section 9.4). Following this series of checks, the second function of the algorithm is to set the unloading of the pallet in motion, subject to the status of the flag (called 'unloadpallet'). These actions consist of putting the buffer into the state where subsequently it will move the pallet to the load/unload position; clearing the number of the pallet from the decision data (indicating that the pallet is being unloaded) and requesting an AGV to be sent to the buffer. Subsequent events such as the arrival of the AGV at the buffer and the arrival of the pallet at the load/unload position are taken care of by the second part of the control strategy (as previously described).
9.4 Deadlock.

As stated by Carrie /CARR85a/ 'Deadlock' or 'blockage' can be an unforeseen problem in automatically controlled FMS. 'Deadlock' is the condition where a pallet has been completed on one machine 'x' and is due to go to machine 'y'. During this time a pallet is undergoing an operation on machine 'y' and, when the operation is completed, is intended to go to machine 'x'. Machine 'x' and machine 'y' have no spare or remaining buffer capacity, so that it is impossible to store another pallet either at machine 'x' or machine 'y'. When the pallet undergoing the operation on machine 'y' is completed, it cannot leave the machine or load/unload buffer because there is no room or space for it to go to because the pallet on machine 'x' is occupying the space. The same statement applies to machine 'x' in reverse (figure 27).

When 'deadlock' occurs, all pallets which must pass through machines 'x' and 'y' (in this example) complete their operations before they require these machines and then remain waiting in the system. The whole system gradually grinds to a halt due to the initial occurrence of deadlock. The principal reasons for deadlock are: firstly, 'difficult' routing, where the pallets flow around the system in opposite directions; secondly, inadequate buffer storage capacity at the machines; thirdly, unsophisticated control rules; fourthly, no alternative routes available. It is probably impossible to completely avoid deadlock in all circumstances, but if it occurs explicit action can be taken to remedy the problem (when it is identified). The first reason, 'difficult' routing, is a problem if the flexibility of the system is to be maintained because if only certain routes were to
Machine 'X' is to be transferred to machine 'Y'.

Pallet on machine 'Y' is to be transferred to machine 'X'.

Deadlock

Figure 27
be offered within the system this would obviously limit the flexibility of the system.

The second of these principal reasons, inadequate buffer capacity, can be solved easily but at a high cost of work-in-progress, machinery and more complex control rules. The third of these reasons, that of unsophisticated control rules, can be solved with techniques such as timing-out the waiting time of the pallets or explicitly deadlock identification and solution. Timing-out the waiting time of the pallets would return pallets to a central store or interim store if they had waited at a machine for a time period over a pre-defined limit. Explicit deadlock identification and solution works only where there is more than one AGV or transporter available. This is the solution incorporated in the control rules of the simulation system described in this thesis, and will now be described in more detail.

During the execution of the pallet storage buffer unload strategy (described in the previous section) a series of checks are made to determine whether there is room for the pallet to occupy at its intended destination; whether there are any AGVS free; whether the buffer at the intended destination is free; and whether the buffer is not already requesting AGV service due to prior involvement in another activity. If the result of these checks is positive, a flag (called unloadpallet) is set to true. This flag can then be set false by the next check which determines whether any other buffer is currently involved in an activity which will subsequently require the services of the buffer for which the checks are being made (or carried out).
Following these two sets of checks or tests, if it is found impossible to unload the pallet (unloadpallet = false) then a procedure is carried out which is called 'testdeadlock' which tests explicitly to find out whether the station or buffer to which the pallet is intended to go has no spare space available, and whether it is trying to unload a pallet intended for the buffer for which the checks are being made, and that the buffer for which the checks are being made has itself no spare space available. If these conditions are all true then deadlock has occurred and a flag (called 'testflag') is set to true. This flag forces the subsequent actions of the procedure to be carried out providing there is more than one AGV available (since two AGV's will subsequently be required); the subsequent actions are those of the second function of the algorithm which set the unloading of the pallet in motion. However this time, a flag called 'changedeadlock' is set to true, and the number of the station or buffer that the pallet is intended for is stored in a variable called 'deadstn'. The control strategy is allowed to continue as normal, but before the control sequence is finished, the flag called 'changedeadlock' causes another flag called 'deadlocked' to be set true. This causes (in subsequent cycles of the control strategy) only the buffer station which is involved in the deadlock to be allowed to unload a pallet, and hence the deadlock is solved explicitly. The flag 'deadlocked' is set back to false when this other buffer commences unloading the pallet. This strategy for solving deadlocks also requires that the AGV which is sent to pick up the pallet from the first buffer involved in the deadlock waits a suitable length of time after loading the pallet and before commencing travel to the destination in order that another (or the other) AGV has sufficient time to travel to the deadlocked
station (the first AGV's destination). This wait time depends on the distance of the second AGV from the deadlocked station but in this case study discussed here was chosen to be a constant time. The wait time is inserted by the AGV module as a consequence of the particular strategy or decision requested by the control module (because it has detected the deadlock).

This strategy has proved to be effective in the particular case study described in the following chapter because no other actions are allowed to take place once deadlock has been detected and the deadlock is only attempted to be solved provided that there are resources available (i.e. at least two AGV's). This strategy also requires that the AGV's can travel round loops i.e. that both AGV's will be able to independently reach their intended buffers without being blocked by each other.

9.5 Results and simulation output.

The results of the simulation are expressed in terms of a table of events which is a list of events happening versus time. For example, if an automatic guided vehicle is moving on a track, then the table of events produced for that simulation cycle will show a number in the column representing the actions of that automatic guided vehicle (AGV). The number represents the track or route on which the AGV is travelling. A line of information containing these numbers (0 if nothing is happening to an entity) is produced every time events occur in the system. No lines are produced if no events have occurred in the simulated system.
Each line of information contains a number representing the number of graphical increments to be processed (to be explained shortly); a number for each entity which carries out actions or moves within the simulated system (pallets, AGV's, moving buffers (e.g. rotary buffers)); and a number which is the time in the simulation when the event occurred. For example, the machining of pallets is shown in the table by a column, which represents a pallet containing a number which is the number of the machine on which it is being processed added to an offset (to make the number so produced sufficiently higher than any other number in the table viz. AGV routes so as not to cause any confusion).

This table of information can either be processed while the simulation is executing (which would slow the simulation down somewhat) or produced as a file of information which can then be processed and re-processed many times without re-running the simulation. Following the output of interim results (explained in the previous section), a question is asked which is 'Do you wish to continue....?'. If the question is answered 'yes' then the number of simulation cycles required must be input; If the question is answered 'no', then one further simulation cycle must take place and then output can be requested in response to a prompt. This latter action will cause the table of events to be written to a disk file and then control of the computer to be passed back to the operating system. The file can then be used by other software programs.

9.5.1 Derivation of Utilisations.

The utilisation of the entities represented in the table is derived by
a separate program. The cumulative time passed is indicated in the
file by the last number in the line, and for each entity represented
in the table a utilisation based on the equation 'time in use / time
passed' is calculated and placed in a temporary array for storage.
This calculation takes place for each line of information in the file
and can be displayed at interim periods (for example, after the
smallest time increment represented by one line, or after, say, 10
time increments which could be represented by a number of lines).
The display is obtained using a SIGMA raster-scan colour terminal
S5660, and is in the form of a bar chart (one bar per entity
represented in the table). The simulation output file could be
analysed for other statistics such as waiting time and queue lengths
etc.

9.5.2 Animated Output.

Animated output can also be obtained from the simulation output file.
Part of the input to the simulation system is in the form of data
describing the spatial relationship between entities in the system,
such as between part storage buffers, machines and automatic guided
vehicles (AGV) docking stations. This data is initially derived
from a layout of the manufacturing system to be simulated. Input is
required in the form of interactive questions and answers to describe
the spatial relationships, length and direction of routes from part
storage buffers to machines, etc.

This 'route data' which describes routes in the simulated system from
one entity to another is also used by a separate program to provide
animated output. A line in the output file of the simulation may, for
example, describe the action of an automatic guided vehicle by indicating in the column which represents the actions of that AGV the route on which it is on. (e.g. route 1). Such a route would be described in the route data as a number of increments or steps, each increment being a small enough step along the route to provide smooth animation relative to the size of the screen on which it is displayed. The first number in that particular line of simulation output data is a number which is the number of these increments which should be output during the current time step.

A separate program is required to process this file and 'drive' the entities which move around the layout according to the instructions in this simulation output file. This program acts upon each line of the file in turn, 'driving' each entity which is moving on a 'route' for one increment (or one movement) a time, until the number of increments is reached which was the first number in that line (the number of increments which should be output during the current time step). Because each entity which moves is moved by one step at a time together with the other entities, the viewer of the animated output is given the appearance of simultaneous smooth motion of the moving entities on the layout. This execution of one line of the file at a time is repeated until the end of the file is reached. This animation requires a graphics terminal which is capable of addressing a group of lines identified as a single number and that is capable of deleting or moving that group of lines by reference only to that single number identification (this is often known as 'sprite' graphics).
9.6 Hardware Operation.

The particular implementation of the software for a distributed simulator using the Intel SYS10 computer hardware is described here.

9.6.1 Start-up State.

Initially, two single board computers were used: an SBC 86/30 and an SBC 86/12A. The SBC 86/30 (which was supplied with the computer system) was located in a slot of the Multibus of higher priority than the SBC 86/12A. This latter board was provided with a copy of Intel's monitor program, SDM86, which is the program or code which is executed on start-up and ensures that the computer board is in a known state. It provides for control of the computer's registers and memory via instructions executed from a terminal connected to the serial input/output on the board. The typical instructions which can be executed by the monitor are to display memory; change memory; dis-assemble memory; copy memory from one location to another; examine registers; change registers; and single-step execution of instructions loaded into memory or ordinary sequential execution of instructions loaded into memory.

The other computer, the SBC 86/30, is allowed to boot-up the rmx86 operating system from disk into memory and to execute commands via a 'human interface' as normal. The human interface allows the operator using a terminal connected to the computer to load programs into memory and execute them. These programs can either be specifically located to start at a particular memory location or can be loaded into available free memory by letting the operating system choose where to locate the programs (located at run-time). The initial state of the
multiprocessor computer system is therefore that the SBC 86/30 computer has loaded the operating system into memory and is waiting for commands via a terminal, and that the SBC 86/12A computer is executing the monitor program which is also waiting for commands via a terminal (figure 28).

9.6.2 Loading the software.

The software is loaded as one complete program into memory by issuing the instruction to the master SBC 86/30 computer to load and execute an executable file from disk. This command is, in fact, the name of the executable file which consists of all the modules of the simulation program linked together. It is possible, when configuring the rmx86 operating system for a particular configuration, to reserve memory from the operating system for other purposes. If the operating system is allowed to load a program into memory, it will load it into free memory which is not reserved i.e. the memory that is known to the operating system. However, it is also possible to locate a program so that even when the operating system loads the program it is loaded into a particular area of memory (which has been previously reserved (see figure 29)).

9.6.3 Executing the software.

When the complete program consisting of all the modules has been loaded into memory, the control module becomes executable and begins executing. Its first task is to initialize the control and synchronization variables (lock1, lock2 and syncloc).
Multiprocessor System: Start-up State

Diagram showing a multiprocessor system with Intel i86 processor, local memory, and monitors.
Common Multibus Memory

Simulation Software

Intel i386 operating system

Local memory

monitor Intel SDM86

Master
Single Board Computer

Terminal

Local memory

monitor Intel SDM86

Slave
Single Board Computer

Terminal

Multiprocessor System:
State after loading Simulation Software

Figure 29
Subsequently some interactive input is requested to determine the amount of debugging required. The data relating to the particular simulation run is then read into memory from disk storage.

In the two processor system there are two control procedures. Each control procedure consists of instructions to

(i) output some acknowledgement that the procedure is executing (a '+' to the controlling terminal screen, for example);

(ii) execute the particular procedure or module called within it (or particular procedures or modules);

(iii) increment a counter variable to count the number of times the procedure has been executed;

(iv) increment a synchronization variable;

(v) wait until the synchronization variable is set to zero, or in the case of the master processor executing the master procedures, wait until the synchronization variable has reached the number of executable modules or processors in the system (because each processor increments the variable) and then execute the 'review' cycle.

The master control procedure also tests to determine whether interim output is required to the screen at this stage and outputs the data if required. It then sets the synchronization variable to zero (which the other processor(s) has/have been waiting for). These sequences of actions for the master control procedure and a control procedure which is not the master (a slave procedure) are shown.
diagrammatically in figure 30.

A slave control procedure also has a test to determine whether it is the first time that it has been executed. If it is, then the modules are not executed but an interrupt is caused. This is necessary to determine the address in memory at which the program has been located by the operating system.

The data for the simulation is read into memory from disk storage and then the slave control procedure is called (in a two processor system), and begins to be executed by the master processor. Because it is the first time the slave procedure is called, an interrupt is caused to the operating system. This interrupt causes execution of the current program to stop and control to be passed to the SDM86 monitor. This monitor allows registers to be examined and code to be displayed, dis-assembled, etc.

The registers 'CS', 'IP' and 'DS' (Current Segment, Instruction Pointer and Data Segment) are examined at this point by the operator. The second processor is at this time executing its own copy of the monitor and waiting for commands from a terminal. The registers CS, IP and DS are interactively changed on the second processor from the terminal to be the same values as those examined at the interrupt point of the master processor. This means that the slave or second processor, if given the command to execute code, will begin executing the same code that the master is currently ready to execute which in fact is the control procedure for that slave processor.

After the registers of the slave processor have been modified, the master processor is then instructed to continue processing. Because
Parallel Multi-processor Simulation

<table>
<thead>
<tr>
<th>Master processor</th>
<th>Slave processor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>the simulation cycle consists of the</td>
<td>the simulation cycle consists of the</td>
</tr>
<tr>
<td>following tasks:</td>
<td>following tasks:</td>
</tr>
<tr>
<td></td>
<td>execute the simulation procedures</td>
</tr>
<tr>
<td></td>
<td>output cycle confirmation signal</td>
</tr>
<tr>
<td></td>
<td>increment cycle counter</td>
</tr>
<tr>
<td></td>
<td>increment synchronisation counter</td>
</tr>
<tr>
<td></td>
<td>wait until processors are synchronised</td>
</tr>
<tr>
<td></td>
<td>execute executive cycle</td>
</tr>
<tr>
<td></td>
<td>re-initialise synchronisation counter</td>
</tr>
</tbody>
</table>

Synchronisation code of a multi-processor simulation

Figure 30
the slave procedure is being executed for the first time, the actual code of the simulation modules is not executed but passed over.

The masters' own control procedure for its simulation module is then executed (the slave processor is still waiting for an interactive instruction to execute code) and the length of simulation required before interim output to the screen is shown is interactively input (viz. 100 cycles etc.). The master's own simulation modules are then executed and the master waits until the synchronization variable has been incremented to equal the number of control procedures or processors, after incrementing it itself.

The slave processor is then instructed to process the code that its program counter registers (CS and IP) are pointing at (these were previously modified to point at the code in the slave procedure) (see figure 31). It executes its own simulation modules (because it is not the first time that the module or control procedure has been executed) and increments the synchronization variable which now is equal to the number of control procedures or processors.

This slave processor waits until the synchronization variable is set to zero. The master processor which was waiting for the synchronization variable to be incremented is now able to execute the review or simulation executive cycle and also to collect data for simulation animation or statistical output. After this executive cycle has completed execution, the master processor sets the synchronization variable to zero (which the slave processor has been waiting for) and both control procedures are begun again by each processor. This cycle is repeated until the number of cycles required is reached, and then the interim output is displayed on the screen
Common Multibus Memory

Slave control procedure and simulation modules

Master control procedure and simulation modules

Intel rmx86 operating system

Local memory

monitor Intel SDM86

Master Single Board Computer

Terminal

Local memory

monitor Intel SDM86

Slave Single Board Computer

Terminal

Multiprocessor System: State when running Simulation Software

Figure 31
after the master has executed the simulation executive cycle but before the synchronization variable is reset by it to zero.
10.1 Introduction

The purpose of this chapter is to describe the use of the distributed simulator in a case study. The application of the simulator to a case study was carried out in order that the proposition advanced in this thesis (i.e. that distributed simulation can be used to satisfy the criteria which have been developed for simulation, during the design stage, of highly automated batch manufacturing systems) could be tested.

In this chapter the case study system is described; configuration of the distributed simulator is described for the case study; and the use of the simulator and the collection of results is described.

10.2 Description of System

A number of highly automated batch manufacturing systems have been simulated using the distributed simulation system. One particular system which possessed most of the elements of a highly automated batch manufacturing system was that proposed by Normalair-Garrett Ltd.
/WILL82/, which included two Automatic Guided Vehicles (AGVS), nine machines and three pallet storage buffers. The original layout which was proposed was provided in a drawing supplied to the Dept. of Eng. Prod., Loughborough University. The system to be simulated is defined (for the purposes of the simulation) as including two machining centres, each with tool drums of capacity 80 tools, a milling machine, an inspection machine, and five lathes each with five tools available. One machining centre has a ten pallet loop conveyor type buffer; the other machining centre has a four pallet rotary buffer. The loop conveyor buffers is only able to rotate anticlockwise and the four pallet rotary buffer rotates clockwise. All the other machines do not have pallet exchange mechanisms but just single pallet load/unload mechanisms. Pallets are supplied to the system via a ten pallet linear storage buffer. One side of the buffer is for access by automatic guided vehicles, and the other side of the buffer is for access by operators who load (fresh) raw material on to the pallets after removing any completed machined parts. There are two automatic guided vehicles, each of which has a 'home' station to which it would return either on request, or if recharging of the battery is required (see figure 32).

The automatic guided vehicles are able to circulate in any direction around the system and there are no 'sidings' from the main track which agv's may take to reach the docking point with a machine; they dock on the main track at each machine. Each pallet can hold a number of different types of parts, all of which must follow the route defined for that particular pallet.
The part storage buffer for introducing pallets into the system, at start up, contains all the pallets which are to be machined in the system. When a pallet has completed its processing (and its journey round the system) it is returned to this buffer, and after a time, 'regenerated' and the pallet becomes 'live' and available for processing again.

10.3 Configuration of the Simulator

10.3.1 General Layout Configuration

The simulator is configured on the basis of a plan of the proposed manufacturing system to be simulated. The plan is numbered in the following manner. Each machine or processing device (including non-cutting machines such as inspection machines) is given a number, commencing with number one (1) (see figure 33). There is no rule which defines which machine should be number 1 or any particular number. However, for ease of reference obviously a rational numbering system should be used.

Each pallet storage buffer or load/unload device is given a number in a similar manner to the already numbered machines i.e. buffer one is at machine one; buffer 2 is at machine 2; .......; buffer 9 is at machine 9; and buffer 10 is not connected with a machine (see figure 34). Each of these entities is then given a 'station number', commencing with buffer 1 or load/unload device 1. This means that buffer 1 is station 1; buffer 10 is station 10; machine 1 is station 11; and machine 9 is station 19 (see figure 35).
The AGV docking stations are then numbered in a similar manner to the numbering of the buffer devices so that: AGV station 1 is the docking point with buffer 1; AGV station 2 is the docking point with buffer 2; AGV station 3 is the docking point with load/unload device 3; ........; AGV station 10 is the docking point with buffer device 10; and when all the AGV stations which are docking points with buffer and load/unload devices have been numbered than all the other AGV stations are numbered consecutively upwards. Other AGV stations are junctions and home or parking positions (see figure 36).

10.3.2 AGV data and Animation route data configuration

Following the numbering of the stations, the routes which pallets and automatic guided vehicles (AGV's) take to travel between stations are numbered. All the routes that it is possible for the AGV to take are numbered first (see figure 37), and then following on from those numbers the routes which pallets must travel on to get to machines via buffers or load/unload devices are numbered (see figure 38). Data relating to those routes can then be entered into an interactive program (the data is the the start and end position of each route; which AGV station the route goes from; which AGV station the route goes to; what sort of route it is (straight; circular or arc; clockwise or anticlockwise etc.). The interactive program generates two data files. The first is the data file 'rf': (the route description file for the transport network data); the second is a file 'am07.dat' which is a description of each route in terms of the incremental steps required to get from start to finish (this is used by the graphics animation program).
Case Study Layout - Buffer Route Numbering

Figure 38
The route description file can then be processed by another configuration program (called 'genpf'). This produces the default proximity data file. This default proximity file can be subsequently modified by the user of the simulation system if preference is to be given to the use of certain routes by the transport vehicles (for example, to make the AGV's circulate round a loop in the AGV track preferentially in a clockwise fashion but not to eliminate the possibility of travelling in an anticlockwise direction). The default proximity file assumes equal weighting to all routes. An example of an interactive session with the route generation program is given in appendix II.

10.3.3 Layout for graphical animation

The layout which is to appear on the screen of the device used for animated graphic output is initially produced using a proprietary CAD system which has facilities for IGES (Initial Graphics Exchange Specification) file creation. The drawing is produced in the normal manner with the CAD system although only the parts of the layout which are to remain static are drawn at this time (i.e. not the pallets which move; or AGV's which move; or lines that show buffer rotation). A post processor is normally used within the CAD system to produce the IGES file which can then be processed by an additional configuration program called 'CGIS' (CGIS is a program written by the design-aid project team). This program produces a data file which can be used by a graphics generator program dedicated to the particular graphics terminal which is to be used.
The graphics data file called 'am06.dat' produced by 'CGIS' is then processed by another configuration program called 'genfort06.dat', in order to add a boundary or border to the picture and a time display area to show the time elapsed during the simulation. A further program, 'gen09', is used to add the moving segments to the picture. These are pallets, AGVs and lines for animating the rotation of buffers. A log file is produced which indicates which segments are which AGVs, pallets and lines, and a file called am09.dat, which is a list of the animated segments in the layout.

During the input session described earlier, these 'collections of lines' which are to be moved around the screen or layout are interactively added to the basic static layout and each is given a unique number by which it can be referenced. Each of these 'collections of lines' is called a segment (in GKS notation). The user, therefore, does not need to know which segment is which, since these segments and their reference numbers are generated automatically during the interactive input session.

The graphics generator program to drive the IMLAC vector refresh graphics terminal used by the simulator during the work described in this thesis was also produced by the design-aid project team and was called 'CRIMP'. It provides facilities to draw lines, circles, arcs, text and rectangles and to open and close segments.

10.3.4 Configuration Process or Technique.

The process of configuring the simulation software for a particular simulation is in two stages, starting from the point where all the necessary data files which describe the simulation have been produced.
This two stage technique firstly produces a number of one line PASCAL statements which are the 'reset' and 'rewrite' statements used to read and write from a particular file. These statements are always located in the same directory in the tree structure of the simulation (figure 39) and each program in the simulation uses a compiler control called 'Sinclude' to actually include the text of these statements, by including the file name, the text of which is the statement.

The files (and thus the statements) are produced by a program 'config1' which interrogates the user which simulation number is to be configured. This simulation number is then inserted into the PASCAL reset or rewrite statement and the whole statement is written to the file in the configuration directory. All the statements are produced at this time for all of these types of 'includes'.

The program 'config2' is then compiled, so that the 'included' statements take effect, and when linked is executed. This second program reads a number of the data files in order to count the entities such as number of parts, number of pallets, number of buffers, number of machines, etc. These numbers are then written into more files which contain PASCAL statements as constants (e.g. 'noofmachines = 10 ;' would be the text of a file called 'machno'). The files are 'included' in a list of the global constants in the simulation system so that when the simulation programs are compiled, the statements become operative (e.g. 'noofmachine = 10 ;') and thus the global constants are set. When this second program is executed, it also provides the user with the opportunity to check that the number of machines, pallets, parts etc. do in fact agree with the
Directory 'emprog'

Directory 'slmconfig'

config programs
config files machno agvno buhno

eg. source no_of_machines = 10;

The files in this directory are configured for a particular simulation

Simulation configuration directory tree structure

Figure 39
number expected.

The reason why this method of recompilation of the simulation programs and the program 'config2' was chosen in order to configure the simulation was that using this method there are no predefined limits in the simulation system on the number of machines, tools, parts, pallets etc. that can be used in the simulation model. The only effective limits are those set by the language, compiler, operating system and computer used which would apply to any simulation system running on that particular computer system. Predefined limits within the simulation system can be considered to be one of the greatest drawbacks of simulation systems which are easy to configure (eg. MAST./CITR84/).

10.4 Loading and executing the software.

The software is loaded as one complete program into memory by issuing the instruction to the master SBC 86/30 computer to load and execute an executable file from disk. This command is, in fact, the name of the executable file which consists of all the modules of the simulation program linked together.

10.4.1 Executing the software.

A more detailed explanation of what actions occur at each point of the sequence of starting the distributed simulation system has been given in the previous chapter. A brief description of this sequence is appropriate here.
When the complete program consisting of all the modules has been loaded into memory, the control module becomes executable and begins executing and some interactive input is requested to determine the amount of debugging required. The data relating to the particular simulation run is then read into memory from disk storage.

Next, the none-master or slave control procedure is called (in a two processor system), and begins to be executed by the master processor and an interrupt is caused to the operating system. This interrupt causes execution of the current program to stop and control to be passed to the SDM86 monitor. The registers 'CS', 'IP' and 'DS' (Current Segment, Instruction Pointer and Data Segment) are examined at this point by the operator and are interactively changed on the second processor from a terminal to be the same values as those examined at the interrupt point of the master processor. After the registers of the slave processor have been modified, the master processor is then instructed to continue processing.

The masters' own control procedure for its simulation module is then executed (the slave processor is still waiting for an interactive instruction to execute code) and the length of simulation required before interim output to the screen is to be shown is interactively input (viz. 100 cycles etc.).

The slave processor is then instructed to process code and the simulation begins executing, and continues until the interim output is required. Further interactive input is requested to find out if the simulation is to be continued. If it is, then the simulation is executed again until the number of simulation cycles have been completed and then the interim results are output again.
If the simulation is not to be continued, then the operator or user is asked whether the results should be output to a file on disk, and appropriate action is taken before the simulation is completed and control of the computer is returned to the normal operating system.

10.4.2 Characteristics of the data used in the case study.

The data used for the operation times and numbers of operations required by the parts in the case study was loosely based on some representative parts which could have been manufactured in the proposed manufacturing system. The characteristics of no actual part produced by Normalair-Garrett Ltd. can or should be inferred from the part characteristics described here. The summary of the characteristics of the parts, buffers and automatic guided vehicle route layout is given in table 2.

10.4.3 Time increment of the simulation.

The time increment used for the case study was that one cycle of simulation corresponded to a time advance of one minute (i.e. the simulation time was incremented in steps of one minute). This time increment should be considered in connection with the smallest time step required by the data, i.e. the time required for a pallet exchange by a pallet exchange buffer. The time step was chosen to be the closest integer time step to the duration of the motion of a pallet exchange buffer. The simulation system described in this thesis does not preclude the use of a real (not integer) value for the time step, but in this particular case study, which was to be used to test the operation of the distributed simulator, the integer value
## Characteristics of the case study data

### Parts data

10 Pallets

- 10 Parts (1 part per pallet)
- 5 Parts of type A : 5 Parts of type B

### Characteristics of a type A part:

- **Average set up time**: 2 min.
- **Number of operations**: 9 operations
- **Average operation time**: 17.5 min.
- **Average number of sub-ops per operation**: 7 sub-ops

### Characteristics of a type B part:

- **Average set up time**: 2 min.
- **Number of operations**: 5 operations
- **Average operation time**: 12 min.
- **Average number of sub-ops per operation**: 5 sub-ops

### Characteristics of the motion of the buffers

- **Time required to move pallet one position on a rotary buffer**: 1 min.
- **Time required to exchange pallet on/pallet exchange buffer**: 1 min.
- **Time required to load pallet to/from a linear buffer**: 1 min.

### Characteristics of the motion of the AGVs

- **Speed of an AGV**: 10 length units / time increment
- **Average length of a route**: 14.4 length units
- **Average number of routes between two stations**: 4.3 routes
- **Average duration of AGV travel**: 6.2 time increments (6.2 min.)
of one minute was considered sufficiently accurate.

10.5 Case Study - Simulation Output.

The output of the simulation is then available to be post-processed. Examples of the output of the simulation in terms of the output data file, graphical animated output and statistical presentation are given. The reasons why graphic animated output was considered important are described in the next section.

10.5.1 Validation of the Operation of the Simulation.

It is important that the output from a simulation has the confidence in it from the people who will use the output to make decisions such as to buy another machine, or another AGV etc. There are several techniques which can be used to obtain this confidence. One example of these techniques is sensitivity analysis, where the sensitivity of the simulation model to changes in input variables or parameters is tested to determine whether it reacts in the expected manner (for example, in a simple case, double the machines to process a workload should double the throughput - assuming transport and tooling availability). However, in a complex system, one of the reasons for building the model is to determine whether it is sensitive to some of the inputs, and as the complexity of the model increases, it becomes increasingly difficult to determine whether unexpected changes in the output of the simulation model are due to valid factors (such as tooling or transport unavailability, deadlock etc.) or are due to inaccuracies or logical mistakes in the model.
One way to improve the understanding of the operation of the model (and therefore improve the understanding of the results obtained from the model) is to provide graphical animation so that it is possible to see a sequence of operations happening in the model and to be confident that the operations are actually occurring in the model. The level or sophistication of the graphical output also improves the 'accessibility' of the display to people who require the output of the simulation model but do not have the time or skill to understand the abstraction of the model from the real world either in the coding of the model itself, in an activity cycle diagram as used with ECSL/CLEM82/; or in an abstract graphic or visual representation/STAN86/.

However, it is also possible to provide too much graphic output which can 'overload' the viewer with too much information while viewing the display. Animated graphic output was provided in the simulation system described in this thesis for several reasons, which were:

- debugging;
- output presentation to people not intimately involved with the work;
- understanding of the operation of a complex system and in particular the effect of decision rules on the operation; and
- validation of the operation of the simulation.

The level of detail of the graphic output was limited to pallet movements, AGV movements, rotary buffer movements, and machining (being indicated by a flashing or blinking pallet). This animation
was based on a scaled layout of the simulated system. The movements of pallets, AGV's and buffers was designed to be as realistic as possible (i.e. apparently smooth incremental motion over the length of a route) to maximise the 'accessibility' of the display to the viewer. Further details regarding the animation have been described in the previous chapter.

This output was considered to be a sufficient validation of the operation of the model. Its main use is during the debugging phase, where if a part was not released from a buffer or another part was attempted to be loaded on to an already full buffer, for example, were cases which were quickly identified and remedied (usually by an alteration to the control rules). These types of cases mainly occurred when several things occurred simultaneously and the control rules were too naive or unsophisticated to cope. These situations obviously only occur in complex systems with a sufficient number of machines, AGV's, buffers etc. in order for simultaneous independent events to take place. The number of these types of situations that occur is also dependent on the time 'discretization' or time step of the model (e.g. seconds, minutes or hours). From the point of view of developing sophisticated control rules it is advantageous to identify these situations and control them in a predetermined way, however it is unknown in practice how often these types of events occur. Deadlock, however, can occur whatever the time step of the model and animation is particularly useful to identify this type of problem during the debugging phase.
CHAPTER 11

RESULTS

11.1 Introduction

The results to be described in this chapter are in four areas. These areas are: software technique; hardware technique; the user interface; and speed of execution.

The distributed simulation system described here was built using a number of published layouts or 'case studies' as standpoints to fix or view the progress of the simulator: the case study which embodied most of the elements of highly automated batch manufacturing systems is the one described in the previous chapter. The results to be discussed here are related to this case study.

11.2 Results in the area of software technique.

The results of using the software techniques described in this thesis can be described by considering the functionality of the modules which can be concurrently executed; by considering the amount of detail which is included in the simulator; and by considering the operation of the simulation executive.
11.2.1 Functionality of the modules.

The novel approach adopted here has successfully distributed the activities related to particular types of entities in a modular way, so that all activities which are related to a particular type of entity are collected together in one module. The modules are then distributed for the purpose of computer processing. The particular types of entities for which activities were collected together into modules were: machines; load/unload and storage buffers; automatic guided vehicles; decision making activities. The operation of the modules has been described in chapters 8 and 9.

11.2.2 Level of detail.

The result of using the software techniques described in this thesis to obtain a high level of detail in the distributed simulator can be described by the activities in which entities are involved (previously described in chapter 9), and by the particular types of entities that are dealt with by the simulator. These are:

In the machining module:

- specific tools are identified during machining operations so that the use and wear of the individual tools can be monitored;
- the use of the tool magazine on the machine is explicitly modelled;
- the pallets used to carry the parts through the machining system are able to carry amounts of different parts on the
In the load/unload and storage buffer module:

- particular pallets can be placed in particular positions in a pallet storage buffer;
- rotary pallet storage or load/unload buffers which move only in a particular direction (clockwise or anticlockwise) are explicitly modelled.

In the AGV module:

- AGV's with particular speeds are modelled;
- the routes on which AGVs travel can be given a 'weight' which describes a directional preference (i.e. the same route could be more preferable to AGV's travelling in one direction than the other, if there is a choice of routes);
- each AGV can have a 'home' station, to which it would return if not in use and in danger of causing a blockage.

In the decision module:

- decisions about which pallet is to be released from a pallet storage or load/unload buffer are made individually i.e. for each buffer;
- two stages of decision making (i.e. two decision points) are provided: the first which chooses a selection of pallets to leave a buffer; the second which applies a rule
to choose from within this selection.

In the control module (simulation executive):

- there is specific identification of deadlock situations and their explicit solution.

These results, regarding the level of detail included within the distributed simulator, can be put into context by considering the intended use of the simulator and also by comparing the level of detail which is included in simulators which are meant to be used for a similar purpose.

The intended use of the simulator is during one of the stages in the design of highly automated batch manufacturing systems. This stage of design has already been described in chapter 3; the stage includes the assessment of operating strategies and control strategies assuming a previously designed layout and system configured with regard to overall capacity (number of machines etc.). The amount of detail incorporated into the simulator was influenced by a consideration of the amount of detail which was desired during this stage of design and hence sufficient detail has to be included in order to test different strategies.

This means that individual tools should be defined; automatic guided vehicle routes should be defined; pallets should carry assorted numbers and types of parts; individual pallet storage buffers should release parts according to an individual strategy for each buffer; deterministic data should be used for processing times etc.
The context of the results is also clarified when compared with a simulator which deals with a similar class of problems (including the same types of pre-defined entities) an example of which is MAP/1/PRIT85/. A particular comparison of the level of detail could be the transport system: the distributed simulator enables a precise network of the transport system to be defined which facilitates the effects of automatic guided vehicles blocking the paths of the other vehicles. This is not possible using MAP/1 at the time of writing: it is only possible to define a distance from one station to another.

Another comparison of the level of detail could be the inclusion of data relating to individual tools within the system so that analysis is possible of part routing based on tooling availability, etc. Individual tools as such are not included as primary entities in MAP/1.

The amount of detail which is included in the class of simulators like MAP/1 (pre-configured for a certain class of problems) is related to the ease-of-use of the simulator and the speed with which a new model can be constructed. However, it is usually possible to include any amount of detail in any simulation which is expressed in a simulation language (as opposed to a pre-defined system). This is a question of length of program, time spent in programming, compiler capacity etc., and hence there is usually a 'trade-off' between the amount of detail which is included and the ease-of-use of the simulator.

The modular arrangement of activities in the distributed simulation system possesses some of the positive attributes of both techniques (a pre-configured simulator and the flexibility of a simulation language) in order to maximise the level of detail which is
The result of using of the software techniques described in this
thesis for the simulation executive was that an executive was
constructed which uses an incremental time step of equal duration. At
each of these time steps, an executive cycle is executed. The number
of tests or checks that are carried out by this type of executive
cycle is larger than the number of tests carried out in a conventional
discrete event-based simulation where time is advanced in irregular
steps to the time of the next event. However, in a conventional
discrete event-based simulation a list processor is also required to
determine which event is due next to begin; this type of list
processing is not required in the approach described here.

11.3 Result of the study of hardware techniques.

The result of the research into hardware techniques for a distributed
simulator using techniques of multiprocessing was that in order to
achieve true parallelism (i.e. separate computers working on
different parts of the task at the same time) a multiprocessor
computer system was required. A number of tests of distributed
simulation systems have been undertaken using multi-tasking computers
(with a single processor) but these can only take advantage of the
modularity of the distributed simulation approach and not of the
inherent parallelism of the approach. The hardware options for
communication between the processors in the multiprocessor system
included a tightly coupled multiprocessor computer using a common bus.
for communication or a loosely-coupled system using a standard proprietary network. The network was not chosen, because of the time overhead involved on the amount of communication required between the modules. The duration of this communication, consisting of the checks to be carried out during the executive cycle, would be minimised by the use of a closely-coupled system.

The hardware configuration chosen (Intel system 310 computer with additional single board computers each with local memory) is described in more detail in chapter 8, which was the result of the investigation into possible hardware configurations to support a distributed simulation technique.

11.4 The User Interface.

The 'user interface' of the distributed simulator is a term used to describe the facilities that are available to the user in order to use the simulator in its area of application. The techniques of distributed simulation described in this thesis have resulted in the following facilities being provided in the 'user interface':

- the configuration of the simulator in order to simulate a particular machining system;
- the configuration of the simulation activities into modules which are suitable for multiprocessing;
- the start-up and use of the multi-processor system;
- the collection of statistics; and
- the use of animation for verification.
11.4.1 Configuration of a particular simulation model.

The facilities which are provided for the user to configure the simulator for the simulation of a particular machining system enable the user to completely configure the particular model from data. This facility is restricted to a particular class of problem i.e. that of highly automated batch manufacturing systems. This restriction is usual when the preconfigured activities are provided for the user since not every possible entity can be imagined and included that the user might wish. However, the ability to configure the simulation from data enabled the connection of the simulation system to an interactive simulation model specification tool, described in chapter 7.

11.4.2 Configuration of the modules.

The facility which is provided to the user to configure the simulation activities into modules which are suitable for multiprocessing is related to the following questions: which modules run in parallel?; which serially?; and how easy is it to change this?.

The facility was provided in a straight-forward way by the use of a control program in which procedures can be created which can relate to a particular set of modules which are required to be run in parallel i.e. there is one control procedure for each collection of modules required to run in parallel, or for each processor. Within each of these procedures, if more than one simulation module is invoked then those modules called or invoked will run serially and the order of invocation can be changed simply.
The simulation executive and statistics/animation data routines can also be invoked serially following the execution of the control procedures (see chapter 9). The software distribution of the simulation modules is therefore a straightforward task.

11.4.3 **Start-up and running of the simulator.**

The facilities provided for the start-up and running of the simulator on the 'prototype' or Intel experimental hardware described in chapter 9 is straightforward, but not very convenient, in that a VDU terminal must be connected to each processor board during the start up phase of the simulation in order to set the registers of each processor to the start address of its particular control procedure (which in turn invokes the simulation modules within its control procedure). Each processor board must also contain a monitor to enable these register changes to be made. It is possible to automate this technique so that each processor will jump to a start address given to it by the master processor without manual intervention via a terminal VDU.

11.4.4 **Collection of statistics.**

The facility provided to the user for the collection of statistics is also relatively straightforward but is not automatically included in the simulation system. A list of events and the time at which those events occurred is provided as the output of the simulation system. It is possible to extract statistics required using a 'post-processor'-type program. A program which collected the utilisation of the machines, buffers and AGV's is included in the
collection of simulation software described in this thesis but the primary aim of the output file and the post processing was to provide some animated output of the operation of the simulation model. It is also possible to compare the ease-of-use of the statistic collection routine with the facilities provided by commercial simulation packages (e.g. 'FORECASP' /DAHM82/). It is usually possible with commercial simulation packages to include histogram generation routines with the model to build up standard histogram output of selected variables and to examine the mean, standard deviation of throughput time, for example. These are standard routines in commercial packages; they are not provided as standard in the distributed simulation system described in this thesis.

11.4.5 Validation and Verification.

The facilities provided for validation and verification of the operation of the simulator are described in chapter 10. Animated output is provided on a computer terminal capable of graphic output so that the actions of material handling equipment such as AGVs can be studied, and so that the user can develop confidence in the simulation model (that actions are occurring when and where they are supposed to occur).

11.5 The speed of execution of the simulation model.

The objective of measuring the speed of execution of the simulation model was to find out what advantage there was in terms of improved response due to multiprocessing. These measurements would also show the amount of 'overhead' due to the operation of the simulation
executive and communication overhead and hence determine the validity of the original hypothesis i.e. that the distribution of the activities of the simulation model could lead to an increase in efficiency with the use of a tightly or closely coupled multiprocessor computer system.

The measurements taken for this purpose were a comparison, for this particular case study, of the duration of time taken to execute a number of cycles of simulation of the case study. In order to determine which of the simulation modules required the largest processing power applied to it i.e. which module required the longest execution time (of those which could be processed in parallel), a number of measurements were taken of the time taken to execute the simulation with varying arrangements of the simulation modules. For each of these arrangements of modules, two different cases were measured: with the generation of animation data, and without generation of animation data. This was because the procedure which generates animation data was invoked sequentially after the simulation executive cycle and therefore incurred an additional overhead since no other procedures were executed in parallel.

It was also possible to animate the operation or results of the execution of the simulation model in each case in order to be completely sure that there were no unforeseen side-effects of rearranging the order or sequence of execution of the modules, or to determine that in fact the modules could be operated in parallel. This is an important point, since without this type of test it is never possible to be completely sure that modules designed to run in parallel but executed sequentially do not have unknown side effects
unless adequate and sometimes sophisticated precautions are taken (such as compiler protection of, and identification of common variables between the modules). In this case it was possible to be completely sure that the modules could be executed in parallel with no side-effects, from the fact that the same results were obtained each time.

11.5.1 The number of cycles used in the simulation experiments.

The number of simulation cycles executed in the simulation experiments was 800 cycles. This corresponds to 800 time increments (one time increment per simulation cycle). As previously described in chapter 10 (section 10.4.3) each time increment corresponded to one minute of simulated time for the case study described in this thesis: 800 cycles of simulation is therefore equivalent to 800 minutes of simulation time, i.e. 13 hours and 20 minutes. In the context of the average time required to complete a part (see section 10.4.2, chapter 10) this simulation period provided enough time for the simulation to be 'run in' i.e. reach a fully loaded, and therefore representative operating situation.

11.5.2 Number of events in the simulation experiments.

The number of events that occurred in the 800 minutes of the simulation experiments is best expressed by describing the number of times that particular activities were started. The separate activities that each type of entity is involved in are described in chapter 9 (section 9.2). For the activities of machines, the number of occurrences of the activities depends on the number of operations...
and sub-operations of each part. The following values are based on the actual data used for the case study, and can be related to the part data average characteristics previously described in section 10.4.2:

- the activities of 'loading', 'setting up', 'completing operation' and 'unloading' were started 70 times; and
- the activities 'tool changing' and 'cutting' were started 430 times.

For the activities of the buffers, the number of occurrences of the activities is similarly related to the number of operations of the parts:

- the activities 'load pallet to machine' and 'unload pallet from machine' were started 70 times;
- the activities 'load pallet from AGV', 'move pallet to load/unload position' were started 80 times; and
- the activity 'moving buffer' was started 300 times.

For the activities of the AGVs, the specific number of occurrences of the activities depends upon the order of introduction of the pallets into the system and the priorities allocated to certain AGV routes. The following values are an indication of the usual number of AGV activities that occurred during this case study:

- the activities 'commence agv service' and 'completing move' were started 80 times;
- the activity 'travelling' was started 344 times;
- the activity 'change route' was started 264 times; and
- the activity 'waiting for related activity' occurred only when deadlock occurred, and was started once in this case study.

Following from these values, the total number of activities that were typically started in the simulation experiments was 2439.

11.5.3 The simulation experiments.

11.5.3.1 Configuration of the experiments - software.

The different configurations of the simulation system which were used are presented in tables 3 and 4. These tables show, for a particular simulator experiment, which simulation modules were collected together in which control procedure (the function of the different control procedures has been previously described in chapter 9).

The simulation modules which could be configured were the decision, buffer, agv, and machine simulation modules, and each configuration could run with or without the animation module. Each combination of modules which forms a particular experiment has a particular control module and a unique name given to the simulation program, such as 'simxy.86'. The 'x' identifies the arrangement of the modules: the second figure 'y' identifies whether animation generation was included (y = 0 with animation, y = 1 without). The seven different arrangements that were studied were designed to show the effects of the distribution of the different modules on the speed-up and efficiency of the distributed simulator.
<table>
<thead>
<tr>
<th>Name of simulation</th>
<th>Name of control module</th>
<th>Simulation Modules in control procedure 1</th>
<th>Simulation Modules in control procedure 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Slave processor SBC 86/12A</td>
<td>Master processor SBC 86/30</td>
</tr>
<tr>
<td>sim00.86</td>
<td>conmod00.86</td>
<td>none</td>
<td>all modules</td>
</tr>
<tr>
<td>sim01.86</td>
<td>conmod01.86</td>
<td>none</td>
<td>all except animation</td>
</tr>
<tr>
<td>sim10.86</td>
<td>conmod10.86</td>
<td>decision module</td>
<td>buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>agv</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules animation</td>
</tr>
<tr>
<td>sim11.86</td>
<td>conmod11.86</td>
<td>decision module</td>
<td>buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>agv</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td>sim20.86</td>
<td>conmod20.86</td>
<td>decision module</td>
<td>buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>agv</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td>sim21.86</td>
<td>conmod21.86</td>
<td>decision module</td>
<td>buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>agv</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>module</td>
</tr>
<tr>
<td>sim30.86</td>
<td>conmod30.86</td>
<td>decision module</td>
<td>buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>agv</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td>sim31.86</td>
<td>conmod31.86</td>
<td>decision module</td>
<td>buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>agv</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>module</td>
</tr>
<tr>
<td>sim40.86</td>
<td>conmod40.86</td>
<td>decision module</td>
<td>buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>agv</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td>sim41.86</td>
<td>conmod41.86</td>
<td>decision module</td>
<td>buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>machine</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>modules</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>agv</td>
</tr>
</tbody>
</table>

Configurations of the simulation system

Table 3
<table>
<thead>
<tr>
<th>Name of simulation</th>
<th>Name of control module</th>
<th>Simulation Modules in control procedure 1</th>
<th>Simulation Modules in control procedure 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Slave processor SBC 86/12A</td>
<td>Master processor SBC 86/30</td>
</tr>
<tr>
<td>sim50.86</td>
<td>conmod50.86</td>
<td>decision ) modules machine ) agv )</td>
<td>decision ) module</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review ) modules animation )</td>
</tr>
<tr>
<td>sim51.86</td>
<td>conmod51.86</td>
<td>decision ) modules machine ) agv )</td>
<td>decision ) module</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review ) module</td>
</tr>
<tr>
<td>sim60.86</td>
<td>conmod60.86</td>
<td>decision ) modules machine ) agv )</td>
<td>decision ) modules buffer )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review ) modules animation )</td>
</tr>
<tr>
<td>sim61.86</td>
<td>conmod61.86</td>
<td>decision ) modules machine ) agv )</td>
<td>decision ) modules buffer )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review ) module</td>
</tr>
<tr>
<td>sim70.86</td>
<td>conmod70.86</td>
<td>decision ) modules agv )</td>
<td>decision ) modules buffer )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review ) modules machine )</td>
</tr>
<tr>
<td>sim71.86</td>
<td>conmod71.86</td>
<td>decision ) modules agv )</td>
<td>decision ) modules buffer )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>review ) module</td>
</tr>
</tbody>
</table>

Configurations of the simulation system
11.5.3.2 Configuration of the experiments - hardware.

Each of the control procedures was allocated to a particular single board computer (SBC). In these experiments, using two SBC's, there were two control procedures. Control procedure 1 was allocated to the SBC 86/12A. This single board computer used an 8086 CPU and 8087 numeric coprocessor running at speed of 5 MHz. Control procedure 2 was allocated to the master SBC, the 86/30 which also used an 8086 CPU and 8087 numeric co-processor running at a speed of 8 MHz.

11.5.3.3 Simulator speed measurements.

The corresponding measurements of the time taken to execute 800 cycles of simulation are presented in table 5. The measurements were taken manually, with a stop-watch, to an accuracy of + or − 0.1 secs. of each reading: two readings were taken for each measurement. This technique was considered to be sufficiently accurate for the purpose of measuring the speed of execution of the model over a length of time of 2-3 minutes. The 'worst-case' accuracy was the largest difference between the two readings, which was 0.4 seconds over a period of 159 seconds, i.e. 0.25 %.

Table 6 shows the duration of the animation cycle (for each experiment which included the generation of animation data) and the speed-up and efficiency of the multiprocesssing arrangement for each experiment. The duration of the animation cycle is the difference between the speed of execution of a particular arrangement of modules (experiment) and its corresponding experiment without animation.
<table>
<thead>
<tr>
<th>Name of simulation</th>
<th>First time measurement</th>
<th>Second time measurement</th>
<th>Difference</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim00.86</td>
<td>216.8</td>
<td>216.4</td>
<td>0.4</td>
<td>216.6</td>
</tr>
<tr>
<td>sim01.86</td>
<td>202.4</td>
<td>202.4</td>
<td>0.0</td>
<td>202.4</td>
</tr>
<tr>
<td>sim10.86</td>
<td>159.0</td>
<td>158.6</td>
<td>0.4</td>
<td>158.8</td>
</tr>
<tr>
<td>sim11.86</td>
<td>144.6</td>
<td>144.4</td>
<td>0.2</td>
<td>144.5</td>
</tr>
<tr>
<td>sim20.86</td>
<td>164.0</td>
<td>163.9</td>
<td>0.1</td>
<td>164.0</td>
</tr>
<tr>
<td>sim21.86</td>
<td>150.0</td>
<td>149.8</td>
<td>0.2</td>
<td>149.9</td>
</tr>
<tr>
<td>sim30.86</td>
<td>171.1</td>
<td>171.1</td>
<td>0.0</td>
<td>171.1</td>
</tr>
<tr>
<td>sim31.86</td>
<td>157.3</td>
<td>157.1</td>
<td>0.2</td>
<td>157.2</td>
</tr>
<tr>
<td>sim40.86</td>
<td>178.9</td>
<td>178.9</td>
<td>0.0</td>
<td>178.9</td>
</tr>
<tr>
<td>sim41.86</td>
<td>165.1</td>
<td>164.7</td>
<td>0.4</td>
<td>164.9</td>
</tr>
<tr>
<td>sim50.86</td>
<td>193.5</td>
<td>193.3</td>
<td>0.2</td>
<td>193.4</td>
</tr>
<tr>
<td>sim51.86</td>
<td>179.3</td>
<td>179.1</td>
<td>0.2</td>
<td>179.2</td>
</tr>
<tr>
<td>sim60.86</td>
<td>199.2</td>
<td>199.0</td>
<td>0.2</td>
<td>199.1</td>
</tr>
<tr>
<td>sim61.86</td>
<td>184.7</td>
<td>184.7</td>
<td>0.0</td>
<td>184.7</td>
</tr>
<tr>
<td>sim70.86</td>
<td>207.8</td>
<td>207.8</td>
<td>0.0</td>
<td>207.8</td>
</tr>
<tr>
<td>sim71.86</td>
<td>193.5</td>
<td>193.5</td>
<td>0.0</td>
<td>193.5</td>
</tr>
</tbody>
</table>
Comparison of simulation speed measurement over 800 cycles

<table>
<thead>
<tr>
<th>Name of simulation</th>
<th>Duration (secs)</th>
<th>Duration of animation cycle</th>
<th>With animation</th>
<th>Without animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim00.86</td>
<td>216.6</td>
<td>14.2 secs</td>
<td>57.8 secs</td>
<td>57.9 secs</td>
</tr>
<tr>
<td>sim01.86</td>
<td>222.4</td>
<td></td>
<td>1.36</td>
<td>1.40</td>
</tr>
<tr>
<td>sim10.86</td>
<td>158.8</td>
<td>14.3 secs</td>
<td>52.6 secs</td>
<td>52.5 secs</td>
</tr>
<tr>
<td>sim11.86</td>
<td>144.5</td>
<td></td>
<td>1.32</td>
<td>1.35</td>
</tr>
<tr>
<td>sim20.86</td>
<td>164.0</td>
<td>14.1 secs</td>
<td>45.5 secs</td>
<td>45.2 secs</td>
</tr>
<tr>
<td>sim21.86</td>
<td>149.9</td>
<td></td>
<td>1.27</td>
<td>1.29</td>
</tr>
<tr>
<td>sim30.86</td>
<td>171.1</td>
<td>13.9 secs</td>
<td>37.7 secs</td>
<td>37.5 secs</td>
</tr>
<tr>
<td>sim31.86</td>
<td>157.2</td>
<td></td>
<td>1.21</td>
<td>1.23</td>
</tr>
<tr>
<td>sim40.86</td>
<td>178.9</td>
<td>14.0 secs</td>
<td>23.2 secs</td>
<td>23.2 secs</td>
</tr>
<tr>
<td>sim41.86</td>
<td>164.9</td>
<td></td>
<td>1.12</td>
<td>1.13</td>
</tr>
<tr>
<td>sim50.86</td>
<td>193.4</td>
<td>14.2 secs</td>
<td>17.5 secs</td>
<td>17.7 secs</td>
</tr>
<tr>
<td>sim51.86</td>
<td>179.2</td>
<td></td>
<td>1.08</td>
<td>1.10</td>
</tr>
<tr>
<td>sim60.86</td>
<td>199.1</td>
<td>14.4 secs</td>
<td>8.8 secs</td>
<td>8.9 secs</td>
</tr>
<tr>
<td>sim61.86</td>
<td>184.7</td>
<td></td>
<td>1.04</td>
<td>1.05</td>
</tr>
<tr>
<td>sim70.86</td>
<td>207.8</td>
<td>14.3 secs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sim71.86</td>
<td>193.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average duration of animation cycle = 14.2 seconds

Table 6
The speed-up is a ratio of the speed of the benchmark experiment to the speed of a different experiment which achieves the same result. The benchmark experiments with and without animation are sim00.86 and sim01.86 respectively, both of which were arrangements in which the second processor was idling and all the modules were executed sequentially on one processor. These experiments, however, did include the control procedures necessary for multiprocessing (i.e. they included code for synchronisation etc.) so that a true comparison could be made (the influence of the synchronisation code on the speed of execution of the programs is estimated below in section 11.6).

The speed-up is given by:

\[
\text{speed-up} = \frac{\text{benchmark execution time}}{\text{actual execution time}}
\]

and its maximum value is equal to the number of processors (i.e. speedup(max) = 2 in this case. The maximum speed up achieved was 1.40, in simulation experiment sim11.86.

The relation between the duration of the simulation experiment and real-time is calculated from:

\[
\text{speed-up over real time} = \frac{\text{time which is simulated}}{\text{actual execution time}}
\]

For the fastest experiment, sim11.86, the speed up over real time is 332 (i.e. the simulator ran 332 times faster than real time).
The efficiency of the multiprocessing arrangement is a measure of how effective the multiprocessing is, and is given by:

\[\text{let the: benchmark execution time} = Tb\]
\[\text{actual execution time of the experiment} = Ta\]
\[\text{number of processors} = n\]

\[\text{efficiency} = \frac{Tb - Ta}{Tb - \left( \frac{Tb}{n} \right)}\]

\(Tb/n\) is the theoretical minimum execution time that is expected. In this case (number of processors = 2) it is half of the benchmark time.

The efficiency is 0\% if the actual execution time of the experiment is the same as that of the benchmark (\(Tb = Ta\)); this would mean that no contribution has been made by the multiprocessing arrangement. The efficiency is 100\% if the actual execution time is the theoretical maximum.

The results given in table 6 show that the increase in speed due to the multi-processing arrangements of the modules is constant whether animation is included or not, hence proving that the overhead is reduced if the generation of animation data is not included.

The highest efficiency, of approximately 57\%, is obtained with experiment sim11.86, when the decision module operates in parallel with the other modules. This can be compared with the well-known rule-of-thumb /NEWM85/ that the contribution of additional processing units to a multi-processor computer shows a diminishing rate of
return: let the time taken to execute a program on a multi-processor computer be $T_a$, the comparable time taken on a uniprocessor computer $T_b$, and the number of processor $N$:

$$T_a = \frac{T_b}{N \times (0.8^{N-1})}$$

($N - 1$ is the number of additional processors)

The usual efficiency according to this rule-of-thumb for a two processor system would be approximately 75%.

The other influences on the speed of execution in a multiprocessor system apart from the arrangement of the modules are the clock speeds of the individual processors; the number of wait states that each processor uses when communicating with common memory; (these remained constant during these experiments); and the priority of access to common memory for each processor. In these experiments, the priority of access to common memory was fixed by the position of the particular single-board computer on the Multibus, and all the software was located in common memory. The influence of this can be seen by comparing the results of experiments sim40.86 with sim00.86, or sim41.86 with sim01.86. In these experiments, all the modules were executed sequentially, but in experiments sim00.86 and sim01.86 they were executed on the processor with least priority, and in the case of experiments sim40.86 and sim41.86 they were executed on the processor with a higher priority.

The results show that the greatest increase in speed occurs when the decision making activity/module is allocated to its own processor so that the other modules simulating the actions of AGV’s, machines and
buffers are executed in parallel with the decision making activity. This module (the decision making module) is also the most influential factor throughout all of the different arrangements of modules to be executed in parallel, and hence there is no advantage to be gained from executing any of the modules sequentially with the decision module.

11.6 The influence of the synchronisation code.

A comparison of the control code of the multi-processing simulation system and a comparable sequential simulation system without synchronisation code is shown in figure 40. The actual time required to execute the instructions that comprise the synchronisation code (i.e. to 'increment the synchronisation counter', 'wait until processors are synchronised' (assuming there is no actual wait) and 're-initialise the synchronisation counter' is approximately 73 microseconds per simulation cycle, on an 8086 processor running at a clock speed of 8 MHz (this was the processor and clock speed used on the SBC 86/30 master single board computer). The total time required to execute this code in 800 cycles of simulation is approximately 58 milliseconds, which has no significance when related to a program execution time of the order of 216.6 seconds (sim00, for example).

11.7 Discussion of Results.

In order to discuss the results of the work which has been described in this thesis, it is necessary to examine the question of whether a simulator using techniques of distributed simulation satisfies the
the simulation cycle consists of the following tasks:
- execute the simulation procedures
- output cycle confirmation signal
- increment cycle counter
- increment synchronisation counter
- wait until processors are synchronised
- execute executive cycle
- re-initialise synchronisation counter

Comparison between multi-processor and serial simulation

Figure 40
criteria developed for a simulator of highly automated batch manufacturing systems at the detail design stage. These criteria have been described in previous chapters and will be briefly re-stated here, followed by a discussion of whether the results show that the distributed simulator satisfies the criteria.

11.7.1 A Review of the Criteria.

The criteria for a distributed simulator of highly automated batch manufacturing systems have been described based on three different views of the simulation system: firstly, from the point of view of a simulator of manufacturing systems at the detail design stage (described in chapter 4); secondly, from the point of view of distributed simulation (described in chapter 5); and thirdly, from the point of view of a multiprocessing system (previously described in chapter 6).

The criteria which have been described for a simulator at the detail design stage of manufacturing systems are:

- the level of detail;
- the ease of building a model and the ease of use;
- speed of execution;
- facilities for validation and verification.
- the level of generality;
- the expressiveness of the simulation language;
- numerical behaviour;
- structural features;
- status of implementation;
The criteria which have been described for a distributed simulator are:

- the way in which sub-division of the problem is achieved;
- the efficiency of simulation (related to events list processing);
- reduction in development time; and
- real-time operation.

The criteria which have been described for a multi-processing system are:

- speed of execution;
- reliability;
- availability and response; and
- cost modularity and flexibility.

Some of these criteria for a simulation system looking from one point of view are related to criteria from a different point of view. The criterion of 'level of detail' is important both from the point of view of a simulator at the detail design stage and from the point of view of a distributed simulator where sub-division of the problem has taken place. This criterion is also related to 'the level of generality' of a simulation system, and the 'expressiveness of the
simulation language which in this case is related to which activities and entities can easily be represented by the simulator.

In the same way, the criterion of the 'ease of building a model and using a model' is important from the same points of view, but additionally from the point of view of distributed simulation where a reduction in development time due to distribution should be achieved. This criterion is also related to the criteria of what structural features for simulation are included, what facilities for model verification and validation are provided, and the cost modularity and flexibility of the system.

The criterion of 'speed of execution' is relevant from the point of view of a simulator at the detail design stage, from the point of view of the efficiency of a distributed simulator and whether real-time operation can be achieved, and from the point of view of the efficiency of a multiprocessing system.

11.7.2 Analysis.

The distributed simulator described in this thesis satisfies the majority of the criteria described above to some degree.

Firstly, with respect to the criteria related to the 'level of detail': the results show that the level of detail contained within the simulation model is sufficient for the analysis of operating and control strategies at the detail design stage. The sub-division of the simulation program into modules resulted in achieving a high level of detail within each of the modules, while each module also remained of amanageably small size for compiling, testing, debugging and
executing. The simulator was, however, only general in relation to the class of manufacturing system for which it was designed, viz. a highly automated batch manufacturing system.

Secondly, with respect to the criteria related to the 'ease of building a model and its ease-of-use': the results show that configuring the model to simulate a particular manufacturing system through data facilitates the use of an interactive configuration program and hence makes it relatively easy for the user to configure a model. A reduction in development time for a particular model is achieved because of this ability. The structural features incorporated in the simulator which increase the ease with which a model can be built, are the sub-division of the model into modules related to the activities of specific types of entities, so that changes in such activities are confined to one module only. The results relating to the criterion of facilities for model verification and validation are reported in chapter 11.4. The simulator satisfies this criterion to some degree regarding facilities for statistical data output and to a high degree by providing facilities for graphical animation.

The ease-of-use of the prototype distributed simulator was not developed to the point where the user can operate the simulator in a 'friendly' way. However, the criterion of cost modularity and flexibility of the simulation system was satisfied by the results described in chapter 11.4.2, in that the arrangement of modules to be processed in parallel could be easily changed in a straight-forward way to take advantage of one or more computer processors.
Thirdly, with respect to the criteria of speed-of-execution: the results show that the distribution of the simulation into modules that could be processed in parallel instead of sequentially, resulted in an increase in the speed of execution of the simulation, particularly when the decision-making module (as implemented in this simulator) was executed in parallel with the other modules. The efficiency of the simulation itself was not optimised—in particular, the simulation executive used a unit time advance mechanism which can increase the number of time steps and tests carried out by the simulation executive. This however, replaced a list-processing algorithm needed to process the list of events due to happen.

The maximum efficiency of multi-processing which was achieved in the experiments was 57%, when the decision module was executed in parallel with all the other modules. This result is more than two-thirds of the efficiency that is usually expected of a two-processor multi-processor system, and more than half of the theoretical maximum, and it can be concluded, in respect to multi-processing efficiency, that the parallelism inherent in this type of simulation model (particularly with regard to the operational decision-making process) can be taken advantage of, in order to increase the computational speed of the simulation.

With respect to the criterion of 'real-time operation', the simulator was not configured for a real-time environment but for a design environment. However, the advantages of the multi-processing simulator, shown by the results of the experiments on the simulator, are also applicable in a real-time environment where the result of using a range of possible operating strategies must be simulated.
before a decision is made.

Fourth, with respect to the criteria of availability and response: the response is given by the duration of a simulation run (in the range of two to three minutes, for the case study used in the experiments), and is fast enough to be an interactive response for this type of work, compared to the batch-processing response (tens of minutes to hours) which is commonly expected for simulations of this type. This response time should be taken in the context provided by the number of simulation activities which were started during this time (2439 activities were started for the case study described in chapter 10). With respect to the criterion of availability, the results show that it is possible within the distributed simulator to allocate the task or module which required the greatest processing power to its own processor and hence achieve the benefits of the parallel execution of that module (the increase in speed).

The remaining criteria are not necessarily applicable to a distributed simulator built for the experimental purposes described in this thesis, i.e. the criterion of 'portability of the system' cannot be satisfied because the simulation system is dedicated for use on the Intel multi-processor computer, the criterion of 'numerical behaviour' is not directly applicable to the case study used for the experiments; similarly, the 'reliability' of the multi-processor system was not studied here.

The status of implementation of the simulator has been described in chapters 8 and 9, and additional documentation about the simulator is contained within the programs and in the following references:/ROBE85/,/ROBE87/.
12.1 Conclusions.

The examination of a simulation system for highly automated batch manufacturing systems which uses distributed computing and multiprocessing techniques has shown that a number of advantages can be achieved through the use of these techniques for simulation during the detail design phase of this class of manufacturing system.

The sub-division of the simulation program into modules based on the activities of particular classes of entities, viz. part storage and load/unload buffers, machines, automatic guided vehicles and operational level decision-making, has enabled a sufficiently high level of detail to be included in the simulation model for the type of analyses required during the detail design phase, particularly the analysis of operating and control strategies.

Collecting the activities of a particular class of entity together into one module has enabled the configuration, through data, of the precise selection of activities that a particular entity is involved in, and hence enabled the complete configuration of the simulator system to a particular simulation model by the use of data. For the
same reason, modifications to the activities of an entity can also be confined to the particular module which represents that entity.

The distribution of the simulation model into modules and the use of interface records to represent the changes in state of the entities in the modules, together with a simulation executive cycle which is only required to test for changes in the interface records, has enabled the simulation modules to be processed simultaneously in parallel instead of in a sequential fashion.

Units of parallelism such as the simulation modules require software coordination and hence a multiple-instruction, multiple-data computing system was found to be necessary in order to process the modules in parallel. It was shown that a multi-processing computer system, based on an industry-standard bus (Intel's Multibus I) and tightly-coupled single-board computers using a master-slave operating system organisation was an appropriate computational vehicle.

Measuring the duration of the execution time of the simulation for a particular case study has shown that when the simulation modules are distributed over two computer processes and executed simultaneously in parallel, the time required to run the simulation can be shortened, at the most, by a factor of 1.4, equivalent to a multiprocessing efficiency of 57%. The greatest influence on the run-time was shown to be the processing time required for operational level decision-making.

12.2 Recommendations for further work.

Further work is required in order to optimise the performance of the
multiprocessing simulator described in this thesis. Most attention should be given to the performance of the decision-making module, because of the length of time required to process this module. It should be possible to find algorithms which are more efficient to calculate which pallet should be released next.

Additional efficiency should be possible by incorporating a 'self-checking' mechanism into each of the simulation modules in order to reduce the number of checks carried out by the simulation executive. This would distribute the checks, which are at present carried out in a centralised, sequential way, and introduce parallelism into the simulation executive. The self-checking mechanism would require only one variable to be tested by the simulation executive: the variable would show whether anything had changed during the last time step or not. Such a mechanism would reduce the minimum number of tests to be carried out by the executive to be equal to the number of modules.

A simulator which is to be used at the detail design stage of a highly automated batch manufacturing system for the analysis of operating and control strategies, requires an internal model of the decision-making strategies which will be included in the control system, and the simulation executive must be a close copy of the control system which is to be used. The distributed simulator enables the separation of these elements in order that they may be built in a modular way into the simulator. However, it is important that the control system and decision-making algorithms (and their counterparts in the simulator) are clearly defined within the total hierarchy of decision-making in a manufacturing system.
The simulator described in this thesis would not execute sufficiently quickly in its present form to simulate the results of a range of possible operational strategies and provide results, on the basis of which one of the possible strategies could be chosen, in so-called 'real time' (the time required for operation at the level of dynamic system control). The execution speed of a simulator is related to the cost of the computer which is used, and the limiting factor in this type of application is normally the cost. The speed-up obtained due to multi-processing shows that it may be possible to use a cost-effective multi-processor computer in this application to achieve the high performance necessary.
References.

/ANST86/ : Anstiss, P.
'A new concept in the control of manufacturing systems.'
Proceedings 5th Int Conf on Flexible Manufacturing Systems,
Stratford, UK 1986
IFS Ltd. Nov 1986 pp 201-214

/ANTO86/ : Antonelli, C.J., Volz, R.A., Mudge, T.N.
'Hierarchical decomposition and simulation of manufacturing cells
using ADA.'
Simulation Vol 46 No 4 April 1986 pp 141-152

/ARDO84/ : Ardo, A., Philipson, L.
'Implementation of a PASCAL-based parallel language for a
multiprocessor computer.'
Software-Practice and Experience Vol 14 No 7 1984 pp 643-657

/BANK85/ : Banks, J., Carson, J.S.
'Process-interaction simulation languages.'
Simulation Vol 44 No 5 May 1985 pp 225-235

'The optimal planning of computerised manufacturing systems.'
School of Industrial Engineering Report No 1 1975
Purdue University, Michigan USA 1975

/BARE85/ : Barel, M.
'Performance Evaluation of the Discrete Event Simulation Computer
DESC.'
Proceedings 18th Annual Simulation Symposium
IEEE, USA 1985 pp 1-14

/BELL82/ : Bell, R., Bilalis, N.G.
'Loading and Control Strategies for an FMS for Rotational Parts.'
Proceedings 1st Int Conf on Flexible Manufacturing Systems,
Brighton, UK 1982
IFS Ltd. 1982 pp 77-87

/BELL83/ : Bell, R., Roberts, E.A.
Private Communication.

/BELL86/ : Bell, R. et al.
'A system for the design and evaluation of highly automated batch
manufacturing systems.'
Proceedings Conf on UK Research in Advanced Manufacture, Dec 1986
Institution of Mechanical Engineers, London 1986

/BERR85/ : Berry, O., Jefferson, D.
'Critical Path Analysis of Distributed Simulation.' in
Distributed Simulation 1985 Ed. Reynolds, P. Vol 15 No 2 pp 57-60
Society for Computer Simulation, USA 1985

/BILA83c/ : Bilalis, N.G.
The Design and Control of FMS for Rotational Parts.
Ph.D. Thesis Loughborough University of Technology, UK. 1983

/BILA83/ : ibid, Ch 2

/BILA83a/ : ibid, p28

/BILA83b/ : ibid, p81
BLEC85: Blech, R.A., Arpasi, D.J.
'Hardware for a real-time multiprocessor simulator.' in
Distributed Simulation 1985 Ed. Reynolds, P. Vol 15 No 2 pp 43-49
Society for Computer Simulation, USA 1985

BONN85: Bonney, M.C. et al.
'Evaluation and use of a graphical robot simulator.'
Proceedings 1st Int Conf on Simulation in Manufacturing,
Stratford, UK 1985
IFS Ltd. 1985 pp 325-329

BOWE80: Bowen, B.A., Buhr, R.J.A.
The Logical Design of multiple-microprocessor systems.
Prentice Hall, 1980, Ch 8

Bowe80a: ibid, p67

BRE85: Brennan, L., Davies, B.J.
'The planning of manufacturing change in a period of contraction.'

BROW85: Browne, J. et al.
'An integrated FMS design procedure.'

BRUN82: Bruno, G., Canuto, E.
'Simulation of Production systems with DESFOR.'
Proceedings MTDR Conf No 23 1982 pp 203-216

BRYA81: Bryant, R.
SIMPAS Users Manual.
Academic Computing Centre, University of Wisconsin-Madison, USA
Technical Report No 54 1981

CADD83: None
'Off-line programming to create optimised production line robot systems.'
CAD/CAM Digest Vol 5 No 2 Oct 1983 p5

CAPU83: Caputo, F., Micheletti, G.F.
'Some problems in design of flexible Manufacturing Systems.'
Annals of the CIRP Vol 32 No 1 1983 pp 417-421

CARR85: Carrie, A.
'FMS simulation: needs, experience, facilities.'
Proceedings 1st Int Conf on Simulation in Manufacturing,
Stratford, UK 1985
IFS Ltd. 1985 pp 205-215

CARR85a: Carrie, A.S., Perera, D.T.S.
Private Communication.
(Paper entitled 'Work Scheduling in FMS', presented at the Conference
of the Operational Research Society, Durham UK Sept 1985)

CATA86: 'Catalog of simulation software.'
Simulation Vol 47 No 4 Oct 1986 pp 152-165

CELL83: Cellier, F.E.
'Simulation software: today and tomorrow.'
Simulation in Engineering Sciences, North-Holland 1983 pp 3-19

CELL85: Cellier, F.E., Rimwall, M.
'Distributed Modelling and database management in simulation.' in
Distributed Simulation 1985 Ed. Reynolds, P. Vol 15 No 2 pp 21-24
Society for Computer Simulation, USA 1985

- 215 -
/CHANB3/ : Chandak, A., Browne, J.C., 'Vectorization of discrete event simulation.' Proceedings 1983 Int Conf on Parallel Processing Aug 1983 pp 359-361 (also see /REED85/)


/CHARB4a/ : ibid., Ch 2

/CHARB4b/ : ibid, Ch 2 p77

/CHARB4c/ : ibid, Ch 2 p81

/CHARB4d/ : ibid, Ch 2 p110

/CHARB4e/ : ibid, Ch 2 pp 113-114

/CITR84/ : Mast FMS Simulation - Preliminary Information. Citroen Industrie UK 1984 (also see /LENZB5/)


/CLEMB5a/ : ibid, p9

/CONC85/ : Concepcion, A.I., 'Mapping distributed simulators onto the hierarchical multi-bus microprocessor architecture.' in Distributed Simulation 1985 Ed. Reynolds, P. Vol 15 No 2 pp 8-13 Society for Computer Simulation, USA 1985

/DAHM82/ : Dahmen, N., 'Modelling and Simulation with FORECASD.' Phillips GmbH Forschungslaboratorium Laborbericht No 528/82, Hamburg 1982


/FIEL85/: Fielland, G., Rodgers, D.  
'Processor Pooling.'  
Systems International May 1985 pp 68-70

/FORS85/: Forsyth, R.  
'Side by Side.'  
Personal Computer World Vol 3 No 12 December 1985 p166

/FORS85a/: ibid, p166

/GHOS85/: Ghosh, J.B.  
'Asynchronous Control of discrete event simulation.'  
Proceedings 18th Annual Simulation Symposium  
Institute of Electrical Engineers 1985 pp 255-264

/GRAI83/: Graiser, H.N.  
'CAD/Robotics Capability helps realise CIM.'  
Industrial Engineering Vol 15 No 8 August 1983 pp 26-30

/GRIT85/: Grit, D.H., McGraw, J.R.  
'Programming divide and conquer for a MIMD machine.'  
Software-Practice and Experience Vol 15 No 1 Jan 1985 pp 41-58

/HADD85/: Haddock, J., Davis, R.P.  
'Building a simulation generator for manufacturing cell design and  
control.'  
Proceedings 1985 Conf of the Institute of Industrial Engineers  
1985 pp 237-244 (copy obtained direct from author)

/HANN85/: Hannam, R.G.  
'Alternatives in the design of flexible manufacturing systems for  
prismatic parts.'  

/HARL85/: Harland, D.M.  
'Towards a language for concurrent processing.'  
Software-Practice and Experience Vol 15 No 9 1985 pp 839-888

/HART84/: Hartley, J.  
FMS at Work.  
IFS Publications Ltd. Bedford, UK 1984 Ch 17

/HENR83/: Henriksen, J.D., Crain, R.C.  
GPSS/H Users Manual.  
Wolverine Software Corporation, Va. USA 1983

/HITO79/: Hitomi, K.  
Manufacturing Systems Engineering.  
Taylor and Francis Ltd. London, 1979 p3

/HOOP86/: Hooper, J.W.  
'Strategy-related characteristics of discrete-event languages and models.'  
Simulation Vol 46 No 4 April 1986 pp 153-159

/INSI00/: Insight International Ltd.  
Optik 11 - Publicity Material  
Insight International Ltd. Woodstock, UK.

/INTE82/: Intel Corp.  
Introduction to the INTEL RMX86 Operating System.  
Intel Corporation publication No 9803124 Swindon, UK 1982

/ISTE00/: Istel Ltd.  
'See-Why' and 'EXPRESS' Publicity Material  
Istel Ltd., Redditch, UK.
IROE82/ : Roehdar, W. et al.  
'A Flexible Multiprocessor System for Simulation Purposes.' in  
Discrete Simulation and related fields. Ed. Javor, A.  
North-Holland 1982 pp 29-56

IROE85/ : Roberts, E.A., Shires, N.  
'The Application of Multiprocessing to Simulation.'  
Proceedings 1st Int Conf on Simulation in Manufacturing,  
Stratford, UK 1985  
IFS Ltd. 1985 pp 313-324

IROE87/ : Roberts, E.A., Shires, N.  
Parallel processing applied to the simulation of Highly Automated  
Batch Manufacturing Systems.'  
Paper presented at the 3rd Int Conf on Advances in Production  

RUSS00/ : Russell, E.C.  
Building Simulation Models with SIMSCRIPT II.5.  
CACI, La Jolle, Cal. USA.

SCHA85/ : Schanehchi, J.  
'EXPRESS: A man-machine interface for simulation.'  
Proceedings 1st Int Conf on Simulation in Manufacturing,  
Stratford, UK 1985  
IFS Ltd. 1985 pp 97-104

SILC85/ : Silcock, B.  
'Superchip to the rescue.'  
The Sunday Times 29 September 1985, p80

SING85/ : Singh, M.G., Allidina, A.Y.  
'Low cost parallel computation for optimisation and simulation.'  
Chartered Mechanical Engineer Vol 32 No 2 Feb 1985 pp 24-25

SOLB81/ : Solberg, J.J.  
'Mathematical Tools for the Design and Operation of Manufacturing  
Systems.' in  
Manufacturing Solutions based on Engineering Sciences.  
American Society of Mechanical Engineers 1981

SOLB82/ : Solberg, J.J.  
'Mathematical Design tools for Integrated Production Systems.' in  
Plenum Press 1982, pp 175-197

STAN86/ : Stanridge, C.R.  
'Animating simulations using TESS.'  
Computers and Industrial Engineering Vol 10 No 2 Sep 1984 pp 121-134

STEI84/ : Steinhilper, R.  
'The right part families and manufacturing tasks for FMS.'  
Proceedings 3rd Int Conf on Flexible Manufacturing Systems,  
Boeblingen FGR 1984  
IFS Ltd, Sep 1984 pp 153-163

SURI84/ : Suri, R., Whitney, C.K.  
'Decision Support Requirements in Flexible Manufacturing.'  
Journal of Manufacturing Systems Vol 3 No 1 1984 pp 61-69

SURI84a/ : Suri, R., Hildebrant, R.R.  
'Modeling Flexible Manufacturing Systems using Mean-Value Analysis.'  
MIT Industrial Liaison report No. 1-41-84  
Massachusetts Institute of Technology 1984

-220-
The data flow approach for MIMD multiprocessor systems. in Parallel Processing Systems. Cambridge University Press 1982 pp 239-273

Tocher, K.D. The Art of Simulation. English Universities Press 1963 Ch 12


'Normalair-Garrett decides on more FMS.' The FMS Magazine Vol 1 No 1 Oct 1982 pp 20-23

'Advanced computer-aided Engineering and Manufacturing.' Proceedings Institution of Mechanical Engrs Vol 197B 1983 pp 81-89

'Some aspects of the efficient use of multiprocessor control systems.' Ph.D. Thesis, Loughborough University of Technology, UK 1981

ibid, Ch 3

ibid, pp 15-18

ibid, Ch 9

ibid, pp 4-5


'Physical Simulation: the use of scaled-down fully functional components to analyze and design automated production systems.' Computers and Industrial Engineering Vol 8 No 1 1984 pp 73-85
APPENDIX I

DATA REQUIREMENTS.

I.1 Data Requirements

The data used to configure and drive the simulation modules of the distributed simulator falls into several different groups. These groups are:

- data relating to the description of the transport network (i.e., the automatic guided vehicle paths) and the control of the AGV's along those paths;
- data describing the numbers used in the simulation output file when generating data suitable for animation purposes;
- data used by the decision module;
- data used for machining purposes (such as part operation times, tool magazine capacities etc.); and
- data describing the storage buffers in the system (buffer capacities, speeds etc.)

There is also a data file for each of the modules in the system which
is the interface data in its initial state.

I.2 Transport Network Data

This data consists of four files, called 'agv'; 'rf'; 'pf'; and 'rq'.

The file 'agv' is one of the two interface data files used by the module which simulates the operation of the agvs's in the system. The fields that it contains on a particular line are:

- the number of the agv;
- its current state;
- its current scheduled time (to next event);
- its current route;
- the current or last object or station which it passed or is stationed at;
- the object or station towards which it is travelling;
- its current ultimate destination during this particular move;
- its priority;
- the pallet which it is carrying;
- an index number; and
- a number which, if greater than zero, is its home station.

The file 'rf' is a route description file. For each route in the system there are three fields. These are: 'from' (the station number which the route goes from); 'to' (the station number that the route goes to); and the length of the route. This length is in
terms of the number of animation increments of the particular route, and the number of animation increments of a route is in turn based upon a step length used to provide smooth animation on the basis of the scale of a particular drawing or layout.

The file 'pf' is a proximity data file. For each route, the number of routes that the agv would have to travel along to reach a station if it set off along a particular route is given, for each possible station and route.

The file 'rq' is a table showing requests made by each station for agv service. This is the second interface data file used by the module which simulates the operation of the agvs in the system. For each station, there are five fields:

- a request field;
- a priority field;
- a field indicating which decision type is to be used for the request;
- a field which describes which agv is servicing the station; and
- a field which shows the number of the pallet involved.

1.3 Machines and Process Data

This data consists of four files, called 'pd'; 'td'; 'md'; and 'pld'.

The file 'pd' is the parts data file and for each part number (or part type) it shows the number of operations that part requires; and
for each of the operations, it shows the setting time for the operation, a list of tools used during the operation together with the cutting or operation time of each of the tools (each usage of a tool during an operation is called a sub-operation).

The file 'td' is a tool data file. For each machine in the simulated system, the tool magazine capacity is given. Then for each pocket in the tool magazine, the number of the tool which occupies that pocket together with its start-up tool life and the remaining tool life of that tool are given.

The file 'md' is the interface data file between the module which simulates the operation of the machines and the control module (or simulation executive). The fields which it contains for each machine are:

- the machine name;
- its station number;
- its state;
- the time remaining (to next event);
- saved state and time (in case of breakdown);
- the current part number, pallet number, number of parts completed and the number of parts to be completed;
- the position of the tool magazine, the current tool being used and its remaining tool life;
- the current operation of the part being processed and its total number of operations;
- the current sub-operation (tool operation) of the part being processed and its total number of
sub-operations on the particular machine; and
- the part and operation that the machine is currently
set up for.

The file 'pld' is data relating to the pallets on which parts are
fixed. For each pallet, the capacity of the pallet is given (in
terms of the number of positions at which parts can be fixtured); and for each of these positions, the quantity of that part type, the
number of the operation to be carried out on that part, and the number
of the sub-operation at which the part should begin the operation is
given.

I.4 Buffer Storage Data

This data consists of three files, viz. 'bf'; 'tf'; and 'bd'.
Included as buffers in the simulated system are load/unload mechanisms
at machines (i.e. anywhere where pallets can be effectively stored -
the machine table is such a position).

The file 'bf' is the buffer and pallet data file. For each buffer or
load/unload device, the storage capacity of the device is given
together with its type and speed of operation. For each possible
storage position on the buffer, a record is kept of the number of the
pallet stored at that position, its current state (the number of the
next station to which it is intended to visit), and a number to show
whether the pallet has been 'allocated' or not.

The file 'tf' is a file which carries a record of interim positions
reached by a buffer: where a rotary buffer may, for example, rotate
one half a position when the next unit step in time takes place. The

- 227 -
increment in position is stored in the temporary file together with the actual integer move in positions requested and the integer quantity actually reached.

The file 'bd' is the interface file for the buffer simulation module and, for each buffer, shows:

- its name and identification number;
- its current state;
- time to next event;
- saved state and time in the event of breakdown;
- the position at which a machine is connected to the buffer;
- the pallet at that position;
- the position at which it can load and unload pallets and the pallet at that position;
- the pallet number that the buffer may have been requested to load or unload;
- and the status of that pallet (its next intended station for processing).

1.5 Decision Data

The data from which decisions are made regarding which pallet to load or unload from a storage device consists of four files: 'bdf'; 'dd'; 'prf' and 'cf'.

The file 'bdf' is the buffer and pallet data file which is similar to the file 'bf' in the buffer storage data. It contains, for each storage device in the system, the capacity of that device, and for
each storage position up to the capacity of the device, the number of the pallet stored at that position, its status (next intended station) and an allocation data variable.

The file 'dd' is the interface data file for the decision making module of the simulator: for each storage device where pallets are stored and therefore decisions can be made about whether to release the pallet, or not, or which pallet to release, there are the following data fields:

- the identification of the station;
- its i.d. number;
- a state field to show whether a decision has been requested or made;
- the result of that decision in terms of a pallet number and its next intended station;
- a number to indicate the type of decision required;
- a column to show whether a particular storage buffer or station should allocate a pallet on which a decision has been made (so that the same pallet is not included in further decisions);
- a pallet number and pallet status of a pallet which should be deleted from the decision data file ('bdf'); and
- the number and status of any pallet requiring unloading from the storage buffer or station.

The file 'prf' is a pallet route file which, for each pallet in the system, gives the number of operations to be carried out on the pallet.
( or number of stations to be visited by the pallet ); the priority of the pallet; a time value which is the time between the completion of all operations on the pallet and when new raw material is loaded on the pallet in place of the finished machined components ( i.e. the time between completion of the operations required for the pallet and when it becomes 'live' again ). For each operation on the pallet there is also a station number, which is the station number to be visited at each stage or operation.

The file 'cf' is a configuration file. For each storage device in the system, the number of storage spaces which must be kept free from being filled with pallets waiting to be machined ( the 'free space' ) is given. This free space is intended for use by completed pallets being unloaded from the machine which the storage buffer serves. It is possible that a storage buffer serves more than one machine and, for this reason, for possible machine positions up to a maximum of five and similarly for load/unload positions up to a maximum of five, the number of actual machines and load/unload positions connected to the buffer is given, and then for each of these positions the number of the station at that position at the buffer is given. For example, a loop conveyor type buffer which serves one machine will have one machine position; the station number of that machine could be station 11; and that station or machine could be at position 8 of the buffer. It may be connected to the automatic guided vehicle network, in which case, if it is connected at only one point, it will have one load/unload position; the number of the station at that load/unload position will be 0 ( zero ) ( connected to the AGV network ); and the agv network connection could be at position 1 ( one ) of the buffer.
I.6 Other Data Requirements

The simulation executive module requires one file: 'sf'. For each station in the system (buffer storage device or machine), a number is given which is the number of the buffer storage device or the machine number (for each type). The last known state of each station is also given together with its current load and index number.

Other data is used for the purposes of generating simulation output data. There are five files which are used for this purpose: 'bbd'; 'stdat'; 'tdata'; 'tcon'; and 'ant'.

The file 'bbd' is a record for each station which stores pallets or loads/unloads them (i.e. buffers), and in each record the following data is contained: the last state of the device, the pallet that was requested from the device and the status of that pallet, the last pallet requested (if the current pallet requested is zero), and a number relating to the last route of that pallet.

The file 'stdat' keeps a record of data used for the simulation output file which is, for each pallet, its current location and the last route on which it travelled; and for each automatic guided vehicle (AGV), four parameters, which are the state of the AGV; the number of the pallet which it is carrying; the route on which the AGV is on; and the last station or 'object' which the AGV was at or is at.

The file 'tdata' is the simulation output file and in its initial state consists of two header lines and any simulation output data that has gone before (at the start of a fresh simulation there is no previous simulation output data).
The file 'tcon' is a configuration file for the generation of simulation output data. For each buffer (load/unload or storage device) the file gives:

- the column that movements of the buffer are recorded in in the simulation output file;
- a route number which takes a pallet from the buffer to a machine to which it may be attached (or which it may serve);
- the route number which takes a pallet from an AGV docking station on to the buffer;
- the route which takes a pallet from a machine back to the buffer;
- the route which takes a pallet from the buffer on to an AGV;
- the number of increments that the routes to and from the machine are;
- the number of increments that the route to and from the buffers are;
- and for pallets which must rotate around the buffer the numbers of routes which the pallets must move on are given together with the number of increments of those routes; and finally,
- the routes which AGV's and pallets must take when moving along a linear pallet storage buffer are given.

The file 'ant' is a simulation configuration file for the generation of simulation output data and it records the number of lines required at the maximum in the simulation output file; the column offset used
when recording AGV movements in the output file; the time advance per increment of simulation; the 'route' number required to record that machining is taking place; and parameters relating to the speed of the AGV's in terms of increments upon routes.
The complete sequence of interactive configuration of the simulator is shown in figure 41.

The first step, that of drawing the proposed layout of the manufacturing system to scale on graph paper is carried out, and subsequently the machine, buffer, station and agv station numbers are assigned to the layout. The way in which this is done has already been described in chapter 10, and for the particular example of a four machine FMS used in this appendix, is shown in figure 42, figure 43 and figure 44. The use of the interactive configuration programs is related to this layout.

The next two steps are those of drawing the static parts of the layout using a CAD system and creating an IGES file of the layout. The fourth step is to use the program 'genroute' to define AGV routes and pallet movements and an example of this is given in figure 45. The final task of the interactive program 'genroute' is to print out the route data which has been input and create the appropriate files. The route data file is also shown in this figure.
1. Draw the proposed layout of the manufacturing system to scale on graph paper
2. Draw a picture of the static parts of the layout using a CAD system
3. Create an IGES file from the CAD system
4. Define the AGV routes and pallet movements using the program 'genroute'
5. Convert the IGES layout file to the correct format using program 'CGIS'
6. Define the initial AGV and pallet positions in the system using program 'gentrot00'
7. List the file 'log06.dat' and define the segments to be animated using the program 'gen09'
8. Define additional information about the buffers in the system using program 'gentcon'
9. Create the proximity data using program 'genpf'
10. Configure the simulation using programs 'config1' and 'config2'
11. Run simulator

Configuration of the simulator

Figure 41
The next two steps of converting the IGES file to the correct format using the program 'cgis', and defining the initial AGV, pallet positions and animated entities using the program 'genfort06' are shown as examples in figure 46. The listing of the file 'log06.dat' is also included in this figure.

The next step is to define the segments to be animated using the program 'gen09' and the use of this program is shown in figure 47. The data that must be input while using the interactive program 'gentcon' in order to define additional information about the buffers in the system is shown in figure 48. The final interactive step of creating the proximity data file is shown in figure 49.

The following step, of using the programs 'config1' and 'config2' is not interactive and hence examples are not included in this appendix.
Case study layout for configuration example

Figure 42

Route numbering for configuration example

Figure 43
Graph paper layout of configuration example
Figure 45: Interactive use of program 'genroute'.

```pascal
genroute
[Sheffield Pascal version 3.0.23]
No errors reported.

Executing ROUTECONFIG

Do you want to create a new file or append/change an existing one?
Answer 1 for a new file or 2 for an existing one... 1

Configuring route no. 1

Is route 1 a track or a number of lines...?
answer 1 for a track or 2 for lines... 1
From which station does this route start... 9
To which station does this route go ....... 10
Is this route the opposite of a previously defined route (y/n)...? n
How many arcs and straight lines make up route 1 ? 1

Input X,Y start points for section 1 ?
seperated by a space... 200 950
Is this section an arc or circle ? answer y or n... n
Input X,Y end points for section 1 ?
seperated by a space... 400 950
this will be divided into 13 steps
Another route... ? answer y or n.... y

Is route 2 a track or a number of lines...?
answer 1 for a track or 2 for lines... 1
From which station does this route start... 10
To which station does this route go ....... 9
Is this route the opposite of a previously defined route (y/n)...? y
Which route then..... ? 1
Another route... ? answer y or n.... y

Is route 3 a track or a number of lines...?
answer 1 for a track or 2 for lines... 1
From which station does this route start... 10
To which station does this route go ....... 12
Is this route the opposite of a previously defined route (y/n)...? n
How many arcs and straight lines make up route 3 ? 1

Input X,Y start points for section 1 ?
seperated by a space... 400 950
Is this section an arc or circle ? answer y or n... n
Input X,Y end points for section 1 ?
seperated by a space... 400 1080
this will be divided into 9 steps
Another route... ? answer y or n.... y

Is route 4 a track or a number of lines...?
answer 1 for a track or 2 for lines... 1
```

- 239 -
Figure 45 (continued): Interactive use of program 'genroute'.

OK, pascal genroute
(Sheffield Pascal version 3.0.2)
No errors reported.

Executing ROUTECONFIG

Do you want to create a new file or append/change an existing one?

Answer 1 for a new file or 2 for an existing one... 2
Attempting to read existing data from file am07.dat
Reading route data from am77.dat...
Reading data from the route file...
There are already 46 routes.

Configuring route no. 47

Is route 47 a track or a number of lines...
answer 1 for a track or 2 for lines... 2
For lines: at what segment do the lines begin.. ? 7
How many lines are there in this route.. ? b
Another route.. ? answer y or n.... y

Is route 48 a track or a number of lines...
answer 1 for a track or 2 for lines... 2
For lines: at what segment do the lines begin.. ? 13
How many lines are there in this route.. ? b
Another route.. ? answer y or n.... y

Is route 49 a track or a number of lines...
answer 1 for a track or 2 for lines... 2
For lines: at what segment do the lines begin.. ? 19
How many lines are there in this route.. ? b
Another route.. ? answer y or n.... y

Is route 50 a track or a number of lines...
answer 1 for a track or 2 for lines... 2
For lines: at what segment do the lines begin.. ? 25
How many lines are there in this route.. ? b
Another route.. ? answer y or n.... n

Attempting to write routedata to am77.dat:
writing data to am77.dat

<table>
<thead>
<tr>
<th>Route</th>
<th>From</th>
<th>To</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>10</td>
<td>13.0</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>9</td>
<td>13.0</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>12</td>
<td>9.0</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>10</td>
<td>9.0</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>11</td>
<td>13.0</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>12</td>
<td>13.0</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>13</td>
<td>8.0</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>12</td>
<td>8.0</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>5</td>
<td>6.0</td>
</tr>
</tbody>
</table>
Figure 45 (continued): Interactive use of program 'genroute'.

```
10  5  13  6.0
11  13  14  8.0
12  14  13  8.0
13  14  6  8.0
14  6  14  8.0
15  14  15  8.0
16  15  14  8.0
17  15  7  6.0
18  7  15  6.0
19  15  16  8.0
20  16  15  8.0
21  16  9  8.0
22  9  16  8.0
23  16  17  57.0
24  17  16  57.0
25  17  1  17.0
26  1  17  17.0
27  1  2  27.0
28  2  1  27.0
29  2  18  17.0
30  18  2  17.0
31  10  19  57.0
32  10  18  57.0
33  17  3  64.0
34  3  17  64.0
35  3  4  27.0
36  4  3  27.0
37  4  18  64.0
38  18  4  64.0
39  0  0  6.0
40  0  0  6.0
41  0  0  6.0
42  0  0  6.0
43  0  0  6.0
44  0  0  6.0
45  0  0  6.0
46  0  0  6.0
47  0  0  6.0
48  0  0  6.0
49  0  0  6.0
50  0  0  6.0
OK, coma -e
```
Figure 46: Interactive use of program 'cgis', 'genfort06' and listing of 'fort06.dat'.

OK, pascalg cgis
[Sheffield Pascal version 3.0.2]
No errors reported.

Executing CRIMGIS

IGES FILE PRODUCED BY DOGS  
DO you want to continue...? y

OK, pascalg genfort06
[Sheffield Pascal version 3.0.2]
No errors reported.

Executing TEST!

Do you wish to start a new layout...? answer y/n y
Do you wish to add a time segment...? answer y/n y
Do you wish to add a pallet, rotary buffer lines or agv...? answer y/n y
Add a pallet : rotary buffer lines : agv
answer 1 : 2 : 3
Your answer.... 2

/ * antici10ck *
/ 
( ----- )
/ 1 / 
\ v / clockwise \
/ 
--------

Input no. of lines... 6
Input start angle (degrees) .... 90
Input end angle (degrees) .... 270
Input x offset (pixels) .... 970 1450
Input y offset (pixels) .... 1450
Input radius .... 90
Are crosslines required...? answer y/n n
Clockwise rotation or antici10ckwise...?
Figure 46 (continued): Interactive use of program 'cgis', 'genfort06' and listing of 'fort06.dat'.

Input Y coordinate of pallet: 1410
Is this pallet visible?: y
More?: y
Add a pallet: y
Answer y/n: y
Input number of pallet: 4
Input sidelength of pallet: 60
Input X coordinate of pallet: 170
Input Y coordinate of pallet: 1520
Is this pallet visible?: y
More?: n
Answer y/n: n

OK,

spool log0b.dat
CSPool rev 19.3.33
PRT002 spooled, records: 1, name: LOG0B.DAT
OK, if log06.dat
rotation lines of buffer 1 start at segment 7
rotation lines of buffer 2 start at segment 13
rotation lines of buffer 3 start at segment 19
rotation lines of buffer 4 start at segment 25
AGV 1 is segment number 31
AGV 2 is segment number 32
Pallet number 1 is segment number 33
Pallet number 2 is segment number 34
Pallet number 3 is segment number 35
Pallet number 4 is segment number 36
OK, comme -e
Figure 47: Interactive use of program 'gen09'.

OK, pascalg gen09
(CSheffield Pascal version 3.0.2)
No errors reported.

Executing GENERATE_AM09

Program to configure files am09.dat & ant ..............
Do you wish to configure file am09? (y/n) ... y
Do you wish to configure file ant? (y/n) .... y
How many moving pallets are there to animate? .. (min. of 1) 4
Which segment is pallet 1 ?.... 33
Which segment is pallet 2 ?.... 34
Which segment is pallet 3 ?.... 35
Which segment is pallet 4 ?.... 36
How many agvs are there to animate? .. (min. of 1) 2
Which segment is avg 1 ?.... 31
Which segment is avg 2 ?.... 32
How many moving buffers are there ?..4
How long is the shortest avg route [see file rt for details] ? 6
How many increments is this route in total? .... 6
How much time does it take an avg to travel this distance? ... 1
How many lines of animation data are there likely to be? ... 700
Is time display segment no. 6? [see file for106.dat] (y/n) y
What is the time advance per cycle of simulation? .... 0.5
Is iflash = 90? (y/n) y
OK, con0 -e

- 244 -
Figure 48: Interactive use of program 'gentcon'.

OK, pascalgentcon
[Sheffield Pascal version 3.0.2]
No errors reported.

Executing GENERATE_TCON

Program to generate Tcon....................

How many buffers are there in the system?.... 0
For Buffer no. 1 input the following data
Input column number in animation data ..... ( 0 if no animation )
( column number = no. of pallets + no. of agvs + 1 + this buffer no. )
Column no. is .... 8
Input route no. to rotate buffer clockwise ( 0 if no rotation ) 47
Input route no. to rotate buffer antclcclock ( 0 if no rotation ) 0
Input route no. to take pallet to machine ( 0 if no route ) 44
Input route no. to take pallet to buffer ( 0 if no route ) 39
Input route no. to take pallet from machine ( 0 if no route ) 43
Input route no. to take pallet from buffer ( 0 if no route ) 42
Input time to move pallet to/from machine ( 0 if no route ) 6
Input time to move pallet to/from buffer ( 0 if no route ) 6
Input maximum clockwise route that a pallet on the buffer takes 41
Input maximum anti-clock route that a pallet on the buffer takes 0
Input minimum clockwise route that a pallet on the buffer takes 40
Input minimum anti-clock route that a pallet on the buffer takes 0
Input time for pallet to rotate one position round the buffer 5
Input route to take agv 'up' a linear buffer 0
Input route to take agv 'down' a linear buffer 0
For Buffer no. 2 input the following data
Input column number in animation data ..... ( 0 if no animation )
( column number = no. of pallets + no. of agvs + 1 + this buffer no. )
Column no. is .... 9
Input route no. to rotate buffer clockwise ( 0 if no rotation ) 48
Input route no. to rotate buffer antclcclock ( 0 if no rotation ) 0
Input route no. to take pallet to machine ( 0 if no route ) 44
Input route no. to take pallet to buffer ( 0 if no route ) 39
Input route no. to take pallet from machine ( 0 if no route ) 43
Input route no. to take pallet from buffer ( 0 if no route ) 42
Input time to move pallet to/from machine ( 0 if no route ) 6
Input time to move pallet to/from buffer ( 0 if no route ) 6
Input maximum clockwise route that a pallet on the buffer takes 41
Input maximum anti-clock route that a pallet on the buffer takes 0
Input minimum clockwise route that a pallet on the buffer takes 40
Input minimum anti-clock route that a pallet on the buffer takes 0
Input time for pallet to rotate one position round the buffer 6
Input route to take agv 'up' a linear buffer 0
Input route to take agv 'down' a linear buffer 0
For Buffer no. 3 input the following data
Input column number in animation data ..... ( 0 if no animation )
( column number = no. of pallets + no. of agvs + 1 + this buffer no. )
Column no. is .... 10
Input route no. to rotate buffer clockwise ( 0 if no rotation ) 49
Input route no. to rotate buffer antclcclock ( 0 if no rotation ) 0
Input route no. to take pallet to machine ( 0 if no route ) 44
Input route no. to take pallet to buffer ( 0 if no route ) 39
OK, pascal genpf
(Sheffield Pascal version 3.0.2)
No errors reported.

Executing TESTPF

Debugging required... y/n ? n

III! NOTE III! Set constant agvstations equal to actual number of agvstations....

reading data from the route file...

<table>
<thead>
<tr>
<th>Proximity Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$rin-$</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>33</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>36</td>
</tr>
<tr>
<td>37</td>
</tr>
<tr>
<td>38</td>
</tr>
<tr>
<td>39</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>41</td>
</tr>
<tr>
<td>42</td>
</tr>
</tbody>
</table>