THE PROGRAM MONITOR: A PROGRAMMABLE INSTRUMENT FOR
COMPUTER HARDWARE AND SOFTWARE PERFORMANCE MEASUREMENT

by

EDWARD CHARLES GIAIMO, III

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREES OF
BACHELOR OF SCIENCE
and
MASTER OF SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1975

Signature of Author. . . . . . . .
Department of Electrical Engineering and Computer Science
May 9, 1975

Certified by . . . . . . . . . . . . . Thesis Supervisor (Academic)

Certified by . . . . . . . . . . . . . Thesis Supervisor (W/A Co-operating Company)

Accepted by . . . . . . . . . . . . . Chairman, Departmental Committee on Graduate Students
THE PROGRAM MONITOR: A PROGRAMMABLE INSTRUMENT FOR
COMPUTER HARDWARE AND SOFTWARE PERFORMANCE MEASUREMENT

by

Edward Charles Giaimo, III

Submitted to the Department of Electrical Engineering
on May 9, 1975 in partial fulfillment of
the requirements for the Degrees of Bachelor of Science
and Master of Science in Electrical Engineering.

ABSTRACT

Designers and users of complex digital systems, especially computer
systems, have the sometimes difficult and often time-consuming tasks to
debug and optimize these systems. Their work involves reading and under-
standing circuit schematics or computer programs to predict the system's
behavior under various operating conditions, and to know how to modify
its operation as desired. Observation rather than prediction of dynamic
system action (measurement) gives a better picture of the system without
the approximations, estimations, or speculations that often go with
prediction. Unfortunately, such observations are difficult to obtain
since desired measurement tools are unavailable, especially for software
measurement.

This thesis describes a real-time programmable hardware monitor
instrument to sample digital circuit signals, search for desired patterns,
and collect data based upon the comparisons. Connection is made from the
monitor, that is programmed by a Hewlett-Packard 2000 series minicomputer,
to a host system via measurement probes with which host status signals
are sensed. The types of information that can be obtained via this
technique are described; they include computer system performance, soft-
ware functionality, software usage and timings, and digital logic opera-
tions. Automated measurement applications are obvious possibilities for
such a programmable instrument. Because of its general applicability to
measure software programs as well as digital circuits, the name "program
monitor" was chosen. Examples of software to program and operate the
program monitor are presented for typical measurement applications.

Thesis Supervisor (Academic): H. D. Toong
Title: Assistant Professor of Electrical Engineering and Computer Science
Thesis Supervisor (VI-A Cooperating Company): J. S. Peachey
Title: Division Product Assurance Manager, Hewlett-Packard Data Systems
Division
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>2</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>3</td>
</tr>
<tr>
<td>List of Figures</td>
<td>6</td>
</tr>
<tr>
<td>List of Tables</td>
<td>8</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>9</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>10</td>
</tr>
<tr>
<td>2 Background of Digital Systems Measurement</td>
<td>13</td>
</tr>
<tr>
<td>Digital Systems Performance</td>
<td>13</td>
</tr>
<tr>
<td>Computer Synectics &quot;SUM&quot;</td>
<td>19</td>
</tr>
<tr>
<td>An Outline of the Program Monitor</td>
<td>22</td>
</tr>
<tr>
<td>Digital System Measuring Instruments</td>
<td>25</td>
</tr>
<tr>
<td>3 Digital Machine Models</td>
<td>28</td>
</tr>
<tr>
<td>Discrete System Model</td>
<td>28</td>
</tr>
<tr>
<td>Asynchronous Sequential Machine Model</td>
<td>31</td>
</tr>
<tr>
<td>Clocked Sequential Machine Model</td>
<td>33</td>
</tr>
<tr>
<td>State Table and State Diagram Descriptions</td>
<td>36</td>
</tr>
<tr>
<td>Measurements Obtainable from Sequential Systems</td>
<td>40</td>
</tr>
<tr>
<td>Fundamental Program Monitor Circuit Model</td>
<td>45</td>
</tr>
<tr>
<td>4 Circuit Description of the Program Monitor</td>
<td>49</td>
</tr>
<tr>
<td>Overview</td>
<td>49</td>
</tr>
<tr>
<td>Circuit Summary</td>
<td>52</td>
</tr>
<tr>
<td>Input Processor</td>
<td>55</td>
</tr>
<tr>
<td>Comparators</td>
<td>58</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Event Collection</td>
<td>60</td>
</tr>
<tr>
<td>Event Initialization</td>
<td>63</td>
</tr>
<tr>
<td>Interrupts</td>
<td>64</td>
</tr>
<tr>
<td>Programmable Time Base</td>
<td>65</td>
</tr>
<tr>
<td>Programming Command Decode</td>
<td>66</td>
</tr>
<tr>
<td>PM-bus Arbitration</td>
<td>69</td>
</tr>
<tr>
<td>I/O Bus Data Interface</td>
<td>70</td>
</tr>
<tr>
<td>I/O Bus Interrupt Interface</td>
<td>70</td>
</tr>
<tr>
<td>I/O Bus Control Interface</td>
<td>71</td>
</tr>
<tr>
<td>Circuit Specifications and Limitations</td>
<td>72</td>
</tr>
<tr>
<td>5 Software Interfacing the Program Monitor</td>
<td>75</td>
</tr>
<tr>
<td>Overview</td>
<td>75</td>
</tr>
<tr>
<td>Clear Command</td>
<td>78</td>
</tr>
<tr>
<td>Load Data Channel Command</td>
<td>78</td>
</tr>
<tr>
<td>Load Time Code Command</td>
<td>79</td>
</tr>
<tr>
<td>Load Input Processor Command</td>
<td>80</td>
</tr>
<tr>
<td>Read Select Command</td>
<td>81</td>
</tr>
<tr>
<td>Load Data Modes Command</td>
<td>82</td>
</tr>
<tr>
<td>Stop Sampling Command</td>
<td>83</td>
</tr>
<tr>
<td>Start Sampling Command</td>
<td>84</td>
</tr>
<tr>
<td>Reading Data</td>
<td>84</td>
</tr>
<tr>
<td>Set Control</td>
<td>85</td>
</tr>
<tr>
<td>Clear Control</td>
<td>86</td>
</tr>
<tr>
<td>Set Flag</td>
<td>86</td>
</tr>
</tbody>
</table>
6 Verification Tests and Applications

Purpose of the Tests

Single CPU Breakpoint

Sampling a 21MX Computer

Real-Time Program Activity Display

Computer System Performance

Computer Software Debugging

Microcode Performance

Digital Circuit Measurement

Measuring Mechanical Apparatuses

Summary

7 Suggestions for Future Work

Circuit Design Cleanup

Circuit Simplification

Higher Performance

More Capability

Appendix

A Program Monitor Circuit Schematics and Timing Diagrams

B Software for Program Monitor Testing

C Program Activity Display Software

Bibliography
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>H-P 200Gx/200 Program Time Measurement by SUM</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>Program Monitor, Minicomputer, Host System Configuration</td>
<td>23</td>
</tr>
<tr>
<td>3.1</td>
<td>Combinational System Model</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>General Asynchronous Sequential Circuit Model</td>
<td>32</td>
</tr>
<tr>
<td>3.3</td>
<td>General Clocked Sequential Circuit Model</td>
<td>34</td>
</tr>
<tr>
<td>3.4</td>
<td>State Table for 1's Counter</td>
<td>38</td>
</tr>
<tr>
<td>3.5</td>
<td>State Diagram for 1's Counter</td>
<td>39</td>
</tr>
<tr>
<td>3.6</td>
<td>Fundamental Diagram of the Program Monitor</td>
<td>46</td>
</tr>
<tr>
<td>4.1</td>
<td>Program Monitor Block Diagram, Data Blocks</td>
<td>50</td>
</tr>
<tr>
<td>5.1</td>
<td>I/O Instructions</td>
<td>76</td>
</tr>
<tr>
<td>5.2</td>
<td>Programming Commands Summary</td>
<td>77</td>
</tr>
<tr>
<td>5.3</td>
<td>Programming Command Word Formats</td>
<td>91</td>
</tr>
<tr>
<td>5.4</td>
<td>Initial Circuit State</td>
<td>93</td>
</tr>
<tr>
<td>5.5</td>
<td>Time Base Divisor Coërtes</td>
<td>94</td>
</tr>
<tr>
<td>5.6</td>
<td>Data Obtained by an I/O Read</td>
<td>95</td>
</tr>
<tr>
<td>6.1</td>
<td>H-P 2100A Self-Measurement Configuration</td>
<td>98</td>
</tr>
<tr>
<td>6.2</td>
<td>H-P 21MX Self-Measurement Configuration</td>
<td>99</td>
</tr>
<tr>
<td>6.3</td>
<td>Measuring an H-P 21MX Computer</td>
<td>101</td>
</tr>
<tr>
<td>6.4</td>
<td>H-P 2000F/200 Controlled Stimulus/Response Measurement System</td>
<td>104</td>
</tr>
<tr>
<td>6.5</td>
<td>Affinity Photographs of H-P 2000F/200 IOP</td>
<td>108</td>
</tr>
<tr>
<td>7.1</td>
<td>Modified Channel Counter Control Circuit and Timing Diagram</td>
<td>116</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.2</td>
<td>Asynchronous Front-End Circuit</td>
<td>119</td>
</tr>
<tr>
<td>A.1</td>
<td>Input Processor Data Circuit</td>
<td>122</td>
</tr>
<tr>
<td>A.2</td>
<td>Input Processor Control Circuit</td>
<td>123</td>
</tr>
<tr>
<td>A.3</td>
<td>Comparator Circuit</td>
<td>124</td>
</tr>
<tr>
<td>A.4</td>
<td>Event Collection Circuit</td>
<td>125</td>
</tr>
<tr>
<td>A.5</td>
<td>Interrupt Control Circuit</td>
<td>126</td>
</tr>
<tr>
<td>A.6</td>
<td>Event Initialization Circuit</td>
<td>127</td>
</tr>
<tr>
<td>A.7</td>
<td>Programmable Time Base Circuit</td>
<td>128</td>
</tr>
<tr>
<td>A.8</td>
<td>Programming Command Decode Circuit</td>
<td>129</td>
</tr>
<tr>
<td>A.9</td>
<td>I/O Bus Data Interface Circuit</td>
<td>130</td>
</tr>
<tr>
<td>A.10</td>
<td>I/O Bus Interrupt Interface Circuit</td>
<td>131</td>
</tr>
<tr>
<td>A.11</td>
<td>I/O Bus Control Interface Circuit</td>
<td>132</td>
</tr>
<tr>
<td>A.12</td>
<td>Control Input Timing Diagram</td>
<td>133</td>
</tr>
<tr>
<td>A.13</td>
<td>Channel Cycle Timing Diagram</td>
<td>134</td>
</tr>
<tr>
<td>A.14</td>
<td>2100A, 21MX I/O Timing Diagram</td>
<td>135</td>
</tr>
<tr>
<td>A.15</td>
<td>I/O Command Timing Diagram</td>
<td>136</td>
</tr>
<tr>
<td>B.1</td>
<td>CPU Breakpoint Test Program</td>
<td>138</td>
</tr>
<tr>
<td>B.2</td>
<td>2100A CPU Measurement Test Program</td>
<td>140</td>
</tr>
<tr>
<td>B.3</td>
<td>21MX CPU Measurement Test Program</td>
<td>141</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.1</td>
<td>Hardware Monitor History</td>
<td>17</td>
</tr>
<tr>
<td>2.2</td>
<td>Hewlett-Packard Digital Measuring Products</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>Intercard Cable (ICC) Signal Assignments</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Logic Equations for 21MX CPU Register Strobing</td>
<td>57</td>
</tr>
<tr>
<td>4.3</td>
<td>Truth Table for Event Encoder</td>
<td>61</td>
</tr>
<tr>
<td>4.4</td>
<td>Program Monitor Circuit Specifications</td>
<td>73</td>
</tr>
<tr>
<td>6.1</td>
<td>2000F/200 Affinity, Program Monitor vs. Synectics SUM</td>
<td>109</td>
</tr>
<tr>
<td>7.1</td>
<td>Modified Sampling Cycle Times</td>
<td>117</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

The author wishes to express his thanks and appreciation to many people who assisted in the creation of this thesis. First of all, thanks go to the VI-A cooperating company, Hewlett-Packard and its employees; to John Figueroa for technical and managerial guidance, to Jim Peachey for creating the open atmosphere conducive to creative engineering, to John Hawkes for running TEPE and the 2000F/200 during measurements, to Phil Gordon for assistance in interfacing the 21MX, and to Dave Crockett for support of the program monitor development. Many friends in California, especially Peter and Marti Ashkin, John Cooper, Karen Blue, and Marcy Rothermal, gave much personal support during the thesis work. Thanks go to Hoo-min Toong, whose technical advice brought together the theoretical and practical aspects of the thesis. The typist, Mrs. Albee was incredibly helpful in the precise details of putting together this final thesis form. Sherry Grobstein deserves special thanks for her love and support that gave this author encouragement. Very special consideration goes to John Tucker, who aside from making the VI-A program available as its director, has given personal interest and understanding for many years.
CHAPTER 1
INTRODUCTION

Despite today's wealth of technical knowledge, the ability to accurately measure and characterize digital systems lags behind the rapid pace to design new systems. Designers in circuitry and programming apply the latest techniques to build computing machines that far exceed their fore-runners in speed, capability, and complexity, yet attitudes are subdued when considering digital systems measurement probably due to marketing uncertainty of such instruments. Users have begun to create a concern for better systems performance which is slow to reach the designers.

With billions of dollars in installed computer equipment deployed worldwide and with vast sums needed to operate, program, and maintain this hardware, computer users are increasingly aware of the need to improve the efficiency of data processing operations.\(^{(1)}\)

Companies such as Allied Computer Technology, Boole and Babbage, Computer Synectics, and Comress have marketed software and hardware tools to assist users in doing performance evaluations. Few of these tools are utilized by the designers to evaluate their own designs, which is unfortunate since, "...the evaluation problem is closely and intimately related to the design problem..."\(^{(2)}\) That is, a fundamental understanding of today's systems and reasons for their performance characteristics can encourage efficient new designs. This understanding could be gained through measurement of existing systems, and the measurement propelled by the development of tools that would be easy to use by engineers and programmers. Flexibility without
sacrificing operational ease is important to individuals' utilization of a
given instrument.

Consider for a moment a person who is deciding among a number of
methods to satisfy a set of project goals. Basically, he evaluates the
cost/benefit ratios of the alternatives and chooses the optimum one. For
example, suppose a designer is deciding between two alternatives that each
would make an existing design operate faster by the same amount. The
choices might be, A) redesign everything with new technology devices and
speed up the entire design, or B) measure the old design and improve only
those parts that are bottlenecks to efficiency. Logically speaking, the
designer would choose A if the cost to totally redesign appeared less than
the cost to measure and partially redesign. Most decisions are not as
simple as in this contrived example, but history has shown that given the
choice, designers opt not to measure.

Designers could create a new design philosophy for themselves through
easily-used flexible measurement tools for digital systems. These tools
could open the possibility for a concurrent "design and measure" philosophy,
one that is already part of the analog systems field. Analog system
designers already have many complex and simply-operated instruments such
as digital voltmeters, phase meters, and oscilloscopes, while the number of
tools for digital systems is few. Designers and users could gain a much
better understanding of system operation through measurement. Redesign and
evaluation could become less of an art and more of a science, as well-
developed test plans and information gathering would result.
It is the intent of this thesis to present a new instrument and its implementation as a research tool for the measurement of digital systems for engineers and programmers. The instrument is less than very simple to use, but its capabilities and design philosophy can point the way to future compact tools for digital systems measurement.
CHAPTER 2

BACKGROUND OF DIGITAL SYSTEMS MEASUREMENT

Digital Systems Performance

The computer performance measurement and analysis profession has been developed to tackle problems related to the efficient use of computer systems. Computer performance work has dealt with the detailed study of a particular class of digital systems, namely programmed computers. These studies constitute the most complete treatise to date of an area of digital systems. Many techniques have been used in studying computer systems and can be divided into four main areas: simulation, analytic modeling, software instrumentation, and hardware instrumentation. A paper that provides a good introduction to the terminology of computer system measurement\(^{(3)}\) defines these terms in the following way:

Simulation

This is a technique for building and operating a discrete state model [finite state machine] of a system. A model of the system, often simplified, is used because tests on the real system are too expensive, the real system is not yet built, or proposed changes to a real system need to be tested before implementation. ...Programming languages have been designed for building and operating simulation models, and there are special purpose simulation systems for determining the performance of proposed computer configurations.

It is difficult to construct an accurate model of the experimental system since accurate parameter values are often unknown. Hence, data derived from simulations can at best serve only as predictors of true performance. Generally, simulation techniques give sufficiently accurate
results to be used in evaluating tradeoffs among proposed design topologies. Performance estimates can be obtained without prototype construction, a distinct advantage over sensing methods. The languages SIMSCRIPT AND GPSS, for example, are well-known simulation languages. Evaluation nets, an outgrowth of Petri nets, have been used to simulate multiprogrammed computer systems such as Control Data Corporation's CDC6400 computing system. (4)

Analytic Modeling

A portion of the system can often be studied by representing it as a mathematical model and then analytically solving the model for the parameters of interest. Queuing and communication problems have been successfully analyzed with this technique.

This method can give good results that satisfy theoreticians developing an understanding of the operation of a process. Unfortunately, most complex real-world processes cannot be analytically modeled in a simple manageable fashion. The equations become difficult to express in a manner solvable by machine, or if solvable, the answers cost too much machine time or take too long to find.

Software Instrumentation

This technique...consists of an instrumentation and a recording component. The instrumentation component is a software modification to the [programmed] system that detects events of interest and collects relevant information about the events. The recording component is a program that moves collected data into [mass storage such as magnetic tape or disc]. A reduction program operating offline is required to analyze the recorded data.

Using software instrumentation, one can observe all parts of the software system and environment. The normal flow of programmed procedures is
intercepted at some point to obtain the required information. Clearly, a price is paid in stolen processing time and memory space from the pre-programmed system for the measuring program. A danger exists if the presence of the measuring software alters the parameter(s) being measured, causing artifacts in the measurements. Therefore, much caution is used when applying ad hoc software measurement tools. In contrast to ad hoc methods, IBM's CALL/360 is an example of successful software measurement. Artifacts were avoided by the design of measuring tools as part of the software system during the basic design phases. Much valuable data was collected that allowed optimization of system performance.

Hardware Instrumentation

The mechanism for this is a special hardware monitor, connected to the computer, that senses signal changes in the computer's circuitry. Some of these signal changes represent events of interest and are recorded by [the] monitor. The monitor can count the frequency of these events and the time between specified events.

The information collected by the monitor can be recorded for subsequent computer-assisted data reduction or viewed for immediate reduction. The results of these measurements can subsequently be graphically presented in useful forms for interpretation.\(^5\) "The primary advantage of a hardware monitor is that the measurements are taken externally to the system operating environment, placing no added load on the operating system, hardware configuration, or user programs. This provides a true measurement of system performance."\(^6\) Just as an engineer uses a high impedance voltmeter to measure circuit voltages without significant effect upon the voltage being measured, it is imperative that non-invasive techniques be
used to measure predesigned systems to avoid the generation of artifacts. Hardware monitors are generally non-invasive, unlike software instruments, and can give measurements without adverse effects upon the system under measurement.

All portions of the hardware system are available for probing, whereas software sections are not always accessible. Conclusions concerning software performance are derived from circuit signals within the system which reflect the operation of software sections. For example, the program counter (also referred to as the instruction address register) is a localized group of signals that specifies the location of the currently executing instruction. Sampling this group of bits at regular intervals of time can provide a profile of how frequently parts of a program are used. Such a graph is an invaluable feedback mechanism to programmers in the process of debugging or optimizing a program.

Combining software and hardware measuring techniques is often advantageous in giving hardware sensors access to remote software sections. In particular, software events can be transformed by software into hardware events that are subsequently measured by circuitry probing. This kind of measuring technique has been used on the M.I.T. Multics time-sharing system, to give measures of system utilization. Such a measurement system is tailored to a specific application and difficult to generalize to a wide class of measurement problems.

A variety of hardware instruments for computer monitoring has evolved during the past decade as shown in Table 2.1, an update from C. D. Warner. The development of advanced measurement systems controlled by dedicated
<table>
<thead>
<tr>
<th>Year</th>
<th>Manufacturer</th>
<th>Equipment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1961</td>
<td>IBM</td>
<td>Channel Analyzer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I/O channel use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPU wait time</td>
</tr>
<tr>
<td>1962</td>
<td>IBM</td>
<td>Program Monitor&lt;sup&gt;(8)&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Control unit and tape drive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Measures software</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Long offline data reduction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Designed for IBM 7090 computers</td>
</tr>
<tr>
<td>1964</td>
<td>IBM</td>
<td>Program Event Counter (PEC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Measures CPU &amp; channel overlap</td>
</tr>
<tr>
<td>1967</td>
<td>IBM</td>
<td>System Analysis Measuring Instrument (SAMI)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Measures all system resources</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 counters, 32 comparators</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Weight 4 tons</td>
</tr>
<tr>
<td>1968</td>
<td>IBM</td>
<td>Basic Counter Unit (2829 BCU)&lt;sup&gt;(9)&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Punched-card output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1MHz sampling rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 counters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>System/360 compatible</td>
</tr>
<tr>
<td>1969</td>
<td>IBM</td>
<td>System Logic and Usage Recorder&lt;sup&gt;(10)&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Associative memory construction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Programmable operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dynamic counter assignment</td>
</tr>
<tr>
<td>1969</td>
<td>Computer Synectics</td>
<td>System Utilization Monitor (SUM)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Portable, magnetic tape output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional comparators</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 MHz counting rate</td>
</tr>
<tr>
<td>1970</td>
<td>Allied Computer Technology</td>
<td>CPM-X&lt;sup&gt;(11)&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Built around mini processor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5MHz counting rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multi-function parallel/serial counter groups</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Real-time data reduction to plotter</td>
</tr>
</tbody>
</table>

Table 2.1 Hardware Monitor History (continued)
<table>
<thead>
<tr>
<th>Year</th>
<th>Manufacturer</th>
<th>Equipment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>Comress</td>
<td>Dynaprobe series</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Portable with magnetic tape output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional printer and teletype</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hardware counters to minicomputer</td>
</tr>
<tr>
<td>1972</td>
<td>Tesdata Systems</td>
<td>X-RAY (12)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Programmable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32 counters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.5MHz counting rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Magnetic tape output</td>
</tr>
<tr>
<td>1974</td>
<td>Computer Performance</td>
<td>Digital-Sampling Processor (DSP-1)</td>
</tr>
<tr>
<td></td>
<td>Instrumentation</td>
<td>Programmable by command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Real-time graphic display</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optional chart recorder output</td>
</tr>
</tbody>
</table>

Table 2.1 Hardware Monitor History
miniprocessors is evident in the more recent products. Each of these systems is relatively expensive due to high development costs and low user volumes. Users must acquire a good deal of familiarity with the systems before producing meaningful results, hence, most manufacturers have training seminars and field engineers to assist users with equipment operation. High investments both initially and over the long term are required to make measurements with the advanced equipment. "The cooperation of the computer system personnel and the monitor support personnel can normally provide the necessary mixture of knowledge for useful system evaluation." (6)

**Computer Synectics "SUM"**

The System Utilization Monitor (SUM), manufactured by Computer Synectics, is representative of the advanced monitoring equipment listed in Table 2.1. Its capabilities were found useful to Hewlett-Packard in measuring the performance of their Time-Shared Basic 2000F I/O processor (IOP) software system. In particular, measurements made of clock time spent in various program areas of the IOP using the non-invasive techniques possible with the SUM equipment, greatly aided pinpointing some fundamental bottlenecks in the 2000F software.

To understand how the measurements were taken, refer to the block diagram in Figure 2.1. The 2000F consists of two H-P 2000 series minicomputers, one acting as control and computing, and the second as I/O and communications. Measurement priority was placed on directly user-visible parameters, hence the IOP, which communicates with user terminals, was chosen for measurement. Seventeen hardware sensor cables, the only
2000F Dual-CPU System

SUM* Equipment

Main CPU
HP 2100A

I/O Processor
HP 2100A

Program Counter

17 Sensor Cables

Address Comparison

Comparator

10-digit Counter

1MHz Pulse Generator

Magnetic Tape Recorder

Disc Storage

1-16 User Terminals

* Tradename Synectics Corp.

Figure 2.1
H-P 2000F/200 Program Time Measurement by SUM*
connection between the 2000P and the SUM, provided the measurement path. Sixteen of the cables carried the sixteen data bits being sampled, while the signal on the seventeenth cable (strobe) indicated the instant at which to sample the data. The cables were clipped to the program counter (P-register) of the IOP which gave the address of each instruction to be fetched and executed from the IOP's main memory.

Within the SUM, data from the P-register entered through some comparison circuitry that compared the data against two sets of toggle switches. If the data value was bounded by the two switch settings, then the comparator output was asserted. As long as the comparator registered a comparison, time was accumulated in a counter in the SUM. The counter therefore corresponded to the total time that the running program spent between the memory address values set on the SUM comparator's toggle switches. By dividing the time accumulated on the counter by the total sample time, time utilization percentage was derived for the program section under study. The toggle switch values were subsequently altered to sample different program sections. A profile of time utilization for selected program sections of the IOP was thusly produced and bottleneck regions of high utilization percentages were identified.

By probing alternative areas of circuitry, other characteristics can be measured with the SUM such as,

1. Instruction type utilization
2. Program section utilization by number of instructions executed
3. Disk sector usage
4. Correlation of disk and program section operations

5. Frequency of selected interrupts from peripheral devices.

It is important to note that a speed limitation of one microsecond per sample is inherent in the SUM's design as in many of the hardware monitors listed in Table 2.1. This constraint generally precludes measurement of individual instruction times, microcode performance, and real-time signalling of detectable events. Many late model monitors designed later than the SUM, provide programmable control circuits that greatly enhance the flexibility of the tool. Speed and software control were the primary motivations for developing a new hardware monitor in this thesis. As design progressed, new applications became apparent and minor circuit modifications were made. The resultant instrument, called the Program Monitor, is outlined in the next section of this chapter and described in detail in Chapter 4.

An Outline of the Program Monitor

The program monitor consists of two Hewlett-Packard 2000 series interface cards that plug into any of the easily accessible I/O slots in a 2000 series minicomputer. The computer provides interrupt service and software interfacing to the cards via the I/O slots. Connection in this manner gives an easy way to program the functions of the program monitor (refer to Figure 2.2). It is simply treated as a peripheral device where communication is implemented via I/O instructions executed by a program in the computer's main memory. Probe connections to the outside world are provided from the exposed edge connector of one of the cards. The circuitry is
thereby completely contained within the computer, eliminating the need for peripheral cabinetry.

Either data recording or real-time data reduction can be performed by the computer program operating the program monitor. Real-time data reduction using main memory or peripheral storage can eliminate the need for a bulky magnetic tape device that is often used on hardware monitors. Typically, a user needs only the computer with storage for results and a method for loading the program monitor software e.g. a paper tape reader, to constitute a complete measurement system.

A great deal of design effort was directed toward high speed of operation to catch occurrences of many hardware events up to five megahertz in rate. This permits measurement of microcoded software, not formerly possible with other monitors. A built-in programmable crystal-controlled time base provides an accurate reference for performing time measurements. Combination of the high-speed, accurate time base, and sophisticated programmability give the program monitor a balanced set of capabilities. A few measurement applications follow:

1. Software usage - measure the time and frequency of software sections (locality phenomenon)
2. Instruction usage - measure the time to execute and the frequency of use of individual instruction types
3. Hardware operation - sense the operation of digital circuitry (hardware debugging, functionality, and performance)
4. Program trace - trace the flow of execution by instruction or by segment of a program (program debugging)
5. Multiple memory fences - cause an I/O interrupt any time selected areas of protected memory are violated (multiprogrammed software systems)

6. Programmable time base - time base programmably adjustable from 1 microsecond to 1 hour

7. High speed data collection - burst mode data sampling at Direct Memory Access (DMA) rates directly into main memory.

It is important to note that the program monitor was constructed to be used as part of the H-P 2000 computer series, providing hardware/software compatibility with a well-established minicomputer product line. This compatibility does not preclude use of the program monitor in measuring non 2000 series equipment. The H-P computer only provides programming of the cards for the particular measurement application; the type of system to be measured is limited only by the characteristics and number of the sensing probes.

Digital System Measuring Instruments

In addition to advanced instrumentation systems tailored to computer system measurement, smaller and not necessarily less complex instruments have been developed to assist digital logic designers. Hewlett-Packard manufactures a line of digital measurement equipment, some of which is listed in Table 2.2, and described in detail in reference (13). These tools are portable, some are handheld, and most are simple to operate. By applying advanced integrated circuit techniques, H-P has been able to build some sophisticated instruments in handheld form. For example, the
<table>
<thead>
<tr>
<th>Year</th>
<th>Model No.</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1968 | 10525     | Logic Probe Family  
A family of handheld pencil-like probes for digital logic families of DTL, TTL, HLL, MOS, ECL. |
| 1970 | 10528A    | Logic Clip  
A handheld unit that clips over in-circuit integrated circuits with 14 or 16 pins and indicates the pins' logic levels via light emitting diodes. The clip is insensitive to installation polarity. |
|      | 10526     | Logic Pulser Family  
A line of handheld pencil-like signal pulsers that generate a short duration pulse of large amplitude to stimulate an in-circuit component. The logic pulsers in conjunction with the logic probes enable an engineer to derive much information about his system. |
|      | 10529A    | Logic Comparator  
A small instrument that connects to an in-circuit IC and compares its response against another IC of the same type. |
| 1973 | 5000A     | Logic Analyzer  
A small test instrument that can store sequences of bits from inputs upon command. |
| 1974 | 1601L     | Logic State Analyzer  
A scope-like instrument capable of storing and displaying 16 consecutive 12-bit words that are often the state of a digital machine. Pattern matching against the 12-bit word is provided to set the precise time to begin or end storing, a negative time strobe can thus be used. |

Table 2.2 Hewlett-Packard Digital Measuring Products
logic clip has a mechanism for seeking out the circuit ground and voltage supply pins of an integrated circuit (IC) to which it is connected and uses those pins to power itself.

Digital logic designers have found these instruments to be great time-savers in debugging (troubleshooting) their designs, mostly due to the compactness and easy application of the tools to particular circuit problems. The reader should note that the instrument developed in this thesis was implemented with a large number of discrete IC's. This implementation was convenient to demonstrate the feasibility of the instrument. Much practical utility can be gained through miniaturization in a manner similar to other H-P instruments.
CHAPTER 3
DIGITAL MACHINE MODELS

Discrete System Model

At this point, it is useful to characterize digital systems in a general manner and investigate the kinds of measurements obtainable from them. Much as any instrument designer must understand what he intends to measure before he determines how to do the measurement, we will look at what constitutes a digital system before considering how to measure one.

"Broadly speaking, a digital system is any system for the transmission or processing of information in which the information is represented by physical quantities (signals) which are so constrained as to take on discrete values." (16) If the signals can have only two discrete values, the system is binary. Since most digital systems are binary, the term "digital" has come to refer to digital binary (switching) systems and will be so abbreviated in this thesis.

Consider the general model of a digital system shown in Figure 3.1. There are n input (excitation) variables \( x_i(t), i=1,2,...,n \) and m output (response) variables \( z_j(t), j=1,2,...,m \). All variables are assumed to be functions of time - a static system is of little interest. If at any given instant (excluding delays through the logic for settling), the values of the outputs are functions only of the inputs at the same instant, then the system is combinational or memoryless. Such a system can be completely described by a set of equations of the form,

\[
\{ z_j = F_j(x_1, x_2, ..., x_n); j=1,2,...,m \} 
\]
\[ Z_j(t) = f_j[x_1(t), \ldots, x_n(t)]; \quad j = 1, \ldots, m \]

Figure 3.1 Combinational System Model
where the $F_j$'s represent a set of logical (boolean) relations. Time dependence is not explicitly shown since $\{F\}$ is time-invariant and $\{x\}$ and $\{z\}$ are taken at the same instant. Many systems do contain memory (storage) elements, and their outputs are functions of the present inputs and all past inputs within the system's history. Such systems are referred to as sequential.

Regardless of whether we are dealing with combinational or sequential circuits, the input and output values can change with time. The sequences and instants of change of these values carry information, just as the sequence of pulses from a dial telephone conveys the telephone number. Input, output, and internal variables of the system that carry the information are all binary in nature and hence can have precisely one of two possible values at any given instant. In viewing the system as a collection of these variables, ideally, the system can only be in a discrete configuration of variable values at discrete instants. The system will undergo non-continuous changes of state as time progresses. In practice, digital circuits have continuous transition regions during which times, the circuit variable can have an ambiguous value. These regions are relatively short-lived and the probability of finding the system in an ambiguous state is small. The idealization of discrete states is therefore acceptable in practicality. Such discrete-state discrete-transition systems can be completely characterized by recording the sequence of input changes, resulting variable value changes, and the times at which the changes occur. A record of the system's history, much like the frames of a motion picture, can subsequently be used to measure the
characteristics of dynamic system operation. Inferences and conclusions can be gleaned from the recorded frames.

A few issues should be touched on at this point. As previously mentioned, a discrete-transition system is an idealization that can be applied in many practical cases of interest. Even so, this idealization can be removed through additional information about the system, which will be presented in the section about clocked machines. Secondly, there has been no mention of a collectively exhaustive set of variables that describes the system in its entirety. One such conveniently obtained set is presented in the next section of this chapter.

**Asynchronous Sequential Machine Model**

A general model of an asynchronous sequential machine is shown in Figure 3.2. As in a combinational circuit, the outputs, \(\{z\}\), are functions of the present inputs, \(\{x\}\). In contrast to a combinational circuit, the outputs are also functions of the history of the inputs, as represented by a feedback path in the diagram. All of the distributed circuit delay has been lumped into delays in the feedback path for conceptual convenience which may vary from feedback path to feedback path. These delays may be considered to be short-term memory in that they enable the circuit to retain the present values of the state variables, \(\{q\}\), long enough to develop new values of the secondary outputs, that in turn, become the new state values after the delay.

Suppose a given set of input values is applied to the circuit and the circuit operation observed. Typically, the output values and state values will change in a manner determined by the function of the model, until
Figure 3.2
General Asynchronous Sequential Circuit Model
steady-state is reached. It is possible the input change causes either no changes in any circuit variables or triggers the circuit into an unending series of changes. Regardless, the "operation" of the machine can be recorded as a series of output (primary and secondary) signal patterns that occur in response to the change from a given input variable pattern to another. In addition, by recording the time instants at which the output changes happen, we have all the information about how the system responds to the input change of interest. A record of all the input/output patterns and their times of occurrence can enable us to completely recreate the system's operation and understand its function. Indeed, such a record would be extremely lengthy to store and time-consuming to obtain. The major point here is that since the input and output variables can be used to totally describe the system, they also contain all of the information necessary to study any desired aspect of the system related to its function. This statement has great impact, for it implies that by judicious choice of variables, we can measure everything about the digital system model from a single measurement port. This choice can be very difficult, and probably a small set of ports would be easier to find than a single one.

Clock\textsuperscript{ed} Sequential Machine Model

In many practical instances, the general model of the previous section can be further specified. Consider the model in Figure 3.3 which has clocked memory elements added to the feedback path in place of lumped delays. The memory serves to sample the secondary outputs while the clock is asserted, then hold and transfer their values to the state variables when
Figure 3.3
General Clocked Sequential Circuit Model
the clock pulse ends. Any secondary output changes are ignored by the memory between clock pulses. By breaking the feedback loop in a controlled manner, we eliminate the multiple state changes and state cycling via the delay elements that are characteristic of the unclocked system model. The clock period and duration are generally chosen to permit signal settling and memory acquisition. Introduction of the following constraints is also a guarantee that the system can be sampled in such a manner as to find it nontransitory conditions that is a danger in any combinational system:

1) All transitions in the input variables must take place during the interval between clock pulses.

2) The interval between clock pulses must be greater than the larger of the following:
   a) the resolution time of the memory
   b) the time required, measured from [the clock pulse arrival], for any changes in the state variables to propagate through the combinational circuit.

3) The duration of the clock pulses must be sufficient to trigger the memory, but less than the transition time of the memory."(16)

During the operation of the circuit model, a sequence of steps generally occurs in the following manner:

1. The variables have stable values

2. An input pattern change is presented to the combinational logic

3. Signal values propagate through the combinational logic and finally stabilize at the primary and secondary outputs

4. The variables are stable
5. The clock is asserted, thereby reading the secondary output values into the memory

6. The clock pulse ends, causing the state variables to take on the values previously held by the secondary output variables

7. Changes in the state variables subsequently propagate through the combinational logic, generating a new set of secondary output values. (This may occur in conjunction with step 2, eliminating the stable condition of step 1).

The sequence then repeats for subsequent input changes and clock pulses. Importantly, note that all variables are stable during steps 4 and 5. These stable intervals just before clock assertion are convenient times to sample the input/output pattern pair. (Some systems use clock edges to trigger the memory, effectively reducing the time period of step 5 to zero). Only one such sample image per clock pulse is necessary to fully record the system's stimulus-response mechanism as related to its functionality. We note that the volume of recorded information has been greatly reduced from that of the unclocked system. The clock has given us a method to synchronize the measurement operation with the system of concern. Synchronization is also possible with an unclocked system if a pseudo-clock signal can be constituted by the combination of chosen system variables, as shown in chapter 7.

State Table and State Diagram Descriptions

The state tables and state diagrams are convenient tabular and graphical representations, respectively, of the input/output relationships for sequential machines. They describe the complete functionality of any given sequential circuit by listing the values of the output and next-state
variables for all of the combinations of input and present-state variables. From a state table, one may visualize the output sequence that results from a given input sequence and discover the circuit's function. For example, the state table in Figure 3.4 illustrates the operation of a circuit that counts the number of 1's that appears in a sequence of input bits. Up to three 1's are counted before the machine resets to the initial state. The input sequence is presented one bit at a time to the sequential machine that in turn produces a two-bit output that is a binary count of the number of 1's in the input sequence.

The state diagram for the same machine is shown in Figure 3.5. Each circled letter denotes a stable state of the machine corresponding to step 4 in the last section. Arrows connecting the states each correspond to an input pattern that causes the machine to change states, as in steps 5 and 6 of the clocked machine model. Each arrow is thusly marked with input pattern that determines the next state and an output pattern that results from the transition. From each state, there will be an arrow emanating for each possible input pattern. This kind of graphical representation greatly assists the visualization of circuit operation, and shows the dynamics associated with a sequential system. These representations may be used for any sequential machine, whether clocked or not. In representing the non-clocked model, the arrow transitions occur whenever the designated input pattern on the arrow is presented to the system's inputs; whereas, a clocked system is prevented from changing state until a clock pulse occurs. This difference is not explicitly shown on the figures and must be kept in mind when analyzing state tables and state diagrams.
<table>
<thead>
<tr>
<th>$q_1^t$</th>
<th>$q_2^t$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial state 0</td>
<td>0</td>
<td>00/00</td>
<td>01/01</td>
</tr>
<tr>
<td>0 1</td>
<td>(B)</td>
<td>01/01</td>
<td>10/10</td>
</tr>
<tr>
<td>1 0</td>
<td>(C)</td>
<td>10/10</td>
<td>00/11</td>
</tr>
<tr>
<td>1 1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Next state / outputs**

$q_1^{t+1}$ $q_2^{t+1}$ / $z_1$ $z_2$

**Figure 3.4**

State Table for 1's Counter
Figure 3.5
State Diagram for 1's Counter
The usefulness of these two forms is primarily to convey the function of a particular digital system. Information pertaining to particular implementation is lacking, and so therefore is the precise timing relationships among the system variables. In clocked systems, the timing relationships from input to output variables is implicitly understood by knowing the clock configuration used. Unclocked systems, though, have many complex timing interactions causing races and hazards that can drastically alter the system's operation, which is not apparent from the figures shown. It is these timing relationships when used in conjunction with the state table or diagram that carry the system's description. Hence, in measuring a system's operation, it is important to record the observed timing as well as the values of system variables.

Measurements Obtainable from Sequential Systems

Until now, the discussion has focussed around descriptions and models of digital systems. These concepts will presently be extended to include what kinds of measurements and how they may be obtained from these models.

Let's consider the model of a sequential machine and its corresponding state diagram description. Such a system can be characterized by the state diagram that is labelled with all of the system's states, input variables, and output variables and the inclusion of variable timing information. Conceptually this is an easy task and rather orderly; we write down all of the states, and draw connecting arrows that originate at each state, where we will have \(2^n\) arrows per state where \(n=\)number of input variables. Note that certain input patterns are unrealizable while in some states and in these cases, the arrows will end on a null state or any state of
the designer's choice out of convenience. Finally, we label each arrow with an input/output pattern pair. For now we will postpone consideration of the timing information.

Now that we have the system diagrammed, consider the attributes that we can measure from the diagram. Let's create a measurement port from which we view the system's operation which will consist of a set of signal variables, \( \{S\} \), of dimensionality \( d = n + m + r \) such that

\[
S(t) = \{x_1, \ldots, x_n, z_1, \ldots, z_m, q_1, \ldots, q_r\}_t
\]

where there exist \( n \) input variables, \( m \) output variables, and \( r \) state variables with specific values at time \( t \). A particular set \( S \) where we assign a set of values to all of the variables is denoted by a subscript, e.g. \( S_j \) for \( 1 < j < 2^d \). Since the values are all functions of time, we include time dependency which will relate the various \( S_j \)'s. This is a collectively exhaustive set of variables that describes system operation. The values of the variables represent all signals that are needed to recreate the state diagram or state table. The relative timing of these values supplies the additional information to uniquely define the system's operation as seen from the external world. A measurement port containing exclusively the input/output variables can hence be used from the outside world to understand the system.

At a given instant, the machine can be found in a specific configuration \( S_j = S(t_0) \). As the machine operates, the signal values may change and create new sets \( S_k = S(t_0 + 1) \), \( S_k = S(t_0 + 2) \), and so on, where \( \{S(t+i), i=1, n\} \) represents the sequence of states seen at the measurement port. In
operating the machine in a sequential manner, cycles of states may be observed, or certain states may appear more often than others. The frequency of occurrence of a specific set, \( S_k \) relative to the other sets in the sequence is a measure of the utilization of the state \( S_k \). For a sequence of length \( j \) (for \( j-1 \) clock pulses in a clocked system) we can define a term called utilization, for the purpose of this thesis, as

\[
U_{t_k} = \frac{N_k}{j}
\]

where \( 0 \leq U_{t_k} \leq 1, \ 0 < j \).

Utilization measures the proportion of times the system was in a given state during a particular sequence of state changes, regardless of the order or time relationships of the states' occurrences.

An analogous parameter, affinity, quantifies the proportion of time spent in a particular state. In a given sequence of \( j \) states, the machine may spend time in and out of a particular state \( S_k \) as defined by

\[
\text{Aff}_{k} = \frac{T_k}{T_0} = \frac{\text{accumulated time spent in state } S_k}{\text{total time of the measurement of } j \text{ states}}
\]

and \( 0 \leq \text{Aff}_{k} \leq 1, \ 0 < j \).

\( \text{Aff}_{k} \) is also the probability of finding the system in the configuration \( S_k \), if the system is randomly sampled while transitioning the given sequence.

Both of these parameters identify states of high activity within the system. For example, if the measurement port were a computer's program counter, high affinity or utilization for a particular state would correspond to a frequently used program section that would be a candidate for optimization.
A third useful parameter is the average time that a given state exists. For this thesis, we shall name this quantity dwell and define it over a sequence of system states of length \( j \) as,

\[
Dw_k \equiv \frac{T_k}{N_k} = \frac{\text{accumulated time spent in state } S_k}{\text{number of occurrences of state } S_k}
\]

where \( 0 \leq Dw_k \leq T_0 \), \( 0 < j \).

Dwell is a good measure of the efficiency of the system in executing the state \( S_k \). Dwell values larger than the system average of \( T_0/j \) indicate areas of structural or design defects that cause certain states to last longer time periods than others. A simple algebraic equation relates these three measures as follows:

\[
\frac{(Aff_k \times T_0)}{(Ut_k \times j)} = Dw_k.
\]

Four measurements \( T_k, N_k, T_0, \) and \( j \) give all the information necessary to calculate the three basic parameters of system operation.

Interpretation of the parameter values and assigning meaning to them should be carefully considered. Utilization is a characteristic of the structure of the entity that utilizes the system to which our measurement port is attached. It does not measure the system itself, however. We are thusly indirectly deriving information about an entity through another system. In direct contrast, dwell shows the degree to which the system itself has been engineered in an efficient manner. Such typical quantities are times for individual instruction executions on a computer system, and are characteristic only of the computer itself, not of the system's user.

Affinity is the least interpretable parameter and also the one that contains
more information than the other two. It is a time measure of a mixture of how frequently a state is used (utilization) and how long the state lasts on the average (dwell). Since efficiency is most often equated with time to perform a given task, affinity is a true indicator of regions that would benefit most from attention toward optimization to improve efficiency. It incorporates the influences upon efficiency due to system design structures and the particular application being evaluated. Unfortunately, this same mixing that gives a good indicator also masks the precise causes of the inefficiencies; either the system itself or the application using the system. Dwell and utilization are guides about how to optimize and affinity is a measure of how well the optimization job got done.

The system's arrival at particular states or sets of states also conveys information to the system viewer. A crosssectional set of states with selected values of variables in common can isolate large regions of system operation. Such state sets, then, can partition the system into large parts (subsystems) without requiring a detailed state-by-state analysis. For example, the system's program counter values can be divided into sets that identify major software sections or pages. An affinity measurement of these sets gives a broad picture of program activity which can be further narrowed down by altering the set sizes and bounds. Other measurements that involve the dynamics of state changes are necessary to fully cover the range of measurable parameters. They are related to the identification of specific states, and sequence of states that occurs between chosen states. We must be able to record the specific state sequence and instants of changes to characterize the system.
In summary, a measurement instrument for digital system should have the following basic capabilities:

A) Detect selected states

B) Record state change sequences

C) Measure and record the duration of selected states

and the elapsed time between selected states.

**Fundamental Program Monitor Circuit Model**

A fundamental block diagram of the program monitor is shown in Figure 3.6, illustrating the basic structure of the circuit. Probes connected to the external system measurement port carry signals that are shaped by the input processor. Much like the Synectics SUM, there are data probes and one control (strobe) line that indicates the instant to sample the data probes. In measuring a clocked system, the strobe probe could connect to the system's clock, thereby generating samples during the stable interval of system operation. The data then passes to a comparator circuit that compares the data against a set of limits to detect selected data values and sets of values. Initialization of the limits is performed by software through the I/O Bus and "Data In" line via computer I/O commands.

Dynamic alteration of the limits while sampling is therefore possible under program control. If the data satisfies the comparison criterion, the "Latch" is set, otherwise it is reset. When set, the latch holds electronic switch S1 closed, thereby connecting electronic switch S2 to the counter's clock line. S2 in turn selects what information the counter is to
accumulate; either clock pulses or individual comparator "true" pulses.

By making S2 also a programmable function much like limit initialization,
dynamic selection to accumulate either time or number of comparison
successes in the counter is provided.

In calculating $Dw_k$, for example, the limits are programmed to catch
occurrences of state $S_k$. Then on two sampling runs, S2 would first connect
to the time base and measure $T_k$, then second connect to the comparator and
measure $N_k$. For each run, the counter contents would be read into the CPU
via the "Data Out" line through the I/O bus; then the two numbers divided
in the CPU to yield $Dw_k$. Similar applications are possible to measure $T_0$
and $j$ in which the comparator is disabled so that direct counting of total
time or number of states is effected. Hence, $Aff_k$ and $Ut_k$ can be calcu-
lated from the four fundamental parameters measured by the program monitor.

A switch, S3, is provided to notify the computer when the counter has
either received a clock pulse or overflowed its counting capacity. In
these cases, a signal is sent to the CPU which interrupts the running
program, to permit immediate servicing of the program monitor. Inter-
rupting for each count can give a blow-by-blow profile of incoming data.
Lastly, a "Direct Path" is provided to allow direct recording of the
sampled data by the CPU. The comparator and counter logic are still
operable and are used to generate an interrupt signal to indicate when
the direct path contains valid data. Random and pseudo-random sampling
of external data via a program in the CPU is also feasible using the direct
feature.
Not shown on the block diagram is interface circuitry to communicate with the I/O bus, decode I/O commands, and store programming information for the programmed circuit elements. Also, in the interest of clarity, an alternate mode of operating the comparator has been omitted. Namely, the limits may be separated to operate independently of each other thereby generating two distinct comparator "True" output signals, call them "A" and "B". The two outputs, as already described, are taken in an "And" function wherein the comparison succeeds if both outputs A and B are true. In separating the comparisons, the outputs connect to S1 in a go/stop fashion whereby asserting A closes the switch to enable counting and asserting B opens the switch to disable counting. This go/stop mode is useful for detecting and measuring sequences of states that begin and end with distinct states programmed into the limits. In this manner, item C in the last section can be satisfied. Detailed presentations of the program monitor circuitry shall follow in later chapters.
CHAPTER 4
CIRCUIT DESCRIPTION OF THE PROGRAM MONITOR

Overview

The circuitry is composed of TTL series digital logic which is partitioned onto two Hewlett-Packard 2000 series interface cards, with related circuit functions grouped together. A block diagram in Figure 4.1 illustrates the primary functional components with the dotted line designating the card boundaries. All computer interface operations are performed by the I/O bus card (IOC) that plugs into the computer I/O cage of an H-P 21XX minicomputer. This card accepts I/O commands and data from the I/O bus, places data on the bus, manipulates the event latch and counter circuits, and signals the computer for I/O interrupt processing. Data sampling is performed by the front-end card (FEC) that plugs into the memory cage of an H-P 21MX minicomputer for direct measurement of computer activity, or plugs into the I/O cage of an H-P 2100 or 21MX minicomputer for remote sensing via hardware probes. The FEC accepts data from the external world, masks out unused bits, compares the data against preset limits using dual comparators, and reduces the data to essentially two signals, "event A" and "event B". Additionally, the FEC decodes I/O commands from the computer via the IOC and sets appropriate registers from the data. Interconnection of the two cards is performed with a fifty (50)-conductor flat (ribbon) cable, tradename "Scotchflex", manufactured by the 3M Company. This inter-card cable (ICC) carries all signals on Figure 4.1 that cross the dotted line, plus additional signals such as for control functions; see Table 4.1 for a list of all ICC signals. Importantly, note the line labelled "PM-bus"
<table>
<thead>
<tr>
<th>Cable Pin No.</th>
<th>Signal Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-16</td>
<td>PM-bus</td>
<td>Bus data bits 0-15</td>
</tr>
<tr>
<td>17,19,...,49</td>
<td></td>
<td>Circuit Ground</td>
</tr>
<tr>
<td>18</td>
<td>ST</td>
<td>Strobe I/O data from PM-bus</td>
</tr>
<tr>
<td>20</td>
<td>WRIT</td>
<td>I/O data enabled to PM-bus</td>
</tr>
<tr>
<td>22</td>
<td>CC</td>
<td>Master Clear</td>
</tr>
<tr>
<td>24</td>
<td>PWR</td>
<td>Power up pulse from CPU</td>
</tr>
<tr>
<td>26</td>
<td>unused</td>
<td>(future use as CPU freeze)</td>
</tr>
<tr>
<td>28</td>
<td>RE</td>
<td>Reset Event latch</td>
</tr>
<tr>
<td>30</td>
<td>RR</td>
<td>Reset Read function</td>
</tr>
<tr>
<td>32</td>
<td>TE</td>
<td>Transfer Enable counter/status to PM-bus</td>
</tr>
<tr>
<td>34</td>
<td>TB</td>
<td>Time Base</td>
</tr>
<tr>
<td>36</td>
<td>LM</td>
<td>Load Mode Register</td>
</tr>
<tr>
<td>38</td>
<td>EI</td>
<td>Enable Interrupts and start sampling</td>
</tr>
<tr>
<td>40</td>
<td>ESO</td>
<td>Event Strobe - Channel 0</td>
</tr>
<tr>
<td>42</td>
<td>ES1</td>
<td>- Channel 1</td>
</tr>
<tr>
<td>44</td>
<td>ES2</td>
<td>- Channel 2</td>
</tr>
<tr>
<td>46</td>
<td>ES3</td>
<td>- Channel 3</td>
</tr>
<tr>
<td>48</td>
<td>EVB</td>
<td>Event data &quot;B&quot; comparator</td>
</tr>
<tr>
<td>50</td>
<td>EVA</td>
<td>Event data &quot;A&quot; comparator</td>
</tr>
</tbody>
</table>

Table 4.1 - Intercard Cable (ICC) Signal Assignments
that consists of sixteen signals plus ground. This is a bus that carries
information to and from each card (bidirectional) and is tri-state in
circuit nature. Primary communication and unification of all of the
program monitor blocks is implemented with the PM-bus. Structuring around
this bus permits future expansion of capabilities by adding functional
blocks to the bus. Care was observed in alternating pulsed control signal
wires with circuit grounds, thereby reducing capacitive and inductive
coupling between otherwise adjacent signal wires that is characteristic of
flat cable.

**Circuit Summary**

A brief description of how the program monitor works is presented in
this section as a synopsis of this chapter. To illustrate how the circuitry
functions as a whole, we shall step through a typical example of a data
sampling cycle, noting important points along the way. First, let's assume
that the circuitry has already been programmed by appropriate I/O commands
and that data sampling has commenced. Refer to Figure 4.1, the program
monitor block diagram, during the following discussion of the collection
of a data sample:

1. An input strobe pulse occurs on the front-end card (FEC)
signalling acquisition of a sample.

2. Data from either the FEC bus edge connector or external probes
(FEC top edge connector) as preselected, is strobed into the
data input latch. Note that all sixteen data bits are sampled
simultaneously. A FEC circuit jumper selects latching on either
the rising or falling edge of the strobe pulse.
3. The sampling clock is started, which generates pulses to cycle through each of the four data collection channels, and to sample the comparator event lines at the appropriate times.

4. The latched data is subsequently masked, to set unwanted bits to zero, and is passed into two sets of comparator circuits. Each comparator utilizes preloaded constants in its limit memories (random access memories, RAMs) to compare against the masked sample word. The comparison functions performed, (i.e. combinations of "<, >, =" separately selected for each channel) are accessed from pairs of RAMs.

5. Two event result signals, A and B, are generated (one from each comparator) and have true values if their comparison conditions (limits and functions) are satisfied. These signals are held by event latches (one per channel) after combinational logic is performed on them as per the event mode bit. Event mode, a programmed function, determines the condition(s) by which each latch is set or reset as follows:

a) Mode=0, latch is set if both event signals are true, no change, otherwise

b) Mode=1, latch is set if event A is true, latch is reset if event B is true, no change, otherwise.
6. Counter mode subsequently selects a pulse source to be counted by each of the four 16-bit channel event counters as follows:
   a) Mode=0, if an event occurs (as defined by the latch being set when a sample it taken), then a pulse is counted
   b) Mode=1, if a clock pulse occurs from the onboard time base generator and the latch is set, then a pulse is counted.

Each channel has an individual counter mode bit to specify either measuring number of event occurrences or event time duration for its specific comparison conditions.

7. When a counter reaches a count of 1 or 65,535 (overflow) as designated by its interrupt mode bit, a computer I/O interrupt is generated by the program monitor I/O card (IOC).

8. Steps 3-6 are repeated sequentially for each of the four independent data collection channels on the program monitor. Different limits and functions for each channel provide different measurements from the same data source being sampled.

Other features of importance include the following:
   a) A query to the IOC status register permits software determination of the interrupting channel(s).
   b) The contents of a counter may be read under program control, during or after data collection without loss of data.
   c) The data source used for sampling is available for direct reading if so desired.
d) All programming of various selection registers and limits and limit functions are done with simple sequences of I/O output instructions.

At this point, an important structural note is in order. The program monitor has been designed to carry on four (4) measurement tasks at a time. Most of the logic on the I/O card has been duplicated to implement the four channels of data collection. All data enters through a common input processor and comparator logic section. The comparators are utilized in a time-shared manner, by cycling through the limits and generating comparator outputs for each channel at a different point in time. For high speed operation (see last section of this chapter), cycling can be reduced to fewer than four channels via a circuit strap. Programming for the channels is independent; i.e., each channel can be configured to measure a different parameter of the input data, as was suggested in chapter 3 for deriving dwell, affinity, and utilization parameters.

Input Processor

The function of the input processor is to interface the external world being measured with the program monitor. Appendix A, Figures A.1 and A.2, illustrate the data and control portions, respectively, of the input processor circuitry. Data enters through two cascaded multiplexors that select the data source to be sampled (hardware probes, 21MX S-bus, or 21MX M-bus) as programmed into the "Source Select" register. Similarly, the control strobes enter through multiplexors in conjunction with some combina-
national logic. Signal lines X0, X1, and X2 select the precise 21MX bus
strobe to use to sample one of the many 21MX CPU registers. The logic equations implemented to select the various registers are shown in Table 4.2. It should be noted that the option is provided, via a circuit strap (jumper), to filter out memory address references from 21MX direct memory accesses (DMA). This allows separation of memory references due to software from those due to direct memory access I/O devices. Signal line XS selects memory backplane or external probes as the source of data and strobe signals. XS=1 (probes) will override any values of X0, X1, or X2. When using the probe strobe, a circuit jumper selects the polarity (normally high or low) of the strobe to trigger sampling.

Following the data multiplexors, the sampled data travels to a set of AND gates that mask out selected bits as programmed into the "MASK" register. The masked data then proceeds to the comparator logic over the "X-bus" for further processing. Masking gives the option of ignoring certain bits that are not of interest to the measurement at hand.

A "Direct Path" exists for additional data processing and storage beyond that provided by the program monitor, by bypassing the comparator and statistics gathering logic, and placing the raw data onto the computer I/O bus. Data can be fed into this path from one of two sources, as selected by signal DS (direct select), and through the multiplexor and bus driver logic. Direct data, without masking, or after masking and offset addition, feeds through the direct path to the PM-bus, over the intercard cable, and out the I/O card to the computer I/O bus under software control. The section in chapter 5 on DMA and DMI will discuss the utility of the direct path in automated medium-speed measurement applications.
CPU Register to Sample

Memory Address (M-reg)
  Read Only
  Write Only
  Read or Write

Instruction Register (I-reg)

Memory Contents (T-reg)

Program Counter (P-reg)

Logic Equation to Constitute Strobe Signal for Register (21MX Memory Cage Signal Names)

<table>
<thead>
<tr>
<th>Including DMA</th>
<th>Ignoring DMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>READ·DMAEN</td>
</tr>
<tr>
<td>WRITE</td>
<td>WRITE·DMAEN</td>
</tr>
<tr>
<td>READ + WRITE</td>
<td>(READ+WRITE)·DMAEN</td>
</tr>
</tbody>
</table>

IRSTF·P5NF

(TEN+TST)·P5NF

FTCH·P5NF

* Direct Memory Access

Table 4.2 - Logic Equations for 21MX CPU Register Strobing
The strobe after all multiplexing becomes signal "ST", and trips an S-R latch (Sample Latch) constructed of cross-coupled NOR gates which in turn enables the VCO (voltage-controlled oscillator) that serves as a clock generator and strobes the input data via signal $\overline{SD}$. Clock pulses ($\overline{Cl}$) begin after the characteristic startup time of the VCO, thereby allowing the sampled data sufficient time to propagate through the input processor, comparators, intercard cable, and event latch circuitry (refer to timing diagrams, Figures A.12 and A.13 for this discussion). $\overline{CLOCK}$, which is generated from $\overline{Cl}$ and the channel counter output $Q_A$, provides an inverted clock signal with one-quarter duty cycle. Each inverted clock pulse is subsequently demultiplexed, according to the channel counter value, onto one of four event strobe lines ($\overline{ESO}$ through $\overline{ES3}$) that travel over the ICC to strobe each channel event latch. The channel counter changes on the rising edge of $\overline{Cl}$ (after the event has been strobed), thereby beginning the next channel cycle. Upon completion of one, two, or four channel cycles, as selected by the channels select circuit jumper, signal "END" rises to reset the sampling latch, clear the channel counter to zero, stop the clock generator, and reenable the strobe multiplexer via signal $\overline{EN}$ for the next sampling cycle. Note that holding the strobe line asserted for longer than a sample cycle, causes a new cycle to commence immediately after the prior one completes, since $\overline{EN}$ will generate a new ST.

**Comparators**

Data for comparison from the input processor enters along the X-bus to the comparator circuits that consist of two-stage cascades of 74S85 magnitude comparators in Figure A.3. The two-stage design gives minimum
propagation delay for a sixteen-bit comparison by avoiding multiple ripple delay through the comparator output pins. Preloaded sixteen-bit limits (A and B) in random access memories (RAM's) are compared against the incoming data on the X-bus. The result is three signal wires from each comparator to indicate whether the sampled data is less than, greater than, or equal to (\(<\), \(>\), \(=\)) the respective preloaded limit. Selection of the particular comparator functions is then performed by AND-OR-INVERT gates in conjunction with preloaded function RAMs, one entry for each channel. The function RAMs thereby set the condition for successful comparison using any combination of the less than, greater than, or equal functions. In this manner, the comparators may be programmed to detect an inclusive range of input values by setting "A" for \(<\) and "B" for \(>\) and loading an A limit that exceeds a B limit. Limits and limit functions for all four channels are contained by the RAMs that are four words deep and twenty bits wide (16 data, 3 function, 1 unused). Accessing of the RAM information is performed sequentially, one channel at a time, thereby reusing the same comparators for all the channel comparisons. The circuitry of the control input processor synchronizes read address signals (R0, R1) with the comparison data and hence governs RAM limit accesses with latching of the event signals (EVA and EVB) by the event strobes. The parallel data and serial channel comparison algorithm provides a good compromise of integrated circuit package count versus speed. A number of design topologies were considered, e.g. bit serial, nibble serial, and channel parallel. Each had significantly worse speed/package ratios than the employed topology, given the present array of off-the-shelf TTL devices. Sampling speed
was a primary concern during implementation and for this reason Schottky type TTL circuits are used throughout most of the front-end card. High-speed RAMs in the form of 4x4 register files with twenty (20) nanosecond typical access time provide fast channel changing. As is noted on the Channel Cycle Timing Diagram in Appendix A, channel changing is complete in 105nS maximum, 73nS typical, fairly high-speed of operation. For very high-speed sampling applications, channel changing can be limited to fewer than the full number of four channels to gain higher sampling speed at the sacrifice of multiple measurements.

**Event Collection**

Following the comparators, the reduced data travels across the ICC to the I/O card where it feeds into the "Event Encoder" (refer to Figure A.4). This circuit performs a unique combinational logic function. If signal "EES" (event encoder select) from the "Event Mode" register is zero, then the comparator outputs are combined in an AND function to set and reset the "Event Latch"; that is, if events A and B both are true then the latch is set, otherwise it is reset. Whereas, if EES=1, then comparator A sets the latch and comparator B resets the latch. EES therefore selects the AND or go/stop modes that are useful in some measurements as mentioned in chapter 3. Implementation is performed with a 74156 (3-to-8 line demultiplexer with open-collector outputs) for each channel, and a 74279 (quad S•R latch).

The three inputs to each 74156, the two comparator outputs and the event encoder select, are decoded to one of eight outputs that are wire ANDed in the desired manner and connected to a latch. Refer to the truth table in Table 4.3 that shows how the wired AND connections were chosen. All outputs
### Inputs

<table>
<thead>
<tr>
<th>EES (C)</th>
<th>EVB (B)</th>
<th>EVA (A)</th>
<th>Outputs</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Y0, 0, 1</td>
<td>&quot;AND&quot; = 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Y1, 1, 0</td>
<td>= 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Y2, 1, 0</td>
<td>= 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Y3, 1, 0</td>
<td>= 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Y4, 1, 1</td>
<td>Both comparators true (ignore event)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Y5, 1, 0</td>
<td>Comparator B true</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Y6, 0, 1</td>
<td>Comparator A true</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Y7, 1, 1</td>
<td>Neither comparator true (no change)</td>
</tr>
</tbody>
</table>

\[
\overline{ES} = Y0 \cdot \overline{Y6} \quad 0 \quad 1 = \text{Set latch}
\]

\[
\overline{ER} = Y1 \cdot \overline{Y2} \cdot \overline{Y3} \cdot \overline{Y5} \quad 1 \quad 0 = \text{Reset latch}
\]

\[
1 \quad 1 = \text{No change}
\]

\[
0 \quad 0 \quad \text{not possible due to 74156}
\]

---

**Table 4.3 - Truth Table for Event Encoder**
of the encoder are normally high, and when the event strobe is asserted, one of the eight outputs falls in accordance with the encoder truth table to set, reset, or leave the latch unchanged. Proper timing of the event strobe by the input processor control circuit, to permit event data settling, avoids glitches at the event encoder's output that might falsely alter the event latch. Resistive pullup is provided for \( \text{ES} \), whereas wire-tying is used for \( \text{ER} \) so that resetting of the latch can be done by associated control circuitry. It is not generally good practice to wire-tie a TTL totem-pole output, but in this case no damage occurs since the event encoder is enabled for such a short time (maximum of one-tenth duty-cycle per encoder).

Whenever the latch is set (\( \text{EL}=0 \)), then the "Event/Time Selector" is enabled to pass information from either the A or B input to the V output, as selected by the signal ETS (event timer select). The selected event strobe pulses or time base pulses are then fed to the clock line of the 16-bit "Event Counter", constructed from four 8554 4-bit synchronous counter/latches with tri-state outputs. Note that the inverted strobe pulse is fed through the event/time selector so that the master-slave type 8554 receives the appropriate sense clock (normally high) for event counting. Also, this circuit is glitch-free, especially in the case where the event latch changes from the set to the reset state.

The event counters, one per channel, each have built-in latches and tri-state driver circuitry that drive the PM-bus to read the counter values into the computer over the I/O bus. The counters' internal latches are normally connected to the counters and constantly monitor the counts. When
the computer wishes to read the value of a particular counter, one of the four transfer enable signals (TE0-TE3) is asserted to disconnect the appropriate latch from its counter and enable its bus driver. By the proper choice of transfer enable signal, any of the four counters may be read under software control. Immediately after latching, the counter is automatically cleared to zero by the appropriate RC (reset counter) signal. The computer may then read the count from the latch that is held static, while data collection can progress in the same counter. All data can therefore be collected during the sampling session, without the loss of data when counter reading occurs.

Event Initialization

Much of the detailed control discipline for event collection is performed automatically in order to free the programmer of these tasks. Refer to Figure A.6 that contains circuitry to manipulate event latch resetting and event counter resetting and transfer arbitration. Each time that a computer I/O write instruction is done to the program monitor, the "Address Latch" retains bits 0-2, the data collection channel address. These address bits are fed to three demultiplexors, the first of which is labelled "Transfer Enable Selector". Upon reception of the "TE" (transfer enable) signal from the front-end card, this selector enables one of its five outputs, in accordance with the input address, to enable data onto the PM-bus for computer reading. TE is retracted by the front-end card upon reception of a computer I/O read instruction. In the case of a counter select (address value 0-3), the TE signal also latches the designated event counter, immediately after which the inverted "RC" (reset counter) pulse is
generated to clear the counter via the "Reset Counter Selector" that generates RC0 through RC3.

Whenever program monitor channel programming is performed (i.e. limit and limit function initialization via I/O commands), \( \overline{RE} \) is generated. It is demultiplexed to the channel being programmed by the "Reset Event Selector" and fed as an inverted pulse into the "Auto Clear Control" multiplexer. Under programming conditions, \( \overline{CC}=1 \) and \( \overline{CLFP}=1 \) cause the pulse to feed through the auto clear control to reset the channel's event latch and counter. This guarantees full clearing of the channel prior to data collection on the channel. \( \overline{CC} \) is produced by a power-up or master clear command, and when it is applied to the auto clear control, will cause all four outputs to fall, thereby clearing all event latches and counters. Thusly, a consistent initialization state on clear and power-on is implemented. When \( \overline{CLFP} \) (clear flag pulse) is asserted due to a computer CLF (clear flag) machine instruction, the auto clear multiplexer switches to input A to connect the "Auto Clear" register to the output. When this happens, selected bits as programmed into the register, clear selected channel latches and counters. This register is primarily intended for use with automated data collection where software intervention for interrupt processing is not possible; see the section on DMA and DMI in chapter 5 for a discussion of automatic data collection.

Interrupts

With an interrupt capability, the computer can continue processing other programs and yet be able to service special requests from the program monitor upon demand. Such requests from the program monitor are signalled
to the computer over the I/O bus via signal "IRQL" (interrupt request). The program monitor causes IRQL to be asserted by dropping signal line "INT" (interrupt); refer to Figures A.4 and A.5. This can be due to interrupt requests on any of the four data collection channels as detected by the 7423 OR-AND-INVERT gate. As selected by signal "EIS" (event interrupt select) in Figure A.4, the "Interrupt Selector" routes either the counter's carry out signal or direct clock pulses from the counter to signals I0 through I3 (one per channel). The "Interrupt Mask" enables each of the four channels to interrupt, as preprogrammed by software. By selectively enabling channels to interrupt, spurious interrupts from unprogrammed channels can be omitted. The rising edge of I0 sets the "Interrupt Latch", Figure A.5, whose output triggers an I/O interrupt. Each interrupt latch remains set until the appropriate channel collection data is processed which generates a reset counter signal from Figure A.6. Therefore, continuous computer interruption will occur from the program monitor until all interrupting channels are processed. To facilitate interrupt identification from specific program monitor channels, an "Interrupt Status" register is given which may be read by software much in the way counter information is read.

**Programmable Time Base**

A crystal-controlled time base generator, Figure A.7, provides a timing reference waveform for gating into the event counters in applications requiring accurate timing information. The basic oscillator frequency was chosen to be 1MHz (1 microsecond per pulse) as limited by the speed of the time base divider chip, a Mostek Corp. MK5009.\(^{17}\) A four-bit code loaded
into the "Time Code" register from software specifies the divisor constant used by the divider to reduce the basic one megahertz frequency to produce signal "TB" (time base). The reader is referred to reference (18) for a discussion of the standard crystal shaping network used at the FB1 and FB2 terminals of the MK5009. Time base accuracy is basically that of the crystal, namely ±0.001% at room temperature. Verification of accuracy under constant room temperature conditions was made using a seven-digit frequency counter to measure TB at 1MHz for a measurement period of two hours. Drift with temperature is possible due to physical changes in the crystal and should lie within the crystal tolerance.

**Programming Command Decode**

The circuitry shown in Figure A.8 performs decoding and executing of programming commands for the program monitor. Each I/O output instruction (OTA or OTB) is received by the I/O card and gated onto the PM-bus, for reception by the front-end card and strobed by signal "ST". The information on the PM-bus is interpreted as either a programming instruction, or data to be loaded into a programming register, as indicated by the state of the "Data Mode" flip-flop. While in command mode (data mode flip-flop = 0), bits 12-14 of the PM-bus are interpreted as a command code by the "Command Decoder" that generates one of eight inverted output pulses, each corresponding to a particular command.

"Clear" generates signal CC (master clear) to all parts of the circuit for initializing the program monitor data collection circuits to predetermined values, which are given in the next chapter.
"Set Channel" produces signals $\overline{RE}$ and $\overline{LF}$ to reset the event latch and load the function RAMs, respectively, of the channel whose address is on PM-bus bits 0 and 1. Also, the "Limits/Processor Select Latch" is set to indicate limits mode. If bit 2 of the PM-bus is on (a request for data mode), then the circuitry switches into data mode for the subsequent two programming words, otherwise it remains in command mode. Basically, the two 7474 flip-flops form a synchronous counter, that in conjunction with the limits/processor select latch, generates strobe pulses for loading either the channel's limit RAM registers or the input processor registers with the next two programming words. Following the two data words, the circuitry automatically reverts to command mode for command interpretation by resetting the data mode flip-flop.

"Set Processor" operates in an analogous way to the set channel command, except that the limits/processor latch is reset to indicate a request for input processor loading.

"Load Time" simply sends a pulse to load the time code register from the PM-bus for the programmable time base in Figure A.7.

"Read Select" initiates the sequence of operations to gather data from an event counter or from the interrupt status register into the computer. This pulse sets the "Read Mode Request Latch" to indicate that a read function has been requested. Upon receipt of $SD$ from Figure A.2, to indicate that data collection is between strobe pulses and therefore the event counters are stable, the "Read Mode Latch" is set. Signal "TE" (transfer enable) is then asserted, disabling the direct path (TED) from the PM-bus, and enabling the transfer enable selector in Figure A.6. The transfer
selector utilizes the address latch that was set from PM-bus bits 0-2 when the read select command was first issued. The appropriate counter, while stable, is then latched by one of TEO-TE3 and then reset by one of RCO-RC3. (In the case of accessing the interrupt status register, it is not altered). Note that the preceding actions all occur between data input samples, thereby guaranteeing valid data and no slowdown in data collection while statistics are read. TEO-TE4 hold the data on the PM-bus indefinitely while the computer software reads the data by issuing an I/O input machine instruction, which generates a computer "I01" I/O pulse in Figure A.11. The read sequence is terminated by the trailing edge of I01 that produces the short "RR" (reset read) pulse to clear the read mode request and read mode latches. Until an I01 pulse arrives, the read circuitry is set to read the counter designated by the address latch in Figure A.6. If any output command is issued without clearing the read with an I01, the address latch will be reloaded and the transfer enable selector output might switch to another counter. This will result in loss of the latched data on the original counter and latching of a different counter, possibly during a data collection cycle to cause a latched transient count. The data read by the software might look peculiar, and in any case is in error. Future program monitor improvements could remedy this pitfall by interlocking programming during an active read cycle.

"Set Modes" generates signal "LM" (load modes) to load programming information into the event mode register in Figure A.4.

"Stop Sampling" resets the "Sampling Enabled Latch", thereby setting SE to zero and disabling future data input strobos in Figure A.2. A
measurement cycle in progress runs to completion if SE falls during the cycle.

"Enable Interrupts/Start Sampling" generates signal EI (enable interrupts) to load the interrupt mask and auto clear registers in Figures A.4 and A.6, respectively. The sampling enabled latch is also set, thereby asserting SE and permitting measurement cycles to commence on the subsequent input strobe pulse.

**PM-bus Arbitration**

Since the program monitor has a multi-source tri-state PM-bus, arbitration of the bus is necessary to prevent two or more users from driving the bus at the same time. A simple bus priority scheme was implemented using a few gates to arbitrate among the 8554 event counters, interrupt status register, HP106A I/O bus drivers, and the 8123 direct path drivers. The HP106A's have the highest priority and can take the bus from the other users; signal WRIT disables TED (direct) in Figure A.8 and all of the counters via their OD2 pins in Figure A.4. The second highest priority sources are the counters and status register that can override TED via signal TE in Figure A.8. Note that the transfer selector arbitrates among the counters and status register. The direct path is the lowest priority source and is always present on the PM-bus whenever the other sources are disabled.

A bug exists in the presented circuitry. Under normal conditions, properly functioning software will complete a read cycle by an I0I before reprogramming is attempted with an I00. If a programming command is issued while the interrupt status register is on the PM-bus due to an incomplete
read cycle, the register is not disabled as the counters are under similar circumstances. This bug should be corrected in future program monitors.

**I/O Bus Data Interface**

Data is exchanged between the PM-bus and the 21XX computer I/O bus (IOBI signals for input and IOBO signals for output) in a 16-bit parallel manner as shown in Figure A.9. Custom H-P tri-state drivers, HPL06A, are used to take data from the I/O bus and place it on the PM-bus whenever signal "WRIT" is asserted, which occurs for a computer I/O output instruction. Information transfer from the PM-bus to the I/O bus occurs for computer I/O input instructions, via signal "OI0I". Special chips with open-emitter outputs are used to be compatible with the H-P 21XX I/O bus that is described in reference (19).

**I/O Bus Interrupt Interface**

Five circuit elements in Figure A.10 determine whether an I/O interrupt can occur to assert signal "IRQL" (interrupt request). "PRH" (priority chain signal from the adjacent higher priority I/O interface card), "IEN" (interrupt system enable), "Flag" and "Flag Buffer" flip-flops, and the "Control" flip-flop all must be asserted to cause an interrupt. The timing diagram in Figure A.14 shows important I/O signals as related to an I/O cycle that is divided into five equally long "T" periods. Note that the length of T varies in accordance with the particular type of 21XX computer.

When signal INT from Figure A.5 falls, thereby requesting an I/O interrupt, the flag buffer flip-flop is set. Upon the first occurrence of
a T2 period, "ENF" (enable flag) gates the flag buffer flip-flop into the flag flip-flop to set it and clears the IRQ flip-flop. Then, upon T5, "SIR" (service interrupt) is asserted to set the IRQ flip-flop to cause an interrupt. In response to IRQL, the computer replies with signal "IAK" (interrupt acknowledge) to reset the flag buffer flip-flop and prevent duplicate interruption. The flag flip-flop must be reset by the interrupt service software to avoid hanging up lower priority I/O channels.

I/O Bus Control Interface

The signals shown in Figure A.11 determine the timing for reading from and writing to the program monitor for software interfacing. Signal "I00" (I/O output from the computer) is buffered to make signal "STROB" that latches the channel address into the address latch in Figure A.6. The rising edge of I00 also triggers the long "WRIT" signal that transfers the I/O bus data onto the PM-bus, as per the timing diagram in Figure A.15. When I00 falls, strobe pulse "ST" is produced that signals the command decoder in Figure A.8 to operate. The long delay from the rising edge of WRIT to generating ST, permits sufficient time for the computer I/O bus and PM-bus to stabilize completely. WRIT is a rather long pulse to guarantee that the PM-bus is stable long enough to perform all of the I/O data-related operations for both the 2100 and the slower 21MX I/O. Signal "IOI" (I/O input to the computer) is buffered to generate OIOI to transfer the PM-bus onto the I/O bus. A read select command designates the source of data on the PM-bus prior to the actual IOI. Absence of such a selection results in the direct path being read by the IOI signal. The falling edge of IOI produces the short "RR" (reset read) pulse, that resets
the read mode latches on Figure A.8 after the read operation completes. Note that reset read is not damaging if the direct path was read, that is, none of the event latches and counters is altered.

**Circuit Specifications and Limitations**

A listing of vital circuit specifications is given in Table 4.4 based upon worst-case parameters for individual circuit components. The input probe loading is approximated for a parallel wire transmission of equivalent geometry to the flat cable.

Sampling cycle time, the total time to process a sample, is given for 1, 2, and 4 channels as selected by the channel select jumper in Figure A.2. Continuous sampling, a high-speed operation initiated by holding the probe strobe asserted continuously, results in samples being taken as fast as the program monitor can cycle. A specification is given for one channel cycling where high speed operation is most justified.

Sample to I/O interrupt time is the time between a sample strobe pulse and the computer I/O interrupt for an event occurrence. This time varies in accordance with the synchronization of the strobe and particular data collection channel with the computer's T2 I/O period, and can be as long as two entire I/O cycles. The specification given does not include the time for the computer to pick up the interrupt and enter the appropriate interrupt software.

The intercard cable can be varied in length according to the particular measurement operation as will be shown in a later chapter. Successful operation of the program monitor has been observed with a "Scotchflex" cable as long as fifteen (15) feet in length.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Probes</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>16 - sampled simultaneously</td>
</tr>
<tr>
<td>Voltage Levels</td>
<td>TTL compatible</td>
</tr>
<tr>
<td>Length</td>
<td>Up to 15', flat cable probes</td>
</tr>
<tr>
<td>Source Loading</td>
<td>One TTL load + 10pf/foot flat cable</td>
</tr>
<tr>
<td>Sampling Cycle Time</td>
<td></td>
</tr>
<tr>
<td>1 Channel</td>
<td>270 nsec</td>
</tr>
<tr>
<td>2 Channels</td>
<td>410 nsec</td>
</tr>
<tr>
<td>4 Channels</td>
<td>700 nsec</td>
</tr>
<tr>
<td>Continuous 1 Channel</td>
<td>145 nsec</td>
</tr>
<tr>
<td>Sample to I/O Interrupt Time</td>
<td>1.96 μsec max. (2100)</td>
</tr>
<tr>
<td>Minicomputer Control</td>
<td>Hewlett-Packard 2100 or 21MX</td>
</tr>
<tr>
<td>Intercard Cable Length</td>
<td>15' max. of 50-pin flat cable</td>
</tr>
<tr>
<td>Time Base</td>
<td></td>
</tr>
<tr>
<td>Resolution</td>
<td>1 μsec to 1 hour, programmed</td>
</tr>
<tr>
<td>Accuracy</td>
<td>±0.005% = 50 parts per million</td>
</tr>
<tr>
<td>Counters</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>4, independently functioning</td>
</tr>
<tr>
<td>Count Capacity</td>
<td>16 bits = 65,536 counts</td>
</tr>
<tr>
<td>Speed</td>
<td>15 Megahertz or 66 nsec/count</td>
</tr>
</tbody>
</table>

Table 4.4 - Program Monitor Circuit Specifications
There are four counters, one for each channel, each sixteen (16) bits in length. Greater counting capacity is possible by using computer software to account for counter overflow occurrences. The maximum specified counter speed corresponds to an event every 66 nanoseconds. Since sampling cycle time is limited to the continuous cycling value of 145 nanoseconds, maximum counter rates are not reached in data collection.
CHAPTER 5
SOFTWARE INTERFACING THE PROGRAM MONITOR

Overview

The program monitor is setup (configured) or "programmed" for data collection by initializing various data registers and control latches via software and by selecting specific circuit jumper wires manually. As explained in chapter 4, the program monitor plugs into the I/O bus of a Hewlett-Packard 2000 series minicomputer and appears to the computer as an I/O peripheral device.\(^\text{20,21}\) Therefore, the program monitor receives programming information from the minicomputer in the form of I/O output instructions. Likewise, I/O input instructions obtain data from the program monitor for processing by the minicomputer software. Refer to Figure 5.1 that shows all of the I/O instructions with brief descriptions of their function; they will be more fully explained with references to other Figures (5.3-5.6) located at the end of this chapter.

All I/O instructions designate the slot address (select code) of the program monitor I/O card that interfaces with the I/O bus. Output instructions (OTA or OTB) send the 16-bit contents of the computer's A or B register to the program monitor for action. These command words (CW) direct specific program monitor actions, as condensed in Figure 5.2 on the next page. As shown in the general command format in this Figure, bits 14, 13, 12, abbreviated CW(14-12), contain the command op-code (eight commands in all). CW(11-0) may supply specific programming data depending upon the particular command. Two commands, "Load Data Channel" and "Load Input
<table>
<thead>
<tr>
<th>I/O Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLC sc*</td>
<td>Prevent interrupts.</td>
</tr>
<tr>
<td>STC sc</td>
<td>Permit interrupts.</td>
</tr>
<tr>
<td>CLF sc</td>
<td>Clear flag flip-flop.</td>
</tr>
<tr>
<td></td>
<td>Also clears interrupt latch, event latch, and event counter for all &quot;auto-clear&quot; channels.</td>
</tr>
<tr>
<td>STF sc</td>
<td>Simulate a program monitor interrupt by setting the flag flip-flop.</td>
</tr>
<tr>
<td>LIA sc</td>
<td>Read an event counter or interrupt status register as selected by the prior &quot;read select&quot; command.</td>
</tr>
<tr>
<td>LIB sc</td>
<td>Read direct data if no prior read select.</td>
</tr>
<tr>
<td>MIA sc</td>
<td></td>
</tr>
<tr>
<td>MIB sc</td>
<td></td>
</tr>
<tr>
<td>OTA sc</td>
<td>Send a programming command or data value contained in the A (or B) register.</td>
</tr>
<tr>
<td>OTB sc</td>
<td></td>
</tr>
</tbody>
</table>

*sc = select code (I/O slot) of the program monitor I/O card.

Figure 5.1 I/O Instructions
General Command Format

<table>
<thead>
<tr>
<th>Bit: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>*   C C C D D D D D D D D D D D D D</td>
</tr>
</tbody>
</table>

Command Op-code
Command Data (if used)

* Bit ignored

Commands in Brief

<table>
<thead>
<tr>
<th>Name</th>
<th>Op-code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear</td>
<td>000</td>
<td>Initializes all circuit functions as per Figure 5.4.</td>
</tr>
<tr>
<td>Load Data Channel</td>
<td>001</td>
<td>Loads function registers and limit registers if requested for specified data collection channel.</td>
</tr>
<tr>
<td>Load Time Code</td>
<td>010</td>
<td>Resets the time base generator to zero and loads the time base divisor register, see Figure 5.5.</td>
</tr>
<tr>
<td>Load Input Processor</td>
<td>011</td>
<td>Selects the input source for sampling and the direct path routing. Mask and offset registers are loaded if requested.</td>
</tr>
<tr>
<td>Read Select</td>
<td>100</td>
<td>Selects the source for software reading by an I/O input instruction, e.g. LIA.</td>
</tr>
<tr>
<td>Load Data Modes</td>
<td>101</td>
<td>Loads event, counter, and interrupt mode registers for all data channels.</td>
</tr>
<tr>
<td>Stop Sampling</td>
<td>110</td>
<td>Terminates data sampling; a sample in progress is completed first.</td>
</tr>
<tr>
<td>Start Sampling</td>
<td>111</td>
<td>Designates auto-clear channels, specifies the interrupt mask, and initiates sampling.</td>
</tr>
</tbody>
</table>

Figure 5.2 Programming Commands Summary
Processor", can request loading of 16-bit program monitor registers via CW(2). For such requests, the program monitor drops into "data mode" wherein the subsequent two data words are used verbatim as data and loaded into one of the 16-bit registers. After the second data word, "command mode" is resumed and command op-codes are interpreted accordingly for subsequent command words. Switching back and forth between command and data mode can be done any number of times during configuration. The reader is advised to refer to the block diagram in Figure 4.1 and to the detailed command formats in Figure 5.3 throughout the following sections that discuss the commands in detail.

**Clear Command** (Op-code 000)

All four data collection channels are cleared and the input processor functions and time base interval are initialized as given in Figure 5.4. This command should be issued upon reconfiguring the circuitry for each measurement session to guarantee clearing of prior collected data. Three clears in a row sent to the program monitor (2 to inactivate data mode, 1 to actually clear) insure that the clear command is processed, just in case data mode was left activated by former software.

**Load Data Channel Command** (Op-code 001)

One of the four comparator channels, whose address is specified by CW(0-1), is configured. CW(11-9) and CW(8-6) set the B and A comparison function registers, respectively. For example, CW(11-9)=110 says that comparator B's output will be true if the sample word is logically greater than or equal to the channel's B limit word. Note that any combination of
function bits is possible; 000=comparison never succeeds, 111=comparison always succeeds regardless of the sample or limit word values.

As in the load input processor command, CW(2)=1 places the program monitor into data mode for the subsequent two programming words. The first such word goes into limit RAM A and the second into limit RAM B. Command mode resumes automatically after reception of the second limit. In order to load all eight limits (A and B for each of the four channels), four load data channel commands are issued each followed by two output commands for loading the limits. Each time that the load channel command is executed, the program monitor clears the designated channel's event latch, event counter, and interrupt latch, thereby clearing any collected data in the channel.

**Load Time Code Command (Op-code 010)**

The four-bit time base divisor register is loaded from CW(3-0), thereby setting the time base generator output period as given by the codes in Figure 5.5. The programmable time base can be used in conjunction with the event counters to measure the duration and time between sampled data events. Note that the program monitor's time base is distinct from and independent of the H-P 2000 series Time-Base Generator (TBG) interface card, (product number 12539C).
Load Input Processor Command (Op-code 011)

CW(0) specifies the input source from which data is sampled; either the external probes from the FEC exposed edge connector, or the FEC bus edge connector for direct 21MX computer sampling is used. CW(9-11) select the specific 21MX CPU register to be sampled from the bus edge connector (when CW(0) specifies such sampling). For cases when CW(0) designates external probes, CW(9-11) are ignored. CW(1) routes data along the direct path, either through or around the mask-adder circuit path. Note that the mask is always used prior to comparison, irrespective of the direct path routing. CW(2) effects loading of the input mask and adder offset registers. If CW(2)=1, then the program monitor is placed into "data mode", and accepts verbatim the next two 16-bit programming words to load first the mask, and second the offset. Data mode is terminated after acceptance of the second data word, reverting to interpretation of programming word op-codes. If CW(2)=0, then the mask and offset registers are left unaltered, and data mode is omitted.

The boolean "and" function is performed on the input data word using the mask register (16-bits each). Each sample bit to use for comparison should have its corresponding mask bit set to 1. Similarly, a mask bit of 0 will clear the respective data input bit to 0 for comparison. Offset addition is done after masking and the result (16-bit unsigned integer, carry ignored) goes onto the direct path, if so selected by CW(1). If not selected for the direct path, the adder has no effect on the direct data. Note that data going to the comparator does not pass through the adder.
Read Select Command (Op-code 100)

This command selects the contents of the interrupt status register, or one of the four event counters for subsequent reading via an I/O input instruction (LIA, LIB, MIA, or MIB) to the I/O card. Codes for selecting the sources are listed in Figure 5.3 (some codes have no selection effect and therefore produce an indeterminate data value). Upon reception of a counter select request, the program monitor waits for the current measurement cycle to complete, then samples and retains (latches) the counter contents until it is read by software. The selected channel's counter and interrupt latch are cleared immediately after the counter contents are held (still between measurement cycles) so that double-counting and interrupting do not occur. Sampling operations can therefore continue to acquire new data between the time of selection and reading. It is emphasized that the counters may be read during data collection and counting without any loss or duplication of data; the program monitor takes care of synchronizing software activity with data collection. The programmer is advised against performing any I/O output command to the program monitor before reading a pending event counter value that has been selected. Doing so can cause loss of the counter value and destruction of other counter data. Completion of the read cycle with an I/O input instruction is normally expected in each instance. Refer to the section of this chapter about reading data for subsequent actions during the read cycle.
Load Data Modes Command (Op-code 101)

The mode bits (event, counter and interrupt) are loaded for all channels wherein each of four octal digits of this command, CW(11-0), specifies a channel's three mode bits; see Figure 5.3.

Event mode sets the conditions required to satisfy an event,

\[
0 = \text{event is set if both comparator A and B have true outputs} \\
\quad \text{event is reset otherwise}
\]

\[
1 = \text{event is set if comparator A is true} \\
\quad \text{event is reset if comparator B is true} \\
\quad \text{no change, otherwise.}
\]

Counter mode selects the source of pulses to count in an event counter,

\[
0 = \text{event pulses are counted; for measuring number of events} \\
1 = \text{time base periods are counted; for measuring event duration.}
\]

Interrupt mode determines the condition for computer I/O interruption by a channel's event counter value,

\[
0 = \text{interrupt for each count} \\
1 = \text{interrupt just before overflow (}2^{16}-1 = 65,535 \text{ counts).}
\]

Specific combinations of mode bits are useful in different measurement situations. For example, we could be investigating the use of a specific software subroutine (the program monitor would then be connected to the sampled computer's program counter). If we set limit A equal to the entry point address and limit B equal to the exit point address of the subroutine (say for data channel 0) and the comparator functions to ",=" , we would bound the subroutine. By choosing event mode =1 for channel
0, its event latch would be set upon subroutine entry and reset upon exit. Any software executed between the times of entry and exit would be included in the measurement (e.g. system interrupt routines). Correspondingly, counter mode =0 would count the number of machine instructions executed, or counter mode =1 would measure total time elapsed from entry to exit. Then, interrupt mode =1 would notify the 21XX minicomputer of a full measurement counter (for either number or time overflow), so that the software can read the counter after wraparound and also account for the number of overflows.

Stop Sampling Command (Op-code 110)

This command stops data sampling by inhibiting sampling strobe pulses. Sampling resumes when a start sampling command is issued. By stopping sampling, the programmer may hold off new measurements in order to make a synchronized record (snapshot) of all event counter values. The user is warned that the event latches are unaltered by sampling suspension and may continue to pass time base pulses to the event counters (if event mode = time and the event latch is set) even when new data is not coming in. Therefore, erroneous time measurements can result in certain circumstances. This problem can be corrected easily; see chapter 7 for future circuit changes.
Start Sampling Command (Op-code 111)

A one ("1") bit in CW(3-0) permits a computer interrupt to be produced from data collection channels 3-0, respectively. Unused channels can thereby be prevented from causing spurious interrupts by coding a "0" bit in the appropriate bit position. Such disabled channels will always show a zero bit in their respective interrupt status register bit position.

CW(9-6) indicate which channels operate in an "auto-clear" fashion. That is, for those data channels designated with a "1" auto-clear bit, a clear flag (CLF) instruction will clear their event latches, event counters, and interrupt latches. Auto-clear may be used by software or by the computer's direct-memory access (DMA) circuitry without program intervention (automatically since DMA sends a CLF after each word transferred), to clear chosen channels before and after collecting data.

The third function of this command is to start data sampling, and hence this command should be given as the last programming command before measurement begins.

Reading Data

The programmer follows a very simple two-step procedure to read event counter data or interrupt status from the program monitor. First, he selects the data source, using a "read select command" (placing the program monitor into "read mode") and second reads the data with an I/O input instruction (i.e. LIA, LIB, MIA, MIB). Without a preceding read select command, the read instruction will obtain data from the direct path. The read instruction will give data to the programmer in the formats shown in Figure 5.6. Note that event counter data is returned as 16-bit unsigned
integers. Each read instruction cancels the effect of a prior read select command; thereby ending the read cycle and reverting to the direct path automatically.

As mentioned in the section about read selection, a selected counter value is held indefinitely until a read occurs. The data obtained is therefore current as of the instant of selection (not reading). In contrast, the interrupt status register is not held fixed until read; therefore, the programmer obtains the status of all interrupting channels as of the instant of reading (a channel will not, of course, retract its interrupt until serviced). This implies that extra interrupt bits can turn on between the time of original interrupt and the instant that interrupt status is retrieved.

The direct path can be read at any time simply by issuing a read instruction without a preceding read select command. This permits DMA transfers into memory directly from the program monitor's direct path. Note that the load processor command selects the specific direct data path routing (either through the mask and adder or around both), so that the programmer (or DMA) can read the source data with or without masking, while the comparators always use the mask (and not the adder). It is the user's responsibility to guarantee that direct read operations can be processed between samples.

Set Control (STC)

A set control instruction permits the I/O card to interrupt the computer when conditions for interrupt on a data channel occur. Pending interrupts before the STC execution will occur immediately upon completion of the STC.
Clear Control (CLC)

A clear control instruction unconditionally prevents the program monitor from interrupting the computer. This instruction is useful within interrupt handling software to prevent external interruption.

Set Flag (STF)

This instruction may be used by software to trigger execution of the software interrupt handler for the program monitor (pseudo-interrupt). There is no effect upon data collection.

Clear Flag (CLF)

This instruction is used following an interrupt from the I/O card. It clears the program monitor's flag flip-flop to permit lower priority I/O devices to interrupt; and resets those event counters, event latches, and interrupt latches whose data channels are designated as auto-clear by the start sampling command. Interrupt latches for channels other than those that are auto-clear, are reset by a load data channel or read select programming command only. Each interrupt latch that is set (software can determine which ones by selecting and reading the interrupt status register) should be reset by one of the mentioned methods. If not reset, an interrupt latch will produce another computer interrupt when interrupts are reenabled (e.g. from an STC instruction).
DMA Operation

Direct memory access (DMA; also referred to as the dual-channel port controller, DCPC, on the 21MX series computers\(^{(20, 22)}\)) is an H-P 2000 series circuit capable of transferring data directly between memory and I/O devices at high data rates. Two separate and independent DMA channels are software assignable to any I/O slot where transfers are done in blocks of 16-bit words. Software initiates and completes the transfer, while the DMA hardware itself communicates directly with the I/O device's interface card to transfer each data word. Transfer directly to and from memory is accomplished by stealing (preempting) memory cycles from the CPU that normally runs program monitor software. Since DMA requires no software intervention during transfer, extremely high data rates, essentially at the memory's maximum speed, are possible. For example, transfer rates of up to one million words per second are possible with the H-P 2100A DMA hardware.

The program monitor is capable of utilizing DMA, especially for high-volume data collection applications. Programming is accomplished much as any other DMA type I/O device.\(^{(19)}\) When DMA is selected for the program monitor and initiated, DMA waits for an I/O interrupt from the program monitor I/O card. Upon an interrupt, DMA intercepts the interrupt and acquires a 16-bit data word from the program monitor by sending an I/O input (read) signal to the I/O card. Data on the program monitor's direct path (no prior read select command was issued) is given to DMA, that in turn writes the word into the computer's memory. DMA uses consecutive memory locations to store the acquired data, until a full block transfer
is completed. Both the first memory address for storing and the number of words to store are specified by the DMA initiation software. After each word that is transferred, DMA sends a clear flag (CLF) signal to the program monitor in order to clear the interrupt that initiated the transfer cycle. The CLF signal will have a clearing effect only on those data channels specified by the auto-clear register, as set by a start sampling command. Since an interrupting data channel that is not specified as auto-clear will continue to interrupt even after the CLF signal, it is advised that all channels be so specified when using DMA. Mixing of DMA with software interrupt processing for different data channels is not possible. Completion of the block transfer triggers execution of the DMA completion software, so that software gains control at transfer end.

Uses of DMA include tracing subroutine or individual instruction execution where the programmer desires to know the "thread" or path of execution through his software. The program monitor, when connected to the instruction register of a computer under measurement, can identify Jump Subroutine (JSB) instructions with its comparator circuits that look for the JSB op-code (the "AND" mask can isolate the op-code). If one is found, the program monitor can interrupt its own 21XX control computer whose DMA can pick up the entire JSB instruction from the direct path (by passing the mask), and store it away in its memory. Each JSB thus executed by the computer under study, is stored consecutively into the 21XX memory, thereby creating a list of JSB's as they occurred chronologically in the measured computer's dynamic execution. The programmer may then examine this compact list to glean information concerning the
dynamic action of his software. In this application, the program monitor serves to sift out only the information of interest, namely JSB instructions. Such a real-time reduction in data, saves many hours of sifting through volumes of data, albeit, probably with a computer program.

**DMI Operation**

Direct Memory Increment (DMI) is an extension of the DMA concept in that data collection between the 21XX control computer and the program monitor is still automatic and uses high-speed DMA transfers. But instead of simply storing the sampled data into the 21XX memory in a linear fashion, each data value is treated as a memory address, and the corresponding memory cell's value in the 21XX memory is incremented by one. Hence, depending upon the actual values of the sampled data, different profiles (histograms) of values (or value sets as mentioned in chapter 3) may be measured. Again as in DMA, memory is accessed directly without program intervention, except for initiation and completion of an entire block transfer. Each DMI event requires two memory accesses (one access to retrieve the previous value to be incremented, and a second access to store the updated value back into the memory cell), hence a maximum of only one-half million increments per second can occur on a 2100A computer.

If, for example, the 21XX control computer were measuring its own software activity with the program monitor tied to the program counter, the sampled data would correspond to specific cells in its own program memory. It would be disastrous to use DMI on these cells that contain machine instructions and increment the instruction; the entire program would soon be destroyed. To permit a computer to measure its own activity,
the adder circuit was appended to the program monitor's direct path. The adder increments the sampled data value by the constant loaded into the adder offset register. 16-bit unsigned addition is performed with the carry-bit being ignored. Also, since bit 15 is not used by DMI for memory addressing (i.e. indirect addressing is not permitted with DMI), wraparound addressing can effectively give a minus offset (subtraction) from the area being measured. The programmer can thereby set aside an area for data collection (incrementing), that is, separate and not overlapping the software being measured, by changing each sampled data value by a fixed amount. The H-P Data Input Card (Product No. 10674A\(^{(23)}\)) that is normally intended to work in conjunction with DMI, served as a model for the adder, mask, and limit detection circuits on the program monitor. Actual implementation of DMI is accomplished by using the H-P DMA/DMI interface kit (H-P Product No. 10673A\(^{(24)}\)) for H-P 2100A computers in conjunction with the program monitor cards. The DMA/DMI kit of two interface cards replaces the standard 2100A DMA board to offer DMI on the lower-priority DMA channel. A similar product does not presently exist for the 21MX computer series.
Clear
- 0 0 0

Initializes all circuit functions as per Figure 5.4.

Load Data Channel
- 0 0 1 1 1 1

B Func, A Func
Data Channel # (binary)
Limit Load Request
Loads Function registers A and B of specified channel.
If limit load request = 1,
a) Next output loads limit A
b) Second output loads limit B

Load Time Code
- 0 1 0

Time Divisor Code, see Figure 5.5
Resets the time base to zero and loads the time base divisor register.

Load Input Processor
- 0 1 1

21MX Register Code
000=P-reg
001=Memory Data
010=I-reg
011=(unused)
100=M-reg (reads)
101=M-reg (write)
110=M-reg (r&w)
111=(unused)

Input Source
0=21MX memory backplane
1=Probes

Direct Path Routing
0=Before Mask
1=After Mask and Adder

Mask and Offset Load Request

Selects the input source for sampling and the direct path routing.

If mask and offset load request = 1,
a) Next output loads the Mask
b) Second output loads the Offset

Figure 5.3 Programming Command Word Formats (continued)
Read Select

- 1 0 0 - - - - - - X X X

Select Code
000-011=Event counter 0-3
100 =Interrupt status register
101-111=(Unused); indeterminate data

Selects the specified source for software reading by an I/O input instruction, e.g.
LIA or LIB.

Load Data Modes

- 1 0 1 X X X X X X X X X

Chan.3 Chan.2 Chan.1 Chan.0

Similar for channels 1-3

Event Mode
0=A AND B
1=Start on B, Stop on A

Counter Mode
0=Count events
1=Count time

Interrupt Mode
0=Interrupt on each count
1=Interrupt on counter overflow

Loads modes for all four data collection channels.

Stop Sampling

- 1 1 0 - - - - - - - - -

Terminates data sampling; a sample in progress is finished before termination.

Start Sampling

- 1 1 1 - - X X X X X X - X X X X

Channel: 3 2 1 0 3 2 1 0

Auto-Clear Channels

Interrupts

Enabled

The specified channels are designated as auto-clear and enabled to cause computer interruption.

Data sampling is initiated.

Figure 5.3 Programming Command Word Formats
Input Processor

Source = external probes
Mask = all zeroes (0)
Offset = all zeroes (0)

Time Base
Period = one (1) microsecond

Counters*
Values = all zero (0)

Modes*
Event = A AND B
Counter = count events
Interrupt = interrupt on each count
Auto-Clear = no channels auto-clear

Interrupt Status*
Mask = all channels disabled
Latches = all cleared to zero (0)

Event Latches*
Values = all cleared to zero (0)

* Initialization occurs on all four data collection channels.

Note: Comparator limits and comparator functions are uninitialized and may be found in random states on power up.

Figure 5.4 Initial Circuit State
(On Power Up or Clear)
<table>
<thead>
<tr>
<th>Command Word (CW)</th>
<th>Time Base Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits: 3 2 1 0</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 microsecond</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>10 microseconds</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>100 microseconds</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1 millisecond</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>10 milliseconds</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>100 milliseconds</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>1 second</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>10 seconds</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>100 seconds</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 minute</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1 hour</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>10 minutes</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>infinite*</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>infinite*</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>20 milliseconds</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>infinite*</td>
</tr>
</tbody>
</table>

* Time base output constantly a logic zero.

Figure 5.5 Time Base Divisor Codes
Event Counter

Bit: 15
\[
\begin{array}{cccccccccc}
\text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} \\
\text{MSB} & \text{LSB}
\end{array}
\]

16-bit unsigned integer.

Interrupt Status

Bit: 15
\[
\begin{array}{cccccccccc}
- & - & - & - & - & - & - & \text{X} & \text{X} & \text{X} \\
\text{Channel: 3 2 1 0}
\end{array}
\]

Bit is a one (1) for interrupt on designated channel(s).

Note that more than one channel may interrupt at a time.

Direct Path

Bit: 15
\[
\begin{array}{cccccccccc}
\text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} & \text{X} \\
\text{MSB} & \text{LSB}
\end{array}
\]

16-bit word from the direct path.

Figure 5.6 Data Obtained by an I/O Read
CHAPTER 6

VERIFICATION TESTS AND APPLICATIONS

Purpose of the Tests

A number of tests were conducted to verify the expected compatibility of the program monitor with its 2100A and 21MX control computers. Tests of a computer sampling itself and sampling another CPU were done to investigate synchronized and unsynchronized measurement. Various intercard cable lengths were tested to guarantee reliable event detection regardless of the distance between the two program monitor circuit cards. At the same time, software was developed to greatly facilitate use of the program monitor in many measurement applications. Most importantly, the program monitor was placed into a known measurement environment to quantify the affinities (percentage of total CPU time, as defined in chapter 3) of pre-measured software routines. Verification of the results with premeasured figures yielded errors of less than 0.3%.

Single CPU Breakpoint

The program listed in Appendix B, Figure B.1, tests the ability of the program monitor to operate in a breakpoint function. That is, when the CPU enters a specified program section for execution, the program monitor detects this fact and interrupts the computer to prevent further execution. Therefore, the thread of execution is "broken" at the point of interest. Programmers recognize breakpoints as one of the more effective ways to debug software. Software sections of dubious functionality can be intercepted with a breakpoint, to permit the programmer to look at current program
results, and possibly step through the section, instruction by instruction. On many minicomputers, breakpoint is implemented with a software "breakpoint module" that modifies the software section to be intercepted, by substituting an instruction to trap execution at the point of interest. Obviously, the old substituted instruction must be saved somewhere and restored to its original location when execution is to proceed after the breakpoint occurs. All of this trickery comes about because most manufacturers do not provide a breakpoint capability in the hardware itself.

One of the capabilities of the program monitor is to act as a highly sophisticated hardware breakpoint, in the hardware configurations shown in Figures 6.1 and 6.2. The program monitor is plugged into the I/O backplane of an H-P 2100A computer and external probes connected to the P-register within the CPU (A5 circuit board). Each time the P-reg is updated by the CPU, the program monitor samples the updated value and tests it to satisfy the breakpoint ranges set into the program monitor's comparator circuitry. A successful test causes an interrupt to the CPU by the I/O card before the P-reg is updated again. If the test is unsuccessful, the CPU continues its normal execution flow; hence, the program monitor incurs no overhead until the instruction(s) to be broken are reached. The software to be interrupted is not altered at all; simply some additional software need be added to the operating system to handle the program monitor programming. A working example of such software is presented in Figure B.1, which was keyed into the computer's memory by hand for this test.

This software configures the program monitor to produce an interrupt whenever the P-reg has the value $100_8$, that is, whenever the wait area
Figure 6.1

H-P 2100A Self-Measurement Configuration
Figure 6.2
H-P 21MX Self-Measurement Configuration
loop is entered at location $100_8$. Upon such interruption, the interrupt processor gains control at location $10_8$. The address of the interrupted instruction is subsequently stored into location $11_8$, the interrupting data collection channel cleared, the event counter value displayed on the CPU front panel, and the CPU halted. It was found that (on a 2100A) the value at location $11_8$ differed from the expected value of $100_8$ by one count; that is, breakpoint was occurring one instruction late, before execution of location $101_8$. By investigation of the timing of 2100A I/O signals relative to CPU signals, it was determined that such a delay is an inherent problem with using the I/O backplane to cause breakpoint interruption. A similar study of the timing within a 21MX computer showed that the faster 21MX CPU relative to its I/O timing causes a longer delay of up to two (2) instructions, when trying to breakpoint a sequence of fast instructions. This drawback of delayed breakpoint on a 21MX is severe and could be remedied by direct CPU interruption without going through the I/O backplane.

**Sampling a 21MX Computer**

Most applications of the program monitor require remote sensing, such that the measurement system is separate from the test system. In the case of measuring an H-P 21MX computer, the FEC plugs directly into the 21MX memory backplane and the IOC into the I/O backplane of any 2000 series computer, as shown in Figure 6.3. This implies that the intercard card cable (ICC) needs to be long due to the distance between the two program monitor cards. It was initially feared that the additional cable length and consequent increased cable capacitance and inductance would
Figure 6.3
Measuring an H-P 21MX Computer
adversely affect control pulse signals carried by the ICC. In order to
test long ICC effects, 21MX memory backplane sampling with an ICC fifteen
(15) feet in length was performed as depicted in Figure 6.3. The 2100A
measurement computer and 2108 test computer were loaded with the programs
listed in Figures B.2 and B.3, respectively. Basically, the 2108 executed
a continuous loop of no-operation (NOP) instructions and accumulated the
number of loop iterations. The total was contained in a two-word loop
counter with the lower order sixteen bits in the A-reg, and the upper
sixteen bits (number of A-reg overflows) in memory location 30\(^8\). Upon each
A-reg overflow, about once every two seconds, the 2108 front panel display
rotated left one bit to give a visual indication of program activity. At
the same time, the program listed in Figure B.2 ran on the 2100A computer,
which used the program monitor to detect each time that the 2108 executed
location 5, P-reg = 5. For each such execution, the program monitor
interrupted the 2100A wait loop to run the software at locations 10\(^8\)
through 22\(^8\). This software incremented the B-reg, and upon overflow,
rotated the 2100A front panel display left one bit, analogous to the 2108's
display.

If the program monitor ever were to miss a 2108 loop iteration, then
the 2100A B-reg would lag behind the 2108 loop counter. Such was the case
when the test was first performed; the lag was measured as about two counts
lost every 10 seconds. As expected, the problem was signal distortion of
the program monitor ICC signals, in particular, signals EVA and EVB that
carry event data. Due to improper cable termination, multiple reflections
from the cable ends caused these signals to rise above 0.8v (the maximum
logic "0" voltage) which was an ambiguous value when strobed by ESO. The problem was soon remedied by a 220 ohm pullup resistor and a 1000 ohm pulldown resistor at the event data signals as they entered the IOC, to bring the signal reflections below 0.5v. After adding the resistors, another measurement yielded no counter lag after two hours of continuous running time. The program monitor was therefore accepted as operational with an ICC cable length of up to fifteen feet. Finally, further tests were conducted with this dual CPU arrangement to verify program monitor acquisition of other 21MX memory backplane signals such as I-reg, M-reg (read, write, read/write). It should be noted at this point that a specially strapped 2108 memory backplane was used to obtain P-reg strobing, since production model 21MX's did not have this capability built-in.

Real-Time Program Activity Display

The concept of isolated system measurement presented in the last section, is expanded in this section to a self-contained measurement system capable of producing a real-time graphical output. Figure 6.4 shows the measurement system attached to the system being measured (host system) which is driven with a well-defined stimulus source (TEPE) that will be described later.

The program monitor (IOC and FEC) acquires affinity and utilization parameters (see chapter 3 for definitions of these measures) of software sections within the host system. An H-P 2100A computer that drives the program monitor also serves to process the measurements and to convert them to analog form for display on an X-Y oscilloscope. Appendix C contains a listing of the measurement control program that is written in H-P Assembly
Figure 6.4 H-P 2000F/200

Controlled Stimulus/Response Measurement System
Language. Basically, a single program monitor data collection channel is used to collect statistics on P-reg values from the host system. Data collection is performed for a period $T_0$ on a designated range of P-reg values. When the time period elapses, sampling is momentarily suspended to read the event counter from the IOC and store the value into a data buffer. The comparator limits are then incremented to the adjacent range of P-reg values and sampling initiated again for the same time period $T_0$. In this manner, a "measurement window" is swept throughout all host system program areas to collect data about utilization or affinity as determined by the program monitor's event/time mode bit programmed by the measurement system software. In addition to the uniform moving measurement window, this program samples five (5) special host system program sections. Between each full memory sweep, these five ranges are swept for the same uniform time period $T_0$, to collect data about four large software areas and total software counts. The data buffer is thereby filled with counts like a histogram, each count corresponding to time duration (affinity) or number of instructions (utilization) for a known region of host system address space.

Time interval $T_0$ is implemented by using the time base generator (TBG) I/O interface card (26) to interrupt the measurement software at regular intervals. The reader should note that the program monitor itself could have served as a time base by using a second data channel, but only one of the four data channels was built when this measurement was made.

Displaying the data is fairly straightforward. A graphical output is produced on cartesian coordinates where $X =$ host system memory address,
Y = utilization or affinity for that memory area. As each count is obtained from the program monitor, it is converted into an X-Y coordinate pair and stored into adjacent halfwords in the data buffer. The Y-coordinate is essentially just the accumulated counts for that cell, and the X-coordinate is a scaled value corresponding to the measurement window location in the host system address space. Direct memory access (DMA) is used to feed the data buffer directly into the dual 8-bit D/A converter interface card that ties to the vertical and horizontal oscilloscope terminals. Coordinate axes are also displayed by simply putting constants into the data buffer that correspond to X-Y coordinate pairs. Measurement updating and histogram displaying are essentially independent asynchronous events and hence run as separate tasks with minimum interaction except via memory itself.

The stimulus was implemented with an H-P proprietary computer system called TEPE (for Time-Sharing Event Performance Evaluator) which can drive a time-sharing system and measure its real-time response as seen from a user's viewpoint. In effect, TEPE appears to the system under study (host system) as a group of users on communications terminals. TEPE can, therefore, present a desired user load to the host system in a manner that is repeatable from one measurement session to the next. This repeatability was used in this thesis to compare earlier results from a hardware monitor, i.e. Synectics SUM (described in chapter 2), with the results produced with the program monitor measurement system.

Affinity measurement of the H-P 2000F/200 I/O processor was thusly performed in this manner with the measurement configuration of Figure 6.4,
to produce the oscilloscope outputs in Figures 6.5a and 6.5b. Measurement
began fifteen (15) seconds after TEPE logged in all sixteen user lines, and
measurement proceeded for sixty (60) seconds total elapsed time. $T_0$, the
sample window time, was chosen to be four (4) milliseconds, and with a
total of 250 address ranges, yielding 60 scans through the host system
memory to generate the cumulative histograms of Figure 6.5. Both histogram
photographs are of the same affinity data, but with different vertical
scale factors to show how some data that hit the top of Figure 6.5a
compares with the comparatively low-level areas shown to scale in Figure
6.5b.

Following measurement termination, the cumulative counts corresponding
to the five special sampling ranges were obtained from the measurement
computer's memory buffer. These numbers were compared percentage-wise to
former measurements obtained under nearly identical conditions as generated
by TEPE and measured by the Synectics SUM equipment. The results are
displayed in Table 6.1 for comparison, which show measurement differences
of less than 0.3% worst-case, excellent agreement. Part of the differences
can be explained by the different measurement session lengths of 2.6
minutes using the SUM, compared to 1.0 minute for the program monitor.

Computer System Performance

The "program activity display" measurement system presented in the
previous section can be directly applied to measure computer parameters in
addition to software execution times. For example, connecting the program
monitor probes to a computer's memory address register (M-reg) produces a
Figure 6.5 Affinity Photographs of H-I-D 200°F and 100°F
<table>
<thead>
<tr>
<th>Software Area</th>
<th>Program Monitor</th>
<th>Synectics Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20,616</td>
<td>92.6</td>
</tr>
<tr>
<td>B</td>
<td>1,166</td>
<td>5.2</td>
</tr>
<tr>
<td>C</td>
<td>265</td>
<td>1.4</td>
</tr>
<tr>
<td>D</td>
<td>119</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 6.1 2000F/200 Affinity, Program Monitor vs. Synectics SUM
histogram of memory segment utilization. Instruction register (I-reg) sampling would yield histograms of instruction type usage, e.g. Load, Store, Add instruction usages, or execution times for individual instructions. Tying into the disk sector address register on a magnetic disk interface circuit can give a picture of dynamic disk sector utilization to track down thrashing problems or to optimize page replacement algorithms in a virtual memory system, possibly in a dynamic feedback mode.

Use of multiple program monitor cards within the same measurement system can measure two or more components of a single computer system for correlation of multiple events, e.g. identifying certain software sections that cause high disk utilization.

**Computer Software Debugging**

Some simple software built upon the "program monitor software system" subroutines in Appendix C can implement a sophisticated software debugging system. The program monitor can serve as a hardware sensor for an H-P minicomputer to look in on its own operation. Detection and notification via I/O interrupts of specific software sections implements a software breakpoint capability, as described in the first section of this chapter. The program monitor's four data channels, each with two limits, lets the programmer specify four separate ranges within his software to break on, e.g. four program subroutines. Note that no modification is made to the software to be interrupted; the program monitor simply probes the P-reg and awaits execution of the software of interest.
Similar breakpointing on protected memory areas, by sampling the memory address register, can give the programmer a handle on how his data is being unexpectedly modified. Obviously, privileged instructions can be implemented by using the program monitor, tied to the instruction register, to interrupt a user's program and pass control to the operating system when a privileged instruction is executed.

The programmer can trace his program execution by using the program monitor to sample the P-reg and selectively store instruction address sequences via DMA into memory. Such tracing is extremely useful in localizing a sequence of software instructions or subroutines that occur just prior to a fatal system crash.

**Microcode Performance**

The high measurement speed of the program monitor permits monitoring and performance measurement of sub-microsecond software such as microcode. Most of the techniques already presented for sampling the program memory of larger and slower minicomputers also apply to the analysis of microcoded software – simply a speed increase is the major constraint.

**Digital Circuit Measurement**

State changes, false states (glitches) and state dwell, affinity, and utilization are all measurable parameters for digital circuit sampling. For example, a first-in first-out (FIFO) or last-in first-out (LIFO, push-down stack) memory can be viewed as a digital system that has a number of distinct states, representable by the number of elements currently stored in the memory. Utilization of the memory space can be plotted by the
program activity display system which samples the output of an up-down counter; where up = add an element to the memory, down = take an element from the memory. The combination of the up-down counter with the program monitor is one example of how a simple hardware add-on can extend the capability of the program monitor greatly.

Another simple circuit appendage adds the capability to measure asynchronous state machines that don't have a built-in clock line. Essentially, the circuit emits a strobe pulse whenever it detects a state change. See chapter 7 for further details.

**Measuring Mechanical Apparatuses**

Many analog systems, both mechanical and electrical, can be converted for digital sensing via analog to digital (A/D) converters. Once converted to digital form, measurement with the program monitor is feasible. Such a converted system output can be modelled as an asynchronous state machine whose characteristics we have already considered in chapter 3. Again, an asynchronous front-end for the program monitor is required to derive a synchronous strobe; see chapter 7.

**Summary**

In summary then, the program monitor is a very flexible tool that can apply to a diversity of measurement situations, such as, computer software performance and debugging, digital and analog hardware measurement, and mechanical system characterization. The enormous flexibility of this instrument, despite its relatively simple fundamental architecture, is
attributable to the salient features of programmability, high sampling rate, compatibility with TTL digital systems, built-in timing reference, and multiple data collection channels.
CHAPTER 7
SUGGESTIONS FOR FUTURE WORK

Circuit Design Cleanup

Three known "bugs" exist in the present program monitor schematics that can adversely affect measurement results; these have been mentioned in preceding chapters and are detailed in this section.

The first problem is the most serious, and appears as abnormally high event counter values when doing time measurements (event/time mode bit for a data collection channel is set to "time"), that is, a measurement error exists for timing measurements. As stated in chapter 5, in the section describing the stop sampling command, a data channel's event latch may remain set when data sampling is halted. If so, time base pulses continue to be counted even though no new samples are being taken. The problem can be solved by feeding the "RO" pin of the MK5009P chip in Figure A.7 with the inverse of signal SE, rather than CC, both from Figure A.8. Time base pulses will thereby cease whenever sampling is stopped; and a full-length time base pulse will occur whenever sampling is restarted.

A second potential problem occurs due to improper PM-bus arbitration. Under circumstances where a programming command directly follows a read select command for interrupt status, without an intervening read instruction to acquire the status, both sources are gated to the PM-bus simultaneously. Modifying the enable signal into the interrupt status register of Figure A.5 to be TE4·WRIT will cure the problem. This will give programming commands higher priority of the bus than the status register.
As stated in chapter 5 in the section titled "read select command", an incomplete read cycle should be completed by a read instruction, e.g. LIA, before another output command is performed. Otherwise, counter data can be destroyed since the address latch contents are altered for each output command. The best fix is to prevent alteration of the address latch during an active read cycle. To do this, the enable line going into the address latch in Figure A.6 would be changed to STROB•TE from just STROB alone.

Circuit Simplification

A most obvious circuit simplification would be to replace the redundant \(<, >, =\) comparison function bits for both limits by simply \(<, =\) for limit A and \(>, =\) for limit B. No loss of comparison ranges and functions would result, and one 74170 function RAM IC could be eliminated while two 74S64's could be replaced by one 74S51; a net savings on two IC packages. Other circuit simplifications are left for future development since they go hand-in-hand with modifications of circuit capability.

Higher Performance

The channel cycle time can be diminished with the modified channel counter circuit in Figure 7.1 to update part of Figure A.2. In effect, CLOCK occurs two Cl clock pulses earlier (72 nanoseconds), and despite the added NAND gate to decode the 74161 outputs, replacement of standard-speed gates with Schottky gates gives a total net speed increase of 80 nsec. Percentagewise, the greatest speed increase occurs for continuous one channel cycling, as shown in Table 7.1
Figure 7.1 Modified Channel Counter Control Circuit and Timing Diagram
<table>
<thead>
<tr>
<th>Number of Cycling Channels</th>
<th>Channel Cycle Times</th>
<th>% Cycle Time Decrease From Table 4.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>190 nsec</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>330 nsec</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>470 nsec</td>
<td>--</td>
</tr>
<tr>
<td>4</td>
<td>620 nsec</td>
<td>13</td>
</tr>
<tr>
<td>Continuous 1 channel</td>
<td>66 nsec (max. counter speed)</td>
<td>55</td>
</tr>
</tbody>
</table>

Table 7.1 Modified Sampling Cycle Times
Since the program monitor can process samples faster than ten megahertz, (continuous sampling) the programmable time base should be able to run at 50 MHz so that accurate high-speed timing measurements can be made. To implement such a clock, the MOSTEK MK5009P must be bypassed at high speed since it can run only as fast as two megahertz. Some work in designing a compact high speed programmable time base is therefore suggested.

More Capability

In many cases, it would be very advantageous to measure asynchronous systems that do not have an internal clocking signal line. That is, we would like the program monitor to derive the clocking signal itself, just from the data probe sensors. This is very simply done by adding an equality comparator circuit to generate a strobe pulse whenever the present probe values differ from their previous values. As shown in Figure 7.2, the 74LS136 comparator is added to the 74298 multiplexor-latch chip already present in Figure A.1. In Figure A.2, the sample latch input signal is disconnected from the 74S153 and tied to the ST output of the new equality comparator. Moving this sample latch connection very simply selects synchronous or asynchronous sampling modes. The feedback loop through the sample latch of Figure A.2 back to the 74298 via signal EN guarantees that asynchronous sampling will occur only as fast as the program monitor's sampling cycle time; faster transitions are ignored.

Other improvements are suggested as follows:

1. Have breakpointing interrupts go directly into the CPU rather than over the I/O backplane. This would alleviate the delayed breakpoint problem described in chapter 6.
Figure 7.2 Asynchronous Front-End Circuit
2. Add a mode of operation wherein data channel event comparison signals can be cascaded from one channel to the next; "channel chaining". This would permit performance measurement of sequences of events, e.g. detection of dynamic sequences of machine instructions.

3. Have the event counters loadable under software control, effectively creating modulus-N counters where the user may select N.

4. Develop the capability for the program monitor to monitor itself, to do self-diagnosis and self-checkout.

5. Design a hardware front panel for the program monitor from which programming and display of measurement results can occur, without requiring an H-P control computer.

6. Incorporate high impedance (low capacitance and inductance) probes for better measurement isolation from the host system.

7. Develop a high-level user interface via software, for easy use of the program monitor in a wide variety of measurement applications.

8. Miniaturize the program monitor circuitry to a portable unit (handheld or desktop) that is easy to program and use, and applies to the same wide variety of applications as the prototype program monitor.

The program monitor is an extremely powerful instrument that is based upon a simple architecture that can be applied in its present form or developed further as one of the more powerful tools for digital systems measurement.
APPENDIX A

PROGRAM MONITOR CIRCUIT SCHEMATICS AND TIMING DIAGRAMS

This appendix contains circuit schematics and timing diagrams for the program monitor circuitry. The following conventions are used throughout this appendix:

1. All timing information is in units of nanoseconds unless otherwise specified.
2. Unterminated signals denote off-page connections to other figures in this appendix, or to standard Hewlett-Packard backplane signal wires.
3. Integrated circuits denoted by,
   a. 74xx are standard TTL series,
   b. 93xx are Fairchild Semiconductor TTL series,
   c. 8Txx are Signetics Semiconductor TTL series,
   d. 8xxx are National Semiconductor TTL series.
4. All monostable timing components are 1% metal film resistors and 5% silver mica capacitors. Otherwise, resistors are 5% molded carbon type.
5. The circuits are partitioned onto the program monitor I/O card (IOC) and Front End Card (FEC) by figures,
Figure A.2 Input Processor Control Circuit
Figure A.3 Comparator Circuit
Figure A.5 Interrupt Control Circuit
Figure 2-6 Event Initialization Circuit
Figure A.7 Programmable Time Base Circuit
Figure A.10  I/O Bus Interrupt Interface Circuit
Figure A.12 Control Input Timing Diagram

* NOTE: Holding Probe asserted causes continuous cycling.
Figure A.14 2100A, 21MX I/O Timing Diagram
Figure A.15 I/O Command Timing Diagram
APPENDIX B
SOFTWARE FOR PROGRAM MONITOR TESTING

The software listings in this appendix represent hand-assembled (as opposed to computer assembled) programs that were manually keyed into the computer's memory. Hence, no formal assembly language listings of these programs exist.
<table>
<thead>
<tr>
<th>Address (Octal)</th>
<th>Contents (Octal)</th>
<th>Symbolic Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>000011</td>
<td>OCT 11</td>
<td>Indirect pointer</td>
</tr>
<tr>
<td>10</td>
<td>114007</td>
<td>JSB 7,I</td>
<td>Entry point; save inst. address</td>
</tr>
<tr>
<td>11</td>
<td>000000</td>
<td>NOP</td>
<td>-) interrupted instruction</td>
</tr>
<tr>
<td>12</td>
<td>107710</td>
<td>CLC 10,C</td>
<td>Prevent more interrupts</td>
</tr>
<tr>
<td>13</td>
<td>060072</td>
<td>LDA 72</td>
<td>A:= Read select command</td>
</tr>
<tr>
<td>14</td>
<td>102610</td>
<td>OTA 10</td>
<td>Select counter &amp; clear channel</td>
</tr>
<tr>
<td>15</td>
<td>106510</td>
<td>LIB 10</td>
<td>B:= Counter value</td>
</tr>
<tr>
<td>16</td>
<td>106601</td>
<td>OTB 1</td>
<td>Display B on front panel</td>
</tr>
<tr>
<td>17</td>
<td>102077</td>
<td>HLT 77</td>
<td>Halt</td>
</tr>
<tr>
<td>20</td>
<td>024052</td>
<td>JMP 52</td>
<td>Continue</td>
</tr>
</tbody>
</table>

* Program Monitor Configurator

<table>
<thead>
<tr>
<th>Address (Octal)</th>
<th>Contents (Octal)</th>
<th>Symbolic Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>107710</td>
<td>CLC 10,C</td>
<td>Prevent interrupts</td>
</tr>
<tr>
<td>41</td>
<td>060056</td>
<td>LDA 56</td>
<td>A-) Command table</td>
</tr>
<tr>
<td>42</td>
<td>164000</td>
<td>LDB A,I</td>
<td>B:= Command</td>
</tr>
<tr>
<td>43</td>
<td>054057</td>
<td>CPB 57</td>
<td>Table end?</td>
</tr>
<tr>
<td>44</td>
<td>024050</td>
<td>JMP 50</td>
<td>Yes, branch</td>
</tr>
<tr>
<td>45</td>
<td>106610</td>
<td>OTB 10</td>
<td>Output command to board</td>
</tr>
<tr>
<td>46</td>
<td>002004</td>
<td>INA</td>
<td>A-) Next command</td>
</tr>
<tr>
<td>47</td>
<td>024042</td>
<td>JMP 42</td>
<td>Do next entry</td>
</tr>
<tr>
<td>50</td>
<td>006404</td>
<td>CLB,INB</td>
<td>B:= l</td>
</tr>
<tr>
<td>51</td>
<td>102100</td>
<td>STF 0</td>
<td>Enable interrupt system</td>
</tr>
<tr>
<td>52</td>
<td>102710</td>
<td>STC 10,C</td>
<td>Enable program monitor interrupts</td>
</tr>
<tr>
<td>53</td>
<td>024100</td>
<td>JMP 100</td>
<td>Branch to wait area</td>
</tr>
</tbody>
</table>

Figure B.1 CPU Breakpoint Test Program (continued)
<table>
<thead>
<tr>
<th>Address (Octal)</th>
<th>Contents (Octal)</th>
<th>Symbolic Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Command Table</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>000060</td>
<td></td>
<td>-) Command table</td>
</tr>
<tr>
<td>57</td>
<td>100000</td>
<td></td>
<td>Table end flag</td>
</tr>
<tr>
<td>60</td>
<td>000000</td>
<td></td>
<td>Clear</td>
</tr>
<tr>
<td>61</td>
<td>012207</td>
<td></td>
<td>Load channel 3; request limits</td>
</tr>
<tr>
<td>62</td>
<td>000100</td>
<td></td>
<td>Limit A</td>
</tr>
<tr>
<td>63</td>
<td>000100</td>
<td></td>
<td>Limit B</td>
</tr>
<tr>
<td>64</td>
<td>037005</td>
<td></td>
<td>Load input processor; request mask &amp; offset</td>
</tr>
<tr>
<td>65</td>
<td>077777</td>
<td></td>
<td>Mask (use bits 0-14)</td>
</tr>
<tr>
<td>66</td>
<td>000000</td>
<td></td>
<td>Offset (0)</td>
</tr>
<tr>
<td>67</td>
<td>050000</td>
<td></td>
<td>Modes; Interrupt each A AND B event</td>
</tr>
<tr>
<td>70</td>
<td>070010</td>
<td></td>
<td>Start sampling; channel 3 enabled</td>
</tr>
<tr>
<td>71</td>
<td>177777</td>
<td></td>
<td>Table end</td>
</tr>
<tr>
<td>72</td>
<td>040003</td>
<td></td>
<td>Read select</td>
</tr>
<tr>
<td>* Wait Area for Interrupt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>000000</td>
<td>NOP</td>
<td>Wait loop</td>
</tr>
<tr>
<td>101</td>
<td>000000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>000000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>000000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>000000</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>024100</td>
<td>JMP 100</td>
<td>Keep looping</td>
</tr>
<tr>
<td>* Execution Starts at Location 40_8; Program Monitor I/O Card in I/O slot 10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure B.1 CPU Breakpoint Test Program
<table>
<thead>
<tr>
<th>Address (Octal)</th>
<th>Contents (Octal)</th>
<th>Symbolic Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>0000005</td>
<td></td>
<td>Lower bound (limit A)</td>
</tr>
<tr>
<td>63</td>
<td>0000005</td>
<td></td>
<td>Upper bound (limit B)</td>
</tr>
<tr>
<td>64</td>
<td>030004</td>
<td></td>
<td>Input processor = 21MX P-reg</td>
</tr>
<tr>
<td>67</td>
<td>054000</td>
<td></td>
<td>Modes; A AND B event collection, interrupt on counter overflow</td>
</tr>
<tr>
<td>15</td>
<td>034001</td>
<td>ISZ 1</td>
<td>B := B+1</td>
</tr>
<tr>
<td>16</td>
<td>024052</td>
<td>JMP 52</td>
<td>Continue</td>
</tr>
<tr>
<td>17</td>
<td>102501</td>
<td>LIA 1</td>
<td>A := Switch register</td>
</tr>
<tr>
<td>20</td>
<td>001200</td>
<td>RAL</td>
<td>A left rotate</td>
</tr>
<tr>
<td>21</td>
<td>102601</td>
<td>OTA 1</td>
<td>Put back into display</td>
</tr>
<tr>
<td>22</td>
<td>024052</td>
<td>JMP 52</td>
<td>Continue after interrupt</td>
</tr>
</tbody>
</table>

* Basically, this program is the same as that listed in Figure B.1

* with the following modifications:

Figure B.2 2100A CPU Measurement Test Program
<table>
<thead>
<tr>
<th>Address (Octal)</th>
<th>Contents (Octal)</th>
<th>Symbolic Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>006404</td>
<td>CLB,INB</td>
<td>B := 1</td>
</tr>
<tr>
<td>3</td>
<td>074030</td>
<td>STB 30</td>
<td>Overflow counter := 1</td>
</tr>
<tr>
<td>4</td>
<td>002404</td>
<td>CLA,INA</td>
<td>A := 1</td>
</tr>
<tr>
<td>5</td>
<td>000000</td>
<td>NOP</td>
<td>Delay instruction</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>17</td>
<td>000000</td>
<td>NOP</td>
<td>Delay complete</td>
</tr>
<tr>
<td>20</td>
<td>034000</td>
<td>ISZ A</td>
<td>A := A+1; skip on overflow</td>
</tr>
<tr>
<td>21</td>
<td>024005</td>
<td>JMP 5</td>
<td>Loop</td>
</tr>
<tr>
<td>22</td>
<td>005200</td>
<td>RBL</td>
<td>B left rotate</td>
</tr>
<tr>
<td>23</td>
<td>106601</td>
<td>OTB 1</td>
<td>Display moving bit</td>
</tr>
<tr>
<td>24</td>
<td>000000</td>
<td>NOP</td>
<td>Patch location for halt</td>
</tr>
<tr>
<td>25</td>
<td>034030</td>
<td>ISZ 30</td>
<td>Bump totals counter</td>
</tr>
<tr>
<td>26</td>
<td>024004</td>
<td>JMP 4</td>
<td>Continue loop</td>
</tr>
<tr>
<td>30</td>
<td>000000</td>
<td>OCT 0</td>
<td>Overflow counter</td>
</tr>
</tbody>
</table>

* Execution starts at location 2, after the measurement computer is running.

**Figure B.3** 21MX CPU Measurement Test Program
APPENDIX C

PROGRAM ACTIVITY DISPLAY SOFTWARE
PAGE 0002 #31 PROGRAM ACTIVITY DISPLAY

ASMB,A,L,C,B

INTERRUPT TRAP CELLS

ORG 2B

ORG 102002 ERROR TRAP

ORG 102003 ERROR TRAP

ORG 102004 ERROR TRAP

ORG 102005 ERROR TRAP

CLC 6,C CLEAR DMA

ORG 102007 ERROR TRAP

JSR TRG,I -> TRG PROCESSOR

ORG 102010 FILLED BY PINIT

JSR REF,I -> DISPLAY PROCESSOR

ORG 1008

INITIALIZE

CLEAR CUMULATIVE BUFFER

LDA N251 AI=BUFFER LENGTH

LDB AKBUFF -> BUFFER

STA TEMP -> BUFFER

CLB B=0

STB TEMP,I CLEAR WORD IN BUFFER

ISZ TEMP -> NEXT WORD

ISZ A DECREMENT COUNT, DONE?

JMP INCLR NO, KEEP LOOPING

HLT 778 HALT

OFF CLC 0,C DISABLE ALL DEVICES

PROGRAM MONITOR CONFIGURATOR

JSB PINIT INITIALIZE P,M,

JSB PINPT CONFIGURE INPUTS

JSB PTIME SET P,M, TIME BASE

JSB PMON TURN ON PROGRAM MONITOR

LDA T8000 AI=TIME BASE
PAGE 0003 #01 PROGRAM ACTIVITY DISPLAY

0056 00126 132610 OTA 10B NOTIFY TRG
0057 00127 133710 STC 10B, C ENABLE TRG
0058 00139 103712 STC 12B, C ENABLE D-A
0059 00131 102100 STF 0 ENABLE INTERRUPT SYSTEM
PROGRAM ACTIVITY DISPLAY

WAIT LOOP - READ S-REGISTER AND SET OPTIONS

BIT 15 - STOP SAMPLING
BIT 14 - RESTART PROGRAM
BITS 3-8 - SAMPLING TIME IN MILLISECONDS

CHECK SAMPLING BIT

CHECK RESTART BIT

SFT SAMPLING TIME INTERVAL.
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode 1</th>
<th>Opcode 2</th>
<th>Opcode 3</th>
<th>Opcode 4</th>
<th>Opcode 5</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>09163</td>
<td>300000</td>
<td>PMON</td>
<td>NOP</td>
<td></td>
<td>ENTRY POINT</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>09164</td>
<td>360354</td>
<td>LDA</td>
<td>CCODE</td>
<td></td>
<td>A: CHANNEL CODE</td>
<td></td>
</tr>
<tr>
<td>0102</td>
<td>09165</td>
<td>315542</td>
<td>JSB</td>
<td>PCHAN</td>
<td></td>
<td>SET CHANNEL</td>
<td></td>
</tr>
<tr>
<td>0103</td>
<td>09166</td>
<td>330000</td>
<td>DB1</td>
<td>OCT</td>
<td>0</td>
<td>LOWER LIMIT</td>
<td></td>
</tr>
<tr>
<td>0104</td>
<td>09167</td>
<td>330000</td>
<td>DB2</td>
<td>OCT</td>
<td>0</td>
<td>UPPER LIMIT</td>
<td></td>
</tr>
<tr>
<td>0105</td>
<td>0170</td>
<td>315656</td>
<td>JSB</td>
<td>PSAMP</td>
<td></td>
<td>START SAMPLING</td>
<td></td>
</tr>
<tr>
<td>0106</td>
<td>0171</td>
<td>321410</td>
<td>OCT</td>
<td>001010</td>
<td></td>
<td>INTERRUPT MASK; AUTO CLR</td>
<td></td>
</tr>
<tr>
<td>0107</td>
<td>0172</td>
<td>320213</td>
<td>DEF</td>
<td>PRM</td>
<td></td>
<td>=&gt; INTERRUPT ROUTINE</td>
<td></td>
</tr>
<tr>
<td>0108</td>
<td>0173</td>
<td>124163</td>
<td>JMP</td>
<td>PMON,I</td>
<td></td>
<td>RETURN TO CALLER</td>
<td></td>
</tr>
</tbody>
</table>
PAGE 8006 #81 PROGRAM ACTIVITY DISPLAY

8111*
8112*
8113* REFRESH INTERRUPT

8114 03174 032175 REFR DEF ++1 ENTRY POINT
8115 00175 800300 NOP ENTRY POINT
8116 00176 020360 STA REGR SAVE A=REG
8117 00177 105706 CLC 6 STOP DMA
8118 00200 361004 LDA DMAC A1=DMA CONFIGURATION
8119 00201 102606 OTA 6 CONFIGURE DMA
8120 00202 136702 CLC 2 PREPARE DMA ADDRESS
8121 00203 801005 LDA DMAR A=DMA BUFFER
8122 00204 102602 OTA 2 NOTIFY DMA
8123 00205 102702 STC 2 PREPARE DMA WORD COUNT
8124 00206 361540 LDA DMAR A=### WORDS
8125 00207 132602 OTA 2 NOTIFY DMA
8126 00210 133706 STC 6,C ACTIVATE DMA
8127 00211 800360 LDA REGR RESTORE A=REG
8128 00212-124175 JMP REFR +1,1 RETURN FROM INTERRUPT

8130*
8131*
8132* PROGRAM MONITOR INTERRUPT

8133 00213 339000 PRM NOP ENTRY POINT
8134 00214 334370 IS7 OVFL FLAG AN OVERFLOW
8135 00215-124213 JMP -PRM,1 RETURN FROM INTERRUPT
0137\* TIME BASE GENERATION INTERRUPT

0138* DEF **+1 

0139* ENTRY

0140 00216 300217 TBG
0141 00217 300218 NOP

0142 00220 103110 CLF 0 

0143 00221 100719 CLC 108 

0144 00222 103440 DST REG1 SAVE A&B

0145 00224 34356 ISZ SCNT TIME INTERVAL EXPIRED?

0146 00225 224336 JMP TBGON NO, TURN ON TBG AGAIN

0147* READ P.M. COUNTERS AND COMPUTE Y/X COORD.

0148* 

0149* 

0150 00226 315675 JSB PSTOP STOP SAMPLING

0151 00227 315634 JSB PREAD READ COUNTERS & STATUS

0152 00230 309363 DEF CNTRS -> COUNTER AREA

0153 00231 364366 LDA CNT3 Ai=COUNT VALUF

0154 00232 364370 LDB OVFL Bi=OVERFLOW FLAG

0155 00233 309302 SZB SKIP IF NO OVERFLOW

0156 00234 364346 LDA N1 SET HIN TO MAX

0157* ACCUMULATE COUNT IN KBUF

0158 00235 32020 JSR COUNT OVERFLOW?

0159 00236 324225 JMP TROVR YES, JUMP

0160 00237 164767 LDB KPTR,I Bi=OLD COUNT

0161 00240 330920 SBB PREVIOUS OVERFLOW?

0162 00241 324225 JMP TBOVR YES, PROCESS OVERFLOW

0163 00242 103101 CLO CLEAR OVERFLOW BIT

0164 00243 34400 ADB A Bi=CUMULATIVE COUNT

0165 00244 322201 SOC SKIP IF COUNT OK

0166 00245 364346 TBOVR LDB N1 OVERFLOW; SET HIN TO MAX

0167 00246 174767 STB KPTR,I STORE NEW CUMULATIVE COUN

0168 00247 306501 LIB 1 Bi=SW=REG

0169 00248 334410 SLB CUMULATIVE DISPLAY?

0170 00250 160767 LDA KPTR,I YES, Ai=CUMULATIVE COUNT

0171* COMPUTE Y/X COORDINATES FOR POINT

0172 00252 36400 LDB A Bi=COUNT

0173 00253 369371 LDA YSCL Ai=SHIFT INSTRUCTION

0174 00254 382003 SZA,RS SCAFl=1 UNIT?

0175 00255 324261 JMP THG1 YES, SKIP SCALING

0176 00256 330373 IOR SLSR Ai=SHIFT INSTRUCTION

0177 00257 370260 STA **+1 STORE FOR EXECUTION

0178 00258 300900 NOP LSR * INSTRUCTION

0179 00261 336020 THG1 SSB SCREEN OVERFLOW?

0180 00262 324267 JMP TBGM YES, SET MAX

0181 00263 302400 CLA CLEAR A FOR SHIFT

0182 00264 344345 ADB .4 OFFSET FROM X-AXIS

0183 00265 101950 LSR 8 Bi=OVERFLOW, Ai=COUNT

0184 00266 306002 SSB OVERFLOW SCREEN?

0185 00267 360372 TBGM LDA YMAY YES, Ai=MAX Y-COORDINATE

0186 00270 331404 IOR DCTR Ai=Y/X COORDINATES

0187 00271 171806 STA DPMM,I STORE NEW BIN VALUE

0188* MOVE-SAMPLING WINDOW

0189* 

0190* 

0191* ISZ DPTR BUMP BIN POINTER

0192*
PAGE 008 #31 PROGRAM ACTIVITY DISPLAY

0192 00273 335404  ISZ DCTR  BUMP BIN COUNTER
0193 00274 334767  ISZ KPTH  BUMP COUNT POINTER
0194 00275 402400  CLA  CLEAR
0195 00276 670370  STA OVFL  OVERFLOW FLAG
0196 00277 864167  LDB DH2  B1=NEW LOWER BOUND
0197 00300 661404  LDA DCTR  A1=# BINS
0198 00301 640351  ADA N251  LEFT TO DISPLAY
0199 00302 240347  ADA N4  ACCOUNT FOR X-OFFSET
0200 00303 302003  SZA,RSS  NOT DONE YET
0201 00304 243421  JMP TBWRP  WRAPAROUND
0202 00335 661424  LDA DCTR  A1=#SCANNING BINS
0203 00306 240350  ADA N246  LEFT TO DO
0204 00307 240347  ADA N4  ACCOUNT FOR X-OFFSET
0205 00310 282280  SSB  SPECIAL BINS YET?
0206 00311 284330  JMP TBNXT  NO, SET NEXT BIN
0207  S1ES BINS
0208 00312 301000  ALS  A1=OFFSET INTO SHUF
0209 00313 841003  ADA ASHF  SPECIAL HOUNDS
0210 00314 104290  DLD A,1  A,B1=LOWER,UPPER BOUNDS
0211 00315 100000 101400  DSB  STORE SPECIAL HOUNDS
0212 00317 200166  324330  JMP TBG2  GO REPROGRAM BOUNDS
0213  WRAP AROUND DISPLAY
0214
0215
0216 00321 460345  TBWRP  LDA  4  A1#4
0217 08322 671404  STA DCTR  DCTR1=OFFSET FROM Y-AXIS
0218 00323 601005  LDA DMPR  DPTR->BUFFER
0219 00324 671006  STA DPIN  START
0220 00325 668770  LDA AKHUF  A->COUNT BUFFER
0221 00326 670767  STA KPTR  STORE NEW KPTR
0222 00327 1850010  LDB DHT1  B1=BOTTOM BOUND
0223  TBWNX STR DR1  DB1=NEW MEM, LOCATION
0224 00330 174166  TBWNX STR DR1  DB1=NEW MEMORY LOCATION
0225 00331 145907  ADB DSIZ  A1=NEW UPFP BOUND
0226 00332 174167  STR DR2  SET NEW DB2
0227 00333 141463  TBG2 JSB Programm Hounds & INTRR
0228
0229
0230
0231 00334 360355  LDA STIM  A1=TIME LOOP COUNT
0232 00335 376356  STA SCNT  SET COUNTER
0233 00336 360353  TBGON LDA TBGT  A1=TAG TIME BASE
0234 00337 102610  OTA 10B  OUTPUT TIME BASE
0235 00340 134200  DLD REGT  RESTORE REGISTERS
0236 00341 389361  10C  ENABLE TRG
0237 00343 123100  STF 0  ENABLE INTERRUPTS
0238 00344 124217  JMP TBR+1,I  RETURN FROM INTERRUPT

RAW TEXT END
PAGE 009 #01 PROGRAM ACTIVITY DISPLAY

CONSTANTS AND STORAGE AREAS

A EQU 0
B EQU 1

A-REGISTER
B-REGISTER

00345 200004 _4 DEC 4 NEGATIVE 1 CONSTANT
00346 177774 N1 DEC -1
00347 177774 N4 DEC -4
00350 177741 N246 DEC -246 # SCANNING BINS
00351 177740 N251 DEC -251 # TOTAL # BINS
00352 300077 SIXR OCT 77 MASK TO SAVE LOWER 6 BITS

00353 300001 IRGT OCT 1 TIME BASE (1 MILLISEC)
00354 001603 CODE OCT 091603 CHANNEL 3 CODE LWR<<SAMPLE<UPR
00355 039000 STIM BSS 1 SAMPLING TIME IN MILISECONDS
00356 0356 177777 SCNT OCT -1 CURRENT TIME COUNTER

TEMPORARY AREA

00357 300000 TEMP BSS 1 A SAVE AREA
00360 300000 PEGR BSS 1 ARB SAVE AREA

00361 300000 REGI BSS 2

00363 300000 CNTRS BSS 5 COUNTERS READ FROM P.M.
00366 CNTRS EQU CNTRS+3

00370 300000 OVLFL OCT 0 COUNTER OVERFLOW FLAG
00371 300000 YSCL OCT 3 POWER OF 2 VERTICAL SCALF
00372 300000 YMAG OCT 177400 MAX Y-COORDINATE
00373 101000 SLSR LSR 16 UNCONFIGURED LOG. SHIFT RIGH

00374 300000 KBUF BSS 251 CUMULATIVE COUNT BUFFER
00375 300000 KPRTR DEF KBUF ->CURRENT DISPLAYED COUNT
00376 300000 AKRUF DEF KBUF ->COUNT BUFFER

00377 300000 SHUF EQU * 5 SPECIAL SAMPLE RANGES
00378 300000 OCT 1611,1747 INTERCONNECT
00379 300000 OCT 1747,6154 TERMINAL I/O
00380 300000 OCT 6154,6426 TBC
00381 300000 OCT 6627,6675 IDLE LOOP
00382 300000 OCT 0,17777 TOTAL

00383 300000 ASHUF DEF SBUF ->SPECIAL SAMPLE RANGES
<table>
<thead>
<tr>
<th>Address</th>
<th>Offset</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0282</td>
<td>01004</td>
<td>000012 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0283</td>
<td>01005</td>
<td>00011 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0284</td>
<td>01006</td>
<td>00111 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0285</td>
<td>01007</td>
<td>00020 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0286</td>
<td>01008</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0287</td>
<td>01009</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0288</td>
<td>0100A</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0289</td>
<td>0100B</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0290</td>
<td>0100C</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0291</td>
<td>0100D</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0292</td>
<td>0100E</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0293</td>
<td>0100F</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0294</td>
<td>01010</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0295</td>
<td>01011</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0296</td>
<td>01012</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0297</td>
<td>01013</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0298</td>
<td>01014</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0299</td>
<td>01015</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0300</td>
<td>01016</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0301</td>
<td>01017</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0302</td>
<td>01018</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
<tr>
<td>0303</td>
<td>01019</td>
<td>00120 DMA PROG ACTIVITY DISPLAY GRAPHIC DISPLAY BUFFER</td>
</tr>
</tbody>
</table>

Y-AXIS EVERY 10 UNITS, EMPHASIZE EVERY 50

X-AXIS EVERY 10 UNITS, EMPHASIZE EVERY 100 (OCTAL)
| 01461 | 00062 |
| 01462 | 39072 |
| 01463 | 00102 |
| 01464 | 00082 |
| 01465 | 001102 |
| 01466 | 001112 |
| 01467 | 00122 |
| 01470 | 00132 |
| 01471 | 00142 |
| 01472 | 00152 |
| 01473 | 00162 |
| 01474 | 00172 |
| 01475 | 00182 |
| 01476 | 00502 |
| 01477 | 01202 |
| 01500 | 00212 |
| 01501 | 00222 |
| 01502 | 00232 |
| 01503 | 00242 |
| 01504 | 00252 |
| 01505 | 00262 |
| 01506 | 00272 |
| 01507 | 00302 |
| 01510 | 00702 |
| 01511 | 00702 |
| 01512 | 00702 |
| 01513 | 00702 |
| 01514 | 00702 |
| 01515 | 00702 |
| 01516 | 00702 |
| 01517 | 00702 |
| 01520 | 00702 |
| 01521 | 00702 |
| 01522 | 00702 |
| 01523 | 00702 |
| 01524 | 00702 |
| 01525 | 00702 |
| 01526 | 00702 |
| 01527 | 00702 |
| 01530 | 00702 |
| 01531 | 00702 |
| 01532 | 00702 |
| 01533 | 00702 |
| 01534 | 00702 |
| 01535 | 00702 |
| 01536 | 00702 |
| 01537 | 00702 |
| 01540 | 177251 |
| 01541 | 102007 |

**Program Activity Display**

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>01461</td>
<td>00062</td>
</tr>
<tr>
<td>01462</td>
<td>39072</td>
</tr>
<tr>
<td>01463</td>
<td>00102</td>
</tr>
<tr>
<td>01464</td>
<td>00082</td>
</tr>
<tr>
<td>01465</td>
<td>001102</td>
</tr>
<tr>
<td>01466</td>
<td>001112</td>
</tr>
<tr>
<td>01467</td>
<td>00122</td>
</tr>
<tr>
<td>01470</td>
<td>00132</td>
</tr>
<tr>
<td>01471</td>
<td>00142</td>
</tr>
<tr>
<td>01472</td>
<td>00152</td>
</tr>
<tr>
<td>01473</td>
<td>00162</td>
</tr>
<tr>
<td>01474</td>
<td>00172</td>
</tr>
<tr>
<td>01475</td>
<td>00182</td>
</tr>
<tr>
<td>01476</td>
<td>00502</td>
</tr>
<tr>
<td>01477</td>
<td>01202</td>
</tr>
<tr>
<td>01500</td>
<td>00212</td>
</tr>
<tr>
<td>01501</td>
<td>00222</td>
</tr>
<tr>
<td>01502</td>
<td>00232</td>
</tr>
<tr>
<td>01503</td>
<td>00242</td>
</tr>
<tr>
<td>01504</td>
<td>00252</td>
</tr>
<tr>
<td>01505</td>
<td>00262</td>
</tr>
<tr>
<td>01506</td>
<td>00272</td>
</tr>
<tr>
<td>01507</td>
<td>00302</td>
</tr>
<tr>
<td>01510</td>
<td>00702</td>
</tr>
<tr>
<td>01511</td>
<td>00702</td>
</tr>
<tr>
<td>01512</td>
<td>00702</td>
</tr>
<tr>
<td>01513</td>
<td>00702</td>
</tr>
<tr>
<td>01514</td>
<td>00702</td>
</tr>
<tr>
<td>01515</td>
<td>00702</td>
</tr>
<tr>
<td>01516</td>
<td>00702</td>
</tr>
<tr>
<td>01517</td>
<td>00702</td>
</tr>
<tr>
<td>01520</td>
<td>00702</td>
</tr>
<tr>
<td>01521</td>
<td>00702</td>
</tr>
<tr>
<td>01522</td>
<td>00702</td>
</tr>
<tr>
<td>01523</td>
<td>00702</td>
</tr>
<tr>
<td>01524</td>
<td>00702</td>
</tr>
<tr>
<td>01525</td>
<td>00702</td>
</tr>
<tr>
<td>01526</td>
<td>00702</td>
</tr>
<tr>
<td>01527</td>
<td>00702</td>
</tr>
<tr>
<td>01530</td>
<td>00702</td>
</tr>
<tr>
<td>01531</td>
<td>00702</td>
</tr>
<tr>
<td>01532</td>
<td>00702</td>
</tr>
<tr>
<td>01533</td>
<td>00702</td>
</tr>
<tr>
<td>01534</td>
<td>00702</td>
</tr>
<tr>
<td>01535</td>
<td>00702</td>
</tr>
<tr>
<td>01536</td>
<td>00702</td>
</tr>
<tr>
<td>01537</td>
<td>00702</td>
</tr>
</tbody>
</table>

**Octal Values**

- 112, 122, 132, 142, 152, 162, 172, 202, 262
- 1202, 212, 222, 232, 242, 252, 262, 272, 302
- 702, 1302, 312, 322, 332, 342, 352, 362, 372
- 373, 373, 1373, 374, 374, 1374, 375, 375
- 1375, 376, 376, 1376, 377, 377, 1377

**Additional Information**

- DMAN: ABS: DHUF:

**# Words In DBUF**

**Error Halt**
CALLING SEQUENCE: LDA CHAN-CODE A=CHANNEL-CODE

JSH PCHAN

DCT LOWER-LIMIT

DCT UPPER-LIMIT

PURPOSE: TO PROGRAM THE SPECIFIED CHANNEL.

WITH GIVEN FUNCTIONS AND LIMITS

CHANNEL-CODE: 0 000 000 00 00 000 000

UPR LWR CHAN

FUNCTIONS NUMR

RETURN ADDRESS

PC

A=CHANNEL COMMAND

OUTPUT COMMAND

OUTPUT LIMIT

NEXT PARAMETER

UPPER LIMIT

OUTPUT LIMIT

RETURN POINT

RETURN TO CALLER
PAGE 0013 #81  ---  PROGRAM MONITOR SOFTWARE SYSTEM  ---

0339-----------------------------
0340
0341*                        P-I-N-I-T  --  INITIALIZATION
0342*
0343*                        CALLING SEQUENCE: JSR PINIT
0344*                        OCT SC  --  P.M. SELECT CODE
0345*
0346*                        PURPOSE: INITIALIZE PROGRAM MONITOR FOR NEW SET OF
0347*                        MEASUREMENTS BY SUSPENDING DATA COLLECTION
0348*                        AND CLEARING COUNTERS, LATCHES, AND
0349*                        REGISTERS.
0350*
0351*                        NOTE: THIS SUBROUTINE MUST BE CALLED BEFORE ALL
0352*                        OTHER PERFORMANCE SUBROUTINES
0353*
0354   01554 200000 PINIT NOP RETURN ADDRESS
0355   01555 161564 LDA PINIT, I A:=SELECT CODE
0356   01556 335564 ISZ PINIT -->RETURN POINT
0357   01557 471713 STA PSC SAVE SELECT CODE
0358   01560 331797 IOR PCLC A:=CONFIGURED CLC,CLF INSTRUCTION
0359   01561 471615 STA PSCP STORE INSTRUCTION
0360   01562 471563 STA ++1 STORE AGAIN
0361   01563 330000 NOP CLC,CLF INSTRUCTION
0362   01564 361711 LDA POTA A:=UNCONFIGURED OTA INSTRUCTION
0363   01565 331713 IOR PSC CONFIGURE OTA
0364   01566 371632 STA PWRT STORE CONFIGURED INSTRUCTION
0365   01567 302400 CLA A:=CLEAR COMMAND
0366   01570 315631 JSR POUT OUTPUT COMMAND
0367   01571 660077 LDA JPINT A:=INT, JSR INST.
0368   01572 171713 STA PSC,I STORE IN TRAP CELL
0369   01573 125554 JMP PINIT,I RETURN TO CALLFR
P I-N-P-I — SET INPUT CIRCUITRY

CALLING SEQUENCE: JSB PINPT

OCT REG-CODE
OCT MASK 
OCT OFFSET DMI OFFSET

PURPOSE: TO SELECT THE DATA SOURCE UPON WHICH TO PERFORM SAMPLING, AND SET THE MASK AND OFFSET REGISTERS.

REG-CODE: 0 = P-REG (NEXT INSTRUCTION ADDRESS)
1 = T-REG (MEMORY CONTENTS)
2 = IR (NEXT INSTRUCTION TO EXECUTE)
4 = M-REG READS (MEMORY ADDRESS, READS)
5 = M-REG WRITES (MEMORY ADDRESS, WRITES)
6 = M-REG (MEMORY ADDRESS READS & WRITES)
1 = PROBES (TOP EDGE CONNECTOR)

PINPT 01574 000000
NOP 01575 161574
LDA PINPT,I A=REGISTER CODE
ISZ PINPT — — NEXT PARAMETER
ALF,ALF ADJUST REGISTER
RAL CODE TO BITS 9-11
SSA PROBE CODE?
CLA,INA YES, SET PROBE RTL
IOR PI A=INPUT SELECT COMMAND
JSB POUT OUTPUT COMMAND
LDA PINPT,I A=AND M MASK
JSB POUT OUTPUT MASK
ISZ PINPT — — NEXT PARAMETER
LDA PINPT,I A=DMI OFFSET
JSB POUT OUTPUT OFFSET
ISZ PINPT — — RETURN POINT
JMP PINPT,I RETURN TO CALLER

01574 01575 161574
01576 335574
01577 201727
01600 231200
01601 332620
01602 332404
01603 231716
01604 215631
01605 161574
01606 215631
01607 235574
01610 161574
01611 215631
01612 335574
01613 125574
PINTR - INTERRUPT SERVICE

CALLING SEQUENCE: NONE - INVOKED VIA TRAP CELL

SET BY PINIT

PURPOSE: TO INTERCEPT PROGRAM MONITOR INTERRUPTS AND JSB TO USER'S SERVICE ROUTINE.

```
0420 01614 600000 PINTR NOP  INT, RETURN ADDRESS
0421 01615 600000 PSTF NOP  CLC, CLF INST.
0422 01616 671724 STA PINIT  SAVE A-REG
0423 01617 661614 LDA PINTR  A->INT, RETURN
0424 01620 171726 STA PUIN1,T SET ADDR FOR USER RETURN
0425 01621 661724 LDA PUINIT RESTORF A-REG
0426 01622 125727 JMP PUIN41,I GO TO USER ROUTINE
```
PMODE -- SET PROGRAM MODES

CALLING SEQUENCE: JSR PMODE

PURPOSE: SET PROGRAM (EVENT, COUNTER, AND INTERRUPT) MODES

MODE CODE: 0 000 ICE ICE ICE ICE

CHANNELS: 3 2 1 0

INTERRUPT MODE: BITS 2, 5, 8, 11

0 = EACH COUNT

1 = OVERFLOW

COUNTER MODE: BITS 1, 4, 7, 10

0 = EVENTS

1 = TIME

EVENT MODE: BITS 0, 3, 6, 9

0 = LOWER 'AND' UPPER

1 = START/STOP

PMODE NOP

LDA PMODE, I

ISZ PMODE

IOR PM

JSR POUT

JMP PMODE, I
PAGE 017 #31 - - - PROGRAM MONITOR SOFTWARE SYSTEM - - -

0457#----------------------------------------------------------
0458#
0459#---------- P O U T ---------- OUTPUT command in A-reg TO PROGRAM MONITOR
0460#
0461 01631 030000 POUT NOP RETURN ADDRESS
0462 01632 030000 PWRT NOP CONFIGURED OTA INSTRUCTION
0463 01633 125631 JMP POUT,I RETURN TO CALLER

0465#----------------------------------------------------------
0466#
0467#---------- P R E A D ---------- COUNTER READ
0468#
0469#---------- CALLING SEQUENCE: JSR PREAD DEF TABLE-ADDR => 5-WORD TABLE
0470#
0471#---------- PURPOSE: TO READ THE 4 PROGRAM MONITOR COUNTERS AND
0472# STATUS REGISTER INTO THE 5-WORD AREA SPECIFIED BY TABLE-ADDR
0473#
0474#
0475# 0476 01634 200000 PREAD NOP RETURN ADDRESS
0477 01635 161634 LDA PREAD,I A->COUNTER AREA
0478 01636 035634 ISZ PRFAA =>RETURN ADDRESS
0479 01637 071730 STA PRPTR STORE TABLE ADDR
0480 01640 061712 LDA PLIB AI=UNCONFIGURED LIR INSTRUCTION
0481 01641 031713 IOR PSC CONFIGURE INSTRUCTION
0482 01642 071647 STA PRX STORE INSTRUCTION
0483 01643 061723 LDA N5 AI=5
0484 01644 071731 STA PRCT SET AS LOOP COUNTER
0485 01645 061717 LDA PR AI=READ COMMAND
0486 01646 015631 PRLOP JSB POUI SELECT REGISTER TO READ
0487 01647 030000 PRX NOP LIB STORED FROM ABOVE (H=COUNT
0488 01650 175730 STB PRPTR,I STORE COUNTER VALUE
0489 01651 035730 ISZ PRPTR BUMP TABLE POINTER
0490 01652 032004 INI AI=NEW COUNTER SELECT
0491 01653 035731 ISZ PRCT ALL COUNTERS READ?
0492 01654 025646 JMP PREAD,I YES, RETURN TO CALLER
0493 01655 125634
PAGE 0018 #81 — PROGRAM MONITOR SOFTWARE SYSTEM —

0495-----------------------------------------------
0496
--- 0497  P.S.A.M.P. BEGIN SAMPLING
0498
0499
0500
0501
0502
0503
0504
0505
0506
0507
0508
0509
0510 01656 300000 PSAMP NOP      RETURN ADDRESS
0511 01657 161656 LDA PSAMP,I AIM=INT/CLR MASK
0512 01664 335656 IS7 PSAMP  NEXT PARAMETER
0513 01664 331722 IOR PE AIM=INTERRUPT ENABLE COMMAND
0514 01664 315631 JSB POUT OUTPUT COMMAND
0515 01663 161656 LDA PSAMP,I A=INTERRUPT PROCESSOR
0516 01664 271726 STA PINT STORE INTERRUPT ADDRESS
0517 01665 202004 INA  A=INTERRUPT ENTRY
0518 01666 271727 STA PINT,I STORE ENTRY ADDRESS
0519 01667 261710 LDA PSTC AIM=UNCONFIGURED STC,C
0520 01669 231713 IOR PSC CONFIGURE STC,C
0521 01670 271672 STA ++ STORE INSTRUCTION
0522 01672 230000 NOP STC SC,C INSTRUCTION
0523 01673 335656 ISZ PSAMP  RETURN POINT
0524 01674-125656 JMP PSAMP,I RETURN TO CALLFR

0526
0527
--- 0528  P.S.T.O.P. STOP SAMPLING
0529
0530
0531
0532
0533
0534 01675 200000 PSTOP NOP      RETURN ADDRESS
0535 01676 261721 LDA PS AIM=STOP SAMPLING COMMAND
0536 01677 315631 JSB POUT OUTPUT COMMAND
0537 01700 125675 JMP PSTOP,I RETURN TO CALLFR
PAGE 0219 #21  --  PROGRAM MONITOR SOFTWARE SYSTEM  --

8541*  P T I M E  F  --  SET TIME BASE  
8542*  CALLING SEQUENCE: JSB PTIME 
8544*  OCT  TIME-CODE 
8546*  PURPOSE: SET TIME BASE DIVIDER INTERVAL 
8548*  TIME-CODE: 0 000 000 000 00T TIT 
8549*  0 = 1 MICROSECOND 
8550*  1 = 10 MICROSECOND 
8551*  2 = 100 MICROSECOND 
8552*  3 = 1 MILLISECOND 
8553*  4 = 10 MILLISECOND 
8554*  5 = 1 SECOND 
8555*  6 = 1 SECOND 
8556*  7 = 1 SECOND 
8557*  10 = 100 MILLISECONDS 
8558*  11 = 1 MINUTE 
8559*  12 = 1 HOUR 
8560*  13 = 10 MINUTES 
8561*  16 = 20 MILLISECONDS 
8562*  17 = 1 MICROSECOND 
8563*  81701 300000 PTIME NOF RETURN ADDRESS 
8565*  81702 161701 LDA PTIME, I = TIME CODE 
8566*  81703 335701 ISZ PTIME => RETURN POINT 
8567*  81704 331715 IOR PT A = TIME BASE COMMAND 
8568*  81705 315631 JSB POUT OUTPUT COMMAND 
8569*  81706 125701 JMP PTIME, I RETURN TO CALLFR
0571# 
0572# 
0573# C O N S T A N T S 
0574# 
0575# 
0576# S U B R O U T I N E S . 
0577# 
0578 01707  107700  PCLC  CLC  0,C  UNCONFIGURED C IC,CLF INST.
0579 01710  103700  PSTC  STC-0,C  UNCONFIGURED S TC,CLF INST.
0580 01711  132600  POTA  OTA  0  UNCONFIGURED OTA INSTRUCTION
0581 01712  106500  PLIB  LIB  0  UNCONFIGURED LIB INSTRUCTION
0582 01713  390000  PSC  BSS-1  PROGRAM MONITOR I/O SELECT CODE
0583# 
0584 01714  410004  PC  OCT  010004  CHANNEL COMMAND
0585 01715  320000  PT  OCT  020000  TIME BASE COMMAND
0586 01716  430004  PI  OCT  030004  INPUT COMMAND
0587 01717  440000  PR  OCT  040004  READ SELECT COMMAND
0588 01720  450000  PM  OCT  050004  MODES COMMAND
0589 01721  360000  PS  OCT  060000  STOP SAMPLING COMMAND
0590 01722  470000  PE  OCT  070000  ENABLE INTERRUPT COMMAND
0591 01723  177773  NS  DEC  -5  DECIMAL =5
0592 01724  200000  PINT  BSS 2  TEMP. AREA DURING INTERRUPT
0593 01726  330000  PINT  BSS 2  USER INTERRUPT ROUTINE ADDR
0594 01730  300000  PXTR  BSS 1  READ-TABLE ADDRESS
0595 01731  330000  PKC  BSS 1  LOOP COUNTER
0596 01732  201614  APINT  DEF  PINTR  -> P.M, INTERRUPT ROUTINE

0598 00077 ORG 77B
0599 00077 115732 JPINT JSB APINT,1 TRAP CELL INSTRUCTION
0600 END INIT

** NO ERRORS **
BIBLIOGRAPHY


