SOFTWARE FOR A MEDIUM-SCALE HYBRID COMPUTER

by

BARRY GEROME MICHAEL CRAWLEY
B.A.Sc., University of British Columbia, 1967

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in the Department of
Electrical Engineering

We accept this thesis as conforming to the
required standard

Members of the Department
of Electrical Engineering

THE UNIVERSITY OF BRITISH COLUMBIA

August, 1969
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and Study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical Engineering

The University of British Columbia
Vancouver 8, Canada

Date August 15, 1969
ABSTRACT

The combinations of analog and digital equipment to form hybrid installations are of such variety that hybrid software routines are generally restricted to particular hardware configurations. The software developed in this thesis is applicable and operational on the EAI PACE-231R-V; DEC PDP-9 Hybrid Computer.

The programs (in the form of subroutines and handlers) are structured, according to the characteristics of this computer, to handle the general situations of hybrid programming; such situations as the synchronization of the analog and digital computers, and the transfer of data and control information.

The present level of software development has considerably simplified the use of the interface for hybrid problem applications. Successive stages of software development would incorporate this stage, in whole or in part; each stage being more comprehensive and sophisticated than its predecessor.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF ILLUSTRATIONS</td>
<td>v</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENT</td>
<td>vii</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. HARDWARE FOR THE HYBRID COMPUTER</td>
<td>2</td>
</tr>
<tr>
<td>2.1 The Interface</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Changes in Logic Design</td>
<td>5</td>
</tr>
<tr>
<td>3. SOFTWARE FOR THE HYBRID COMPUTER</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Software Considerations</td>
<td>13</td>
</tr>
<tr>
<td>3.3 Digital Programming Aids</td>
<td>16</td>
</tr>
<tr>
<td>3.4 Hybrid I/O</td>
<td>19</td>
</tr>
<tr>
<td>3.4a A/D Conversion Routines</td>
<td>20</td>
</tr>
<tr>
<td>3.4b A/D Errors</td>
<td>26</td>
</tr>
<tr>
<td>3.4c D/A Conversion Routines</td>
<td>27</td>
</tr>
<tr>
<td>3.4d D/A Errors</td>
<td>29</td>
</tr>
<tr>
<td>3.4e The A/D and D/A Loop</td>
<td>31</td>
</tr>
<tr>
<td>3.4f Control of the Analog Computer from the Digital Side</td>
<td>36</td>
</tr>
<tr>
<td>3.4g Control of the Digital Computer from the Analog Side</td>
<td>50</td>
</tr>
<tr>
<td>3.5 Utility Library</td>
<td>52</td>
</tr>
<tr>
<td>3.5a Time Delay</td>
<td>52</td>
</tr>
<tr>
<td>3.5b Function Generation</td>
<td>58</td>
</tr>
<tr>
<td>3.6 MAN-MACHINE Interface Routines</td>
<td>62</td>
</tr>
<tr>
<td>3.6a Light-pen Handler</td>
<td>62</td>
</tr>
<tr>
<td>3.6b Visual-Aid Monitor (Hybrid Monitor)</td>
<td>64</td>
</tr>
<tr>
<td>3.6c Error Diagnostic Routines</td>
<td>68</td>
</tr>
</tbody>
</table>
4. SUMMARY AND CONCLUSIONS

APPENDIX I  PDP-9 INSTRUCTION SET

APPENDIX II  PATCH-PANEL CONNECTIONS FOR HYBRID COMPUTER UTILIZATION

APPENDIX III  FLOW-CHART FOR FUNCTION GENERATION (Scope and Teletype)

REFERENCES
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Block Diagram of a hybrid system</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Organization of the System Hardware</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Logic diagram for signal selector function</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Mode control logic block diagram</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>SFC Interrupt detector</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>Hybrid software package outline</td>
<td>16</td>
</tr>
<tr>
<td>7a,b,c,d,e,f,g</td>
<td>ADC handler and Sampling-Frequency-Control software flow diagram</td>
<td>23,24</td>
</tr>
<tr>
<td>8</td>
<td>Flow diagram of D/A setting routines</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>Information-flow out of the PDP-9</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>Simultaneous-Update code configuration</td>
<td>31</td>
</tr>
<tr>
<td>11a</td>
<td>A/D-D/A delay as a function of input-output pairs</td>
<td>33</td>
</tr>
<tr>
<td>11b</td>
<td>Scope measurement of A/D software sampling time</td>
<td>34</td>
</tr>
<tr>
<td>12</td>
<td>'Block' diagrams for Sensitivity analysis</td>
<td>35</td>
</tr>
<tr>
<td>13</td>
<td>Comparison of hybrid computer solutions with actual solutions</td>
<td>36</td>
</tr>
<tr>
<td>14</td>
<td>Mode selection software flow diagram</td>
<td>36</td>
</tr>
<tr>
<td>15</td>
<td>Integrator-mode selection software flow-chart</td>
<td>39</td>
</tr>
<tr>
<td>16</td>
<td>DAS facility software flow-chart</td>
<td>41</td>
</tr>
<tr>
<td>17</td>
<td>Signal selector software flow-chart</td>
<td>42</td>
</tr>
<tr>
<td>18</td>
<td>EDVM sampling routines</td>
<td>44</td>
</tr>
<tr>
<td>19</td>
<td>Servo-set potentiometer routines</td>
<td>45</td>
</tr>
<tr>
<td>20</td>
<td>Potentiometer tolerance set-up routine</td>
<td>46</td>
</tr>
<tr>
<td>21</td>
<td>Teletype copy of a coefficient buffer</td>
<td>47</td>
</tr>
<tr>
<td>22a,b</td>
<td>Flow diagram of DATA and BLOCK routines</td>
<td>49</td>
</tr>
<tr>
<td>23</td>
<td>Comparator-interrupt setup routines</td>
<td>52</td>
</tr>
<tr>
<td>24</td>
<td>Input-Output of time delay facility</td>
<td>53</td>
</tr>
<tr>
<td>25</td>
<td>.DELAY software flow-chart</td>
<td>57</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scope; light-pen input media for 2 dimensions</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>Scope; light-pen input media for 1 dimension</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>Teletype Interrogation for data</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>FUNCTN software flow-chart</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>Light-pen handler flow chart</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>Hybrid-monitor screen display</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>'MODES' section</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>'DDT' section</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>'HYBRID-MONITOR' software flow-chart</td>
<td>69</td>
<td></td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENT

Acknowledgement is gratefully given to the National Research Council of Canada for the financial support provided with a bursary in 1967-68 and a scholarship in 1968-69.

I would like to express thanks and appreciation to Dr. J.S. MacDonald for his encouragement and supervision afforded me as one of his graduate students. The experience and knowledge gained, as a result of this project's work and through time spent in lectures taught by him, has been of great benefit to me.

The co-reader, Dr. E.V. Bohn, was both generous and helpful in his comments and suggestions. The secretary in the Electrical Engineering office who laboured at typing the manuscript, fellow graduate students who devoted time to proof-reading, and the Department's photographer, Mr. H.H. Black, who took special time and care in his work, all deserve sincere appreciation for their efforts.

I wish to express special appreciation to my wife for her assistance and patience during the project, and for her help in preparing the manuscript.
1. INTRODUCTION

Individually, the digital and analog computers of a balanced hybrid system could solve a wide variety of complex problems. By interconnecting them through an interface, a more powerful computer system is realized.

Of the several motivations for mating digital and analog computers, the speed of an analog computer combined with the accuracy and memory of a digital machine might appear to be one of the most significant. In addition various tasks prove either cumbersome or difficult when executed completely on one machine or the other. On the analog, tasks such as multi-variable function generation and time delay realizations become awkward. The simulation of flight control systems in which a digital computer forms an element of the system is almost impossible using only conventional analog techniques. On the other hand, an analog subroutine may be employed to solve simultaneous differential equations or invert matrices as required in the digital solution of partial differential equations.

The hardware that matches the characteristics of the two computers is generally termed an interface. Two general classes of information flow bi-directionally across the interface to be utilized in either machine; (a) control information and (b) data. It will be the purpose of this thesis to organize and control this information flow, on a software basis, to present this software on an elevated and understandable level, and to keep the software flexible enough to accommodate future system-hardware expansions and changes.
2. HARDWARE FOR THE HYBRID COMPUTER

2.1 The Interface

The installation is comprised of the EAE PACE 231-RV analog simulator and the DEC PDP-9 digital computer. The interface is a 2-way multi-purpose I/O peripheral which allows digital information to be strobed into and out of the digital computer through its data buffers from the outside world; the outside world for this particular application being the analog computer. The interface (a) permits the digital computer to steer, guide, process data from, check and control the analog machine and (b) allows the analog simulator to interrupt or initiate action in the digital computer; all in as flexible a manner as is required by various hybrid problems.

The interface was designed by a previous graduate student.\(^{(1)}\) For clarity, the hardware capabilities and block
(a) The interface has a multiplexer (MUX) which will permit information from 16 unique analog lines to be read into the accumulator of the PDP-9 through a twelve bit A/D converter. There are sample and hold units on each of the input channels.

(b) A sampling control for the A/D converter allows any of the 16 MUX inputs to be continuously sampled at periodic intervals. Data can be read into the digital accumulator on completion of each conversion.

(c) Four D/A converters (DACs) are available; two 12 bit and two 9 bit. One 12 bit DAC is reserved for the 'hybrid' system, the remaining three are available for the user's hybrid problem.

(d) Individual control of up to 17 integrators can be achieved for repetitive-operation (REP-OP) of the PACE. Each integrator can be in either Operate or Initial Condition mode, and switching can be done either electronically or by relays. By slaving more than one integrator to some of the control lines, all 24 integrators of the PACE can be thus controlled.

(e) Facility exists for up to ten comparator outputs to set a 'flag' and cause a program interrupt on the PDP-9.
Fig. 2 Organization of the system hardware(1)
(f) The digital attenuator system (DAS) may be controlled for purposes of setting, reading and checking the Q (Servo-Set) potentiometers.

(g) Any one of 540 points on the PACE patch panel may be interrogated and read into the digital accumulator. This facility is used in problem setup and in setting and checking the servo-set potentiometers.

(h) The entire analog simulator can be put into one of six modes of operation; Hold, Initial Condition, Pot Set, Operate, Static Test and Rate Test.

(i) Data flow between the digital computer and the interface is double buffered, effecting an 'hold' element and permitting a simultaneous update of several output registers at the same time.

During the construction of the hardware, several logic-design changes were implemented.

2.2 Changes in Logic Design

Signal Selector Address Control

The Signal Selector Address Control facility is used in interrogating the 540-point analog patch panel (Slow Speed Multiplexer). The 3 independent banks of mechanical stepping switches increment until coincidence is sensed between the digital input address and the actual switch address. Previous design took the 18 bit address code (3 six-bit ASCII characters), coded it into a 3 digit decimal radix number and then compared ten lines for each bank to sense coincidence.
Since only 4 bits of each six-bit ASCII character are required to uniquely distinguish ten numbers, redesign centered around the approach of realizing the function:

\[ F = x_0y_1y_2y_3y_4 + x_1\bar{y}_1y_2y_3y_4 + x_2\bar{y}_1y_2y_3y_4 + x_3\bar{y}_1y_2y_3y_4 + \]
\[ x_4\bar{y}_1y_2y_3y_4 + x_5\bar{y}_1y_2y_3y_4 + x_6\bar{y}_1y_2y_3y_4 + x_7\bar{y}_1y_2y_3y_4 + \]
\[ x_8\bar{y}_1y_2y_3y_4 + x_9\bar{y}_1y_2y_3y_4 \]

where \( x \) are the analog input lines and \( y \) are the input address bits.

By using Shannon's Expansion Theorem and expanding in the variables \( y_1, y_2, y_3, y_4 \) respectively, a much simpler logic realization of the control network resulted (Fig. 3).

---

**Fig. 3** Logic diagram for Signal Selector function
The 100's (letter) unit network has a slight variation in that the bits taken from the 6 bit digital word are different: \[ XXX \] are the chosen bits.

Mode Selecting and Sensing Control

As can be seen from Fig. 2, the present mode of the analog machine is sensed by reading back into the PDP-9 the contents of the MODE Control register. The same 'memory' effect can be accomplished by software. It would appear more desirable to read the mode from the MODE BUS bar of the analog and in this way serve two purposes.

Because of relay switching, digital hybrid programs must account for the 'dead' time between state changes. Since the switching time will undoubtedly vary from time to time and relay to relay, some means of detecting the 'switching-

Fig. 4 Mode Control Logic Block Diagram
complete' status would be desirable. This is realized by 'comparing' the state of the MODE BUS with the digital input. Switching is complete when coincidence exists, and a mode can be sensed by 'rapidly' scanning for such a coincidence (the relays are slow compared to the digital computer).

Hence, a completely new logic circuit was incorporated for the mode control (Fig. 4).

'Sampling Frequency Control' Ready Status

The SFC, as it exists at present, issues pulses, at programmable time intervals, to initiate 'start-conversions' for the A/D converter. The MUX channel that is sampled is determined by the contents of the MUX address buffer at the time of the pulse. Should the contents of this buffer be changed, between pulses, to carry out a programmed 'random'-sample, there exists the danger of having an incorrect address in the MUX address buffer. Freedom to 'random'-sample at any time while the SFC is enabled necessitates an 'a-priori' knowledge of the arrival of the start-conversion pulse.

The SFC, by issuing just one pulse, initiates software sampling action on pre-specified channels. On completion, the software must initialize for the next cycle. Whenever the SFC is 'free-running', every random-sample will require the software to:

(1) check to see if enough time exists for such a sample without potential interference to SFC action, and

(2) restore addresses and buffers upon completion,
in anticipation of the next start conversion from the SFC.

Should the default of (1) exist then:

(3) this random-channel can be appended to the pre-specified SFC channels temporarily, or

(4) a future attempt can be made when the A/D converter is momentarily available.

The default of condition (1) is obtained from the remaining SFC interval count. \( \text{Min } t_g \) \( (t_g = \text{time to go}) \) should be such that the actions of (1) and (2) can be successfully completed, since together they establish a bound on a required remaining clock count. \( (t_g \text{ of 64 usecs. was chosen}) \). \( t_g \) restricts the interval of the SFC to be \( \geq t_g \) usecs. if random-sampling will at anytime be attempted while the SFC is enabled.

Upon coincidence of \( t_g \) and the SFC interval counter, an R-S flip-flop is set (see Fig. 5). Prior to issuing the

![Fig. 5 SFC interrupt detector](image-url)
start conversion for the intermediate channel, the software will check for the 'flag-set' condition and take the appropriate action as previously outlined.

In this way the SFC maintains top priority on the A/D converter.
3. SOFTWARE FOR THE HYBRID COMPUTER

3.1 Introduction

The PDP-9 is supplied with an extensive software package to aid in the creation and execution of digital computer programs. A library consisting of FORTRAN routines and extensive Input/Output handlers, and capable of being updated with user subroutines, foregoes repetitious, burdensome programming for various users. By appending the hybrid software to the existing powerful digital programs as subroutines, all the features of the digital computer in its 'stand-alone' mode become incorporated in the hybrid mode.

With sophisticated program links, extensive error checkouts, and with software on a semi-elevated level, a programmer can operate the digital machine with great efficiency. However, programming is not such that one becomes isolated from the hardware of the hybrid computer. An analog computer programmer has never been able to isolate himself from the hardware and the details of its operation. By being 'on-line' with his problem he gains greater insight into the dynamic behaviour of the system being studied and the solutions have a more significant meaning. With the bulk of modern computers being 'batch' processors, a digital computer programmer is removed from the computer itself, is relatively unconcerned with the mechanics of operation, and usually communicates with the machine with a high-level language such as FORTRAN. His ability to make dynamic program changes
on the basis of solution results is severely restricted and problem insight is gained perhaps only after several runs interspersed by turn-around time.

With the two machines mated, both types of programmers can expect to undergo changes in philosophy. Digital programmers would have to acquaint themselves with the conversational ability of the computer and become knowledgeable in the electrical simulation of mathematical problems. Analog programmers, inbred with the ability to interact freely with a program, should expect some loss in flexibility and certain procedures to become quite formalized.

For the operation and utilization of the hybrid installation, it will be necessary to:

(a) be familiar with the basic structure of the PDP-9. The input/output provisions coupled with the peripheral equipment encompasses an important section of the system's configuration. A knowledge of the interrupt facility, memory configuration, and the machine language are all essential to the operation of the computer. (3)

(b) understand the operation of the PACE 231-RV analog computer, and be familiar with all the facilities available, and to be capable of 'simulating' a problem. (4)

(c) have a general understanding of how the interface operates to accomplish the various
control functions and data transfers. Such a knowledge would be extremely useful for adapting the interface for 'special' needs. (1)

3.2 Software Considerations

The degree of variability in hybrid installations prohibits the formulation of an all-purpose, unique software package. The philosophy of software development for this project is (a) to utilize the interface in all aspects while keeping hardware applications flexible and general but not problem-oriented (b) to minimize the complexity involved in using the software and (c) to present algorithms and ideas that might be helpful in developing software for some other hybrid system.

Both analog and digital computers should be capable of instigating data transfers in either of the two directions in the hybrid loop depicted in Fig. 1. The transfer direction and the problem independency of the data loop are best emphasized and achieved by a READ-WRITE philosophy in the software. Real-time computation (accepting data, doing digital computation and responding with an answer within a specified time interval) demands efficient software to be able to close the loop with a minimal time delay. The efficiency and speed of the READ-WRITE routines should be predominantly masked by the digital computation time of most real-time applications.

This data flow and its synchronization with the two computers becomes the task of the interface and the
hybrid software respectively. Given an interface with certain capabilities (page 2) the software requirements are paramount in optimizing the hardware unit with respect to the computers at either end. The hybrid programs must match the sequential, time-dependent characteristics of the digital computer with the contrasting parallel properties of the analog machine.

A major software requirement is reflected in the need for analog control routines. With the vast logic capabilities of the digital computer, and with the link provided by the interface, the automated operation of the analog computer can be easily programmed. These control routines include such tasks as setting parameter coefficients with potentiometers and DACs, selecting an operating mode for the analog simulator, and controlling the integrators individually.

The successful operation of system hardware requires exhaustive hardware check-out routines. Certain hardware functions are inherently checked, by simple tests, each time the facility is used. Others require special time-consuming routines. For example, in setting a servo-set potentiometer, a 'flag' is set on completion. Failure of this set condition is detected by the software and is flagged as an error. Several such hardware failures are detected at run-time in a similar way. On the other hand, the check-out of amplifier drift, multiplier accuracy, integrator rate-checks, etc., would best be confined to a separate software package that could be left running unattended while results were being
recorded on some hard-copy device. Interface hardware analysis would normally also be confined to the same package, although certain malfunctions may be detectable during run-time as well. In the long run, execution time is more efficiently used when the detection of system malfunctions of various types are given preference over efforts to making the routines 'fool-proof'.

A class of hybrid software routines can and rightfully should be devoted to problem algorithms and function subroutines. Such an 'utility' library could be appended to as problem types demand and could be expanded as the system expands to allow more complex problems and algorithms.

As to the hybrid software, one must give some attention and consideration to the 'stand-alone' digital software since the majority of it functions with, or is useful to, the former. Such programs as the DEC EDITOR and MACRO-9 Assembler still remain prime of importance in program preparation. More important still is the influence of the digital computer's software and hardware characteristics in molding a hybrid-package format.

And last but not least, one must attain in the software, the ability for the 'problem-analyst' to interact freely with the hybrid problem. The highly interactive nature of hybrid computation requires that operator interrogation of programs be possible during run-time. The normal digital monitor system must be oriented in a fashion such as to allow access to direct on-line control of program action at any time. The routines would in effect interface the analyst to his problem.
In light of all these software requirements, the presentation of the hybrid package will involve two parts: the discussion and development of some basic program building blocks and then the creation of some useful hybrid routines from these fundamental ones. Fig. 6 depicts the over-all package.

Fig. 6 Hybrid Software Package Outline

3.3 Digital Programming Aids

This section of the package was supplied by the manufacturer of the digital computer. In lieu of the numerous volumes of publications for the utilization of their routines,
it will suffice here to briefly describe the function of the programming aids.

PDP-9 ADVANCED software provides a complete system for program preparation, compilation, assembly, debugging and operation. System programs include FORTRAN IV, a sophisticated macro assembler, an on-line debugging system, an on-line editor, and a peripheral interchange program. A versatile and flexible I/O programming system frees the user from the need to create device-handling subroutines and from the concerns of device timing.

The PDP-9 ADVANCED software monitor allows overlapped I/O and computation, simultaneous operation of a number of asynchronous peripheral devices, and device-independent programming. The MONITOR operates in conjunction with the I/O programs and provides a complete interface between the user's programs and the standard peripheral hardware.

Various features of the digital hardware and the 'stand-alone' digital programs are exploited when the hybrid programs are executed with the aid of the ADVANCED MONITOR'. The advantages of creating a 'Hybrid Monitor' and appending it to the existing one become increasingly important in attaining a high degree of 'man-machine' inter-action.

MACRO Assembler

This assembler processes the machine language coding peculiar to the PDP-9 and generates binary output in several formats; (1) relocatable, (2) absolute binary or (3) full binary (hardware READIN mode). The MACRO facility is one of
the more prominent features of the assembler and is used
to setup Input/Output calling sequences (.WRITE, .READ, etc.).
This feature plays a significant role in the HYBRID software
and will be used extensively.

When a program is being written, it often happens
that certain coding sequences are repeated several times with
only the arguments changed. The repeated sequence can be
generated by a single statement. It is first necessary, how­
ever, to define the coding sequence with dummy arguments as
a MACRO instruction and then use a single statement referring
to the MACRO name, along with a list of real arguments which
will replace the dummy arguments and generate the desired
sequence.

For the hybrid package, a tape consisting of nothing
more than definitions of the form [Subroutine name; ARGUMENTS;
Links to the LOADER] will be assembled prior to all others
such that a particular CALL expands into an appropriate
subroutine reference plus arguments. This type of format
was thought to be the most useful.

For example:

/Definition Tape

[.DEFIN POTSET, ARG1, ARG2
.GLOBL POTSTE     /Lcader Link
JMS* POTSTE
.SIXBT 'ARG1'     /Pot number
.DEC ARG2         /Pot Coef.
.OCT
 ENDM

Remanring MACRO
definitions

[.DEFIN
 .DEFIN
 .ENDM
 .EOT

ONE
MACRO
DEFINITION
In a machine language assembly, should the following statement occur, POTSET Q22,9876, the following coding will be generated in the assembled output:

```
JMS* POTSTE
2262628  /6-bit packed ASCII for 'Q22'
232248   /Binary for 987610
```

During execution POTSTE will operate on the arguments following its CALL, accomplish its function and continue execution of the digital program following the argument string. The subroutine itself is guaranteed to be in core during execution because the LOADER has been directed to load and link this program to all references at load time by the MACRO '.GLOBL'.

On a smaller scale, definitions of the form 'ONE=1' can be made. The definition of allowed arguments with the MACROs guarantees that all others will be 'flagged' during assembly as 'undefined' and not at execution time as 'not-allowed'. Considerable error-diagnostic programming can be dispensed with in this manner. However, program-generated arguments may still require format-validity checks at run-time.

3.4 Hybrid I/O

With reference to Fig. 1, one can see that analog to digital and digital to analog converters are required in the data transfer loop. Hence two separate sections of software have been devoted to A/D and D/A conversion routines. Since not only data lines traverse the interface, but control-lines, interrupt lines, status lines, etc., additional software will accommodate this bi-directional flow of information.
3.4a A/D Conversion Routines

The A/D converter, a Radiation Inc. 5516, is currently multiplexed to 16 channels, with the switching time of the multiplexer in the order of 1 usec. Sample and hold units are connected to all inputs which can be sampled at periodic intervals through use of the 'Sampling Frequency Control' (SFC) at intervals from 26 usecs to .26 secs. Each sample requires 26 usecs conversion time*; at the completion of which an interrupt for service on the digital computer occurs. The A/D converter has ±100v, ±10v and ±1v inputs and has a 12 bit output.

Since the A/D converter is in essence an I/O device, the software has been structured in the form of an I/O handler. An exhaustive description pertaining to the use of the handler can be found in the 'Hybrid Manual'. Here it suffices to mention the three most prominent MACROS defined for the use of the A/D converter.

Routines that set up the multiplexer for data input through the A/D converter are accessible with the two MACROS '.SAMPL' and '.ADRED'. .SAMPL is specifically for a one-channel sample and .ADRED will read from 1 to 64 channels on each execution. .SAMPL exists primarily for efficiency as fewer instructions are executed on a one-channel sample, making for faster entries and exits from the handler routines and in effect reducing delay time.

*Because of the structure of an I/O handler (i.e. the execution of the instruction CAL) the resident monitor assumes control before execution passes to the handler. This 'monitor-execution' time extends sampling time from 26 usecs to 250 usecs. for the 'first' sample of each request for ADC data. Each successive sample of a request takes 50 usecs.
CALL: LOC .SAMPL A,ARG1

where (1) A is .DAT SLOT of device
(2) ARG1 is the channel # to be sampled (0B - 77B)
LOC = USER's tag for current memory location of digital program

EXPANSION: LOC CAL+A
LOC+1 10
LOC+2 ARG1

The handler puts the data into LOC+2 in the 12 most significant bits leaving the channel number in the six least. By not destroying these lower order bits, a program loop on .SAMPL does not have to re-initialize the channel number everytime a sample is made. However any arithmetic operation on the data may require that these bits be masked off.

.ADRED will sample several channels on each call and store the data in core space convenient to the user's program. With .ADRED two buffers are involved; one containing the channel numbers to be sampled, the other storing the digital data obtained by the handler from the MUX channels.

CALL: LOC .ADRED A,L1,L2

where (1) A is .DAT SLOT of device
(2) L1 points to a buffer string of channels to be sampled
(3) L2 points to the data storage area where data is stored left-justified. Unused bits of the 18 bit word remain zeroed.

EXPANSION: LOC CAL+A
LOC+1 2
LOC+2 L1
LOC+3 L2

By left-justifying the data in .SAMPL, as well as in .ADRED, the sign bit can be utilized making arithmetic
manipulation of data easier, and future hardware expansion can be accommodated with a minimum of software modifications (in preference to propagating the sign bit). All data taken from the A/D converter is checked for one of two saturation values, and an error comment echoed on the teletype should the ADC overflow.

**UNLIKE** other I/O device handlers, any attempt to use data from a buffer need not be preceded by a wait for I/O completion, since I/O will be completed on the initial entry into the handler routines. Due to the speed of the converter and multiplexer, conversion times are elapsed handling data and incrementing pointers so that when exit is made from .ADRED (or .SAMPL) all requested sampling has been completed. This is in contrast to the 'Initiate action; proceed with user's program; wait for interrupt' policy of slower I/O devices.

The SEC feature of the interface allows multi-channel sampling at regular intervals. At the periodic expiration of a present time interval, a .ADRED can be repeatedly executed by the software setup of the macro .SFCIN .

**CALL**: LOC .SFCIN A,L1,L2,ARG3,INT10,PRIOR

where

1. A,L1,L2 are as in .ADRED
2. ARG3 is the subroutine that will process data on the completion of one cyclic sampling of L1 at one of two priority levels specified by PRIOR
3. INT10 = period of cycle in usecs. \(\ll^{218-1}\) -- decimal radix.
4. PRIOR is either 0 or 1. If 1, the subroutine ARG3 is executed in the A/D handler at the priority level of the handler at the time of
interrupt of the first conversion complete. ARG3 will be non-interruptable when the API is on, since the ADC is on the highest priority level. On the other hand, if PRIOR = 0, ARG3 is executed at API software level 7.

EXPANSION: LOC CAL+A
LOC+1 3
LOC+2 INT10
LOC+3 PRIOR
LOC+4 ARG3
LOC+5 L1
LOC+6 L2

The external counter is an 18 bit binary counter and is toggled by the 1 MHz clock of the PDP-9. For larger intervals (up to 4369 secs.) in increments of 1/60 th of a second, the real-time clock of the PDP-9 can be programmed to loop on a .ADRED (.TIMER MACRO)(5).

The urgency of analyzing data is determined by PRIOR because processing by ARG3 can be delayed by other interrupts. INT10 has a lower bound of sampling time for

![Diagram](image)

Fig. 7a Sequence of MACRO CALLS to ADC Handler
Initialization

Initialize A/D routines by:
1. Clearing flags and switches
2. Setting up interrupt skip chain to enable entry into A/D routines on 'conversion complete'
3. Setting 'A/D Data Requests' to skip 'Fail-to-initialize test'
4. Pick up 'RESTART' address for A/D saturation test

Return to USER's program

Fig. 7b

Interrupt entrance for 'conversion-complete' (interrupt either from 'wait-loop' or USER's program [if SFC on])

Fig. 7e

Error on interrupt

Read A/D Converter

Convert windowed?

ADRED

SAMPL

Put data in proper location

Leaving channel #, restore ADRED address, nothing else that SFC uses on its ADRED has been used

Put data in appropriate location

Update MAX; start conversion; load next address into MAX's double buffer; increment data pointers

Fig. 7d

Restore all ADRED registers and switches for an entry into an SFC initiated, ADRED

Set next addresses into MAX control's double buffer

Start Conversion

Get next address into MUX control's double buffer

Enter small wait loop in which all interrupts will occur

Save the current SFC and set software switch 'OFF'

Fig. 7g

SFC start request (SFCST)

SPCU setup request

Set a SPCU switch

Pick up SPC interval and set up counter but don't start it

Return to USER's program

Fig. 7f

Programmed A/D data requests, SAMPL and ADRED

No service

Fig. 7c

SFC STOP request (SFCSP or D.USA)

Enable SFC any set software switch 'OFF'

Clear flags

Return to USER's program

Fig. 7a

Failure to initialize handler:

ERROR (1)

Wait for interrupt and SFC routines to be serviced completely

Issue sample and hold pulse

Fig. 7c

Failure to initialize handler:

ERROR (1)

IClear flagol
[Return to USBR]

Fig. 7c

Interrupt entrance for 'conversion-complete' (interrupt either from 'wait-loop' or USER's program [if SFC on])

Fig. 7e

Error on interrupt

Read A/D Converter

Convert windowed?

ADRED

SAMPL

Put data in proper location

Leaving channel #, restore ADRED address, nothing else that SFC uses on its ADRED has been used

Put data in appropriate location

Update MAX; start conversion; load next address into MAX's double buffer; increment data pointers

Fig. 7d

Restore all ADRED registers and switches for an entry into an SFC initiated, ADRED

Set next addresses into MAX control's double buffer

Start Conversion

Get next address into MUX control's double buffer

Enter small wait loop in which all interrupts will occur

Save the current SFC and set software switch 'OFF'

Fig. 7g

SFC start request (SFCST)

SPCU setup request

Set a SPCU switch

Pick up SPC interval and set up counter but don't start it

Return to USER's program

Fig. 7f

Programmed A/D data requests, SAMPL and ADRED

No service

Fig. 7c

SFC STOP request (SFCSP or D.USA)

Enable SFC any set software switch 'OFF'

Clear flags

Return to USER's program

Fig. 7a

Failure to initialize handler:

ERROR (1)

Wait for interrupt and SFC routines to be serviced completely

Issue sample and hold pulse

Fig. 7c

Failure to initialize handler:

ERROR (1)

IClear flagol
[Return to USBR]

Fig. 7c

Interrupt entrance for 'conversion-complete' (interrupt either from 'wait-loop' or USER's program [if SFC on])

Fig. 7e

Error on interrupt

Read A/D Converter

Convert windowed?

ADRED

SAMPL

Put data in proper location

Leaving channel #, restore ADRED address, nothing else that SFC uses on its ADRED has been used

Put data in appropriate location

Update MAX; start conversion; load next address into MAX's double buffer; increment data pointers

Fig. 7d

Restore all ADRED registers and switches for an entry into an SFC initiated, ADRED

Set next addresses into MAX control's double buffer

Start Conversion

Get next address into MUX control's double buffer

Enter small wait loop in which all interrupts will occur

Save the current SFC and set software switch 'OFF'

Fig. 7g

SFC start request (SFCST)

SPCU setup request

Set a SPCU switch

Pick up SPC interval and set up counter but don't start it

Return to USER's program

Fig. 7f

Programmed A/D data requests, SAMPL and ADRED

No service

Fig. 7c

SFC STOP request (SFCSP or D.USA)

Enable SFC any set software switch 'OFF'

Clear flags

Return to USER's program

Fig. 7a

Failure to initialize handler:

ERROR (1)

Wait for interrupt and SFC routines to be serviced completely

Issue sample and hold pulse

Fig. 7c

Failure to initialize handler:

ERROR (1)

IClear flagol
[Return to USBR]
'n' samples plus execution time of ARG3. If INT10 is less than these two combined times, either previous data will be destroyed as new data is being read into L2 or ARG3 will be entered recursively (destructively). This precaution must be taken by the user, as a software check would encounter difficulty in assessing the execution time of ARG3.

Because the SFC is running asynchronously with the PDP-9, any programmed use of the ADC, while the SFC was enabled, would offer potential complications to SFC service and usability. If some programmed sampling were in progress when a SFC start conversion occurred, then at least one A/D sample would be invalidated, and instigation of the pre-specified SFC sampling for this pulse would not have occurred.

This inefficient use of the A/D converter, while the SFC was enabled, was relieved by an addition to the SFC hardware, as outlined on page 8. With the hardware to detect if enough time remained for one A/D sample, and with the software set up to acknowledge only a .SAMPL when the SFC was on, the ADC was relatively free to service the SFC and acknowledge user sampling requests. However, priority of use always remains with the SFC.

Some program examples may be found in the discussion of A/D-D/A errors, page 26, and in that of the TIME DELAY facility, page 52. The flow charts of Fig. 7 give more exacting detail of software operations.

Separate routines exist for stopping and starting the SFC; .SFCIN just sets up a .ADRED that will be started by an SFC pulse. .SFCIN will be referenced again later in a TIME DELAY function.

The following is a list of teletype error messages.
that might be obtained when the ADC software is used:

ERROR MESSAGE:  
- ADA 01 XXXXX /Illegal function
- ADA 02 XXXXX /Zero not an allowed
  /number of channels to be
  /sampled
- ADA 03 XXXXX /No servicing sub. for
  /SFC data
- ADA 04 XXXXX /A/D converter overflow
- ADA 05 XXXXX /SFC flag should not be
  /patched into PI
- IOPS 04 XXXXX /Device not on!!

'XXXXXXXX' is the location of the CALL, in octal, for which arguments are in error or in which trouble has been detected. After typing out the message 'ADA YY XXXXX', the software waits for a command from the teletype keyboard.

The keyboard command is chosen from the following:

KEYBOARD COMMAND:  
- †S /Transfer control to problem's
  /start address; or to the man-machine
  /monitor (if in core) for further
  /diagnosis — see page 64
- †C /Transfer control to DEC System
  /Monitor
- †R /In response to IOPS 04, continue;
  /in response to ADA 04, transfer
  /control to ADC restart address
  /specified in ADC initialization
  /routines.
- †QX /Dump (save) entire memory core
  /on device X, then effect a †C

3.4b A/D Errors

The Sample and Hold units eliminate the effect of slewing errors because all A/D conversions refer to the same instant of time. The 12 bit ADC does not utilize the full 18 bit capacity of the digital computer and inevitably there is an uncertainty equivalent to $\pm \frac{1}{4}$ of the least significant bit of the converter.
The non-linear phenomenon of quantization introduces a significant error and can be shown to be equivalent to the addition of random noise to the analog signal.\(^{(9)}\)

3.4c D/A Conversion Routines

Many hybrid problems require 'dynamic' variables from the digital computer. The operation of the digital servo-set potentiometer hardware makes it impossible to use a potentiometer as a 'run-time' variable, because the computer must be put into Pot Set Mode to accomplish the coefficient change and analog computer voltages are hence destroyed. Use of the servo-set potentiometer would be restricted to iterative-type problems where one makes a computer run and then adjusts parameters for the next try to obtain a better solution.

The number of DACs in the present system severely restricts problem capabilities, and future hardware expansion will undoubtedly incorporate more converters, or a de-multiplexer. Of the four DACs available, one is used in setting potentiometers; the remaining three are for the user and can be individually connected to one or several analog points.

The user DACs are labelled 1, 2 and 3; odd numbers specifying the 9 bit DACs and the even designating the 12. Each converter has a range of \(\pm 10\)\(v\) with finer increments being achieved on the 12 bit converter. A \(\pm 100\)\(v\) range may be obtained by patching the DAC output into an analog amplifier with a gain of 10.
A typical MACRO call for setting a DAC is given:

CALL: LOC DAC1 -9872

where (a) -9872 is the signed coefficient for the DAC (number 1 in this case)

EXPANSION: LOC JMS* DTAST1 .DEC
LOC+1 -9872 .OCT

The program assembler converts LOC+1 from decimal into binary radix and the D/A setting routine then formats the number for the appropriate DAC. But in many situations, LOC+1 will be program generated. Two possibilities have been accounted for in the present setup.

LOC+1 may already be in the correct format (left-justified 12 or 9 bit signed number) in which case no further formatting is necessary. Such would be the case if the A/D converter was the source of the coefficient. With the argument in the correct format, the D/A setting time is more than halved!!

LOC+1 may be set up from a binary coefficient table (right-justified ±10v number) of the decimal counterparts (.DEC pseudo operation on a block of assembled data). The additional formatting that is still required must be accomplished by the DAC setting routines.

LOC+1 is interpreted according to the present status of a FORMAT switch which is either 'ON' or 'OFF' at the time of entering the setting routines. An appropriate MACRO exists for this facility. The 'format' status is initially on.
CALL: LOC FORMAT ON
or
LOC FORMAT OFF

EXPANSION: LOC JMS* FORMAT
LOC+1 ON /or OFF ➔ defined on MACRO
definition tape

---

Fig. 8 Flow Diagram of D/A setting routines

ERROR MESSAGE: HYBRID 10 XXXXX /DAC being used
by system routines.

3.4d D/A Errors

As in the A/D path of the interface loop, time
delay between outputs of the DACs (skewing error) can
cause serious degradation of certain hybrid solutions.
The double buffers of the interface registers are used
to advantage here. DAC coefficients are first loaded into
the first buffer register of Fig. 9. When all DACs have
been loaded accordingly, a command may be issued (simultaneous-
UPDATE) to transfer all blocks of data into the respective
'second' buffers simultaneously.

\[ \text{18 bit word from PDP-9} \]

\[ \text{18 2-input NAND gates} \]

\[ \text{load buffer register IOT2} \]

\[ \text{buffer register} \]

\[ \text{clear buffer register IOT1} \]

\[ \text{output register} \]

\[ \text{load output register IOT4} \]

\[ \text{buffer amplifiers} \]

\[ \text{DAC} \]

Fig. 9 Information flow out of the PDP-9

Software exists to turn an UPDATE mode 'ON' or 'OFF'. At the request to set a DAC, data will be transferred only as far as the first buffer in Fig. 9 (UPDATE ON) or all the way across to the DAC (UPDATE OFF). With the UPDATE software facility 'ON', a call to another routine will result in the 'simultaneous-UPDATE' pulse being issued and cause the specified output buffers of the interface to be updated. The UPDATE status is initially 'OFF'.

CALL: LOC UPDATE ON

or

LOC UPDATE OFF

EXPANSION: LOC JMS* UPDATE

LOC+1 ON /or 'OFF'

CALL: LOC SIMDATE ARG1

where (1) ARG1 is a code for the output buffers of the interface (Fig. 10) (two of which the DACs are connected to)
Fig. 10 Simultaneous-Update code configuration

The features of UPDATE and FORMAT could undoubtedly be incorporated in the D/A routines themselves by extending the argument string. But this would be detrimental to achieving 'fast' routines because of the needless and repetitious argument analysis that would result. Instead, by altering the program prior to many DAC requests, an optimal program results. The software looks after the masking required for the 9-bit DACs that share an 18-bit SOI (Standard Output Interface buffer).

The only remaining source of error is now due to the fact that the zero-order hold function does not have the transfer function that is ideally required. The output of the DAC is a staircase function and this type of reconstruction, of the samples of a continuous function, does not generate a continuous function.

3.4e The A/D and D/A loop

Serious errors arise in the hybrid loop due to the time-delays inherent in A/D conversions and in digital
computations; i.e. in matching the serial nature of the
digital computer to the parallel characteristics of the
analog simulator. The hybrid software can only be expected
to optimize that portion of the time delay which is due to
A/D conversions and D/A setups. That portion of delay
due to digital computations is a function of the algorithm.

At this point it would be instructive to measure
the 'turn-around' (A/D - D/A) time (Figs. 11a and 11b)
and give a relevant example. The turn-around measurement
consisted of nothing more than a simple digital program
which reads in an analog variable and outputs it immediately
on a DAC.

/Turn-around measurement
.IODEV 3
BEGIN
UPDATE OFF
FORMAT OFF
EEM
.INIT 3,0,BEGIN
LOOP1 705604
.SAMPL 3,00
LAC .-1
DAC .+2
DAC1 XX
705602
JMP LOOP1
EEM
.INIT 3,0,LOOP2
LOOP2 705604
.ADRED 3, BUFFER, BUFFER2
Read one channel via .ADRED
LAC BUFFER2
DAC .+2
DAC1 XX
705602
JMP LOOP2
BUFFER 01
000000
BUFFER2 0
/Results of above two small tests recorded in Fig. 11a,
/row A
/The next section reads 'X' variables (.ADRED) and then
/Outputs X variables; results in Fig. 11a, rows B and C
EEM
.INIT 3,0,LOOP3
```
LOOP3
LAS
/Read accumulator switches for
DAC IN
/number=X channels to sample
CMA
/Form 2's comp. of X
TAD (1
DAC COUNT

LOOPX
705604
/Set flag
ADRED 3,IN,OUT
/Read X channels
LAC COUNT
DAC COUNTR

LOOPY
LAC OUT
/Output X dummy inputs sequent-
DAC .+2
/ally
DAC1 XX
ISZ COUNTR
JMP LOOPY
705602
/Clear flag
JMP LOOPX

IN
01
030201; 060504; 111007; 141312;
171615; 222120; 252423; 302726;
333231; 363534; 414037; 444342;
474645; 525150; 555453; 605756;
636261

COUNT
0
COUNTR
0
OUT
.BLOCK 100
.END BEGIN
```

![Diagram](image)

Fig. 11a A/D-D/A delay as a function of input-output pairs.
A simple modification on this program resulted in the measurement of only the sampling time (one-way) of the A/D handler.

Fig. 11b Scope measurement of software A/D sampling time

The digital-computation delay will be over and above those shown in Figs. 11a, b. To illustrate the effects of this turn-around delay, the solution of a linear damped second order differential equation was done on the hybrid computer; 

\[ \ddot{y} + 2\omega \epsilon \dot{y} + \omega^2 y = 0, \]  

(Figs. 12 and 13). The sensitivity of the solution to problem 'mechanization' can be illustrated by employing the digital computer to multiply by \(-\omega^2\) in one case, and by \(2\omega^2\epsilon\) in the other.
Fig. 12 'Block' diagrams for sensitivity analysis

```plaintext
/Error propagation using digital multiplication
.IODEV 3
EEM
BGIN .INIT 3,0,BGIN
MODE IC /Put computer in initial
       /condition mode
HLT /Wait
.SAMPL 3,00 /Read IC voltage; \dot{Y}(0) for test 'A'
LAC -.1 /Y(0) for test 'B'
JMP .+3 /JMP .+3 for test A; NOP for test B
CMA /Multiply by -1
TAD (1000
RTR /Divide by 2
DAC .+2
DAC1 XX /Set DAC to close feedback loop
HLT /Wait
MODE OP /Put computer in operate mode
.SAMPL 3,0 /Read multiplicand
LAC -.1
JMP .+3 /JMP .+3 for test A; NOP for test B
CMA
TAD (1000
RTR
DAC .+2
DAC1 XX /Set DAC to product value
JMP LOOP
.END BGIN-1
```

unscaled \[ \dot{y} + y + 16y = 0 \]
unscaled \[ \dot{Y}(0) = 0 \]
unscaled \[ Y(0) = -40 \]

scaled \[ \dot{y} + .5\dot{y} + 2y = 0 \]
scaled \[ \dot{Y}(0) = 0 \]
scaled \[ Y(0) = -80 \]
3.4f Control of Analog computer from the Digital side

The control of the analog simulator has been set up on two levels; (a) on a digital-computer software level and (b) on a man-machine level (which is inherently backed up by the former). The latter is discussed later under the 'Man-Machine' category of Fig. 6.

During the execution of a hybrid problem, one usually has four major programs in core; (a) the hybrid problem, (b) the analog-control and hybrid I/O routines, (c) the man-machine link (monitor) and (d) the DEC system (resident) monitor. The routines for controlling the analog simulator from the digital computer to the extent permitted by the present interface (page 2) have been defined as MACROs and are linked to the user's program by

Fig. 13 Comparison of hybrid computer solutions with actual solution
.GLOBLS.

The control routines update 'last-action' registers so that the state of the analog computer is always known. In cases where subroutine arguments have a likelihood of being program generated, an analysis for validity is usually carried out. In other cases, the hardware structure masks out illegal arguments. In discussing the control routines individually, a list of teletype error diagnostics will be given, along with the software structure.

**MODE CONTROL**

**CALL:** MODE ARG  
**EXPANSION:** .GLOBL MODSET  
JMS* MODSET  
ARG

The hardware takes only the 3 least bits of an 18 bit word, ARG, to effect control. In the decoding logic, a multiple assignment to two of the six possible modes utilizes all 8 possibilities for three bits; making an illegal code impossible and simplifying the decoding hardware. There are six modes, defined with the MACRO on the definition tape; from which ARG may be selected.

**FUNCTION**  
**DEFINITION**  
POTSET PS = 0 and 4  
RATE TEST RT = 2  
STATIC TEST ST = 5  
INITIAL CONDITION IC = 1  
OPERATE OP = 7 and 6  
HOLD HO = 3

The above features forego any need for software argument checks. Only one error diagnostic is provided; for the detection of hardware failure when the 'correctness-bit' of Fig. 4 fails to come up in 5 ms. This 'correctness-bit'
provides the facility for detecting when the relays have finished switching so that the digital program may be allowed to continue following the MODE request (see Fig. 14).

Fig. 14 Mode-selection software flow-diagram

**INTEGRATOR CONTROL**

CALL: \texttt{INTEGR CODE}  
EXPANSION: \texttt{.GLOBL INTEGRA}  
\texttt{JMS* INTEGRA}  
\texttt{CODE}

All integrators can be controlled individually or in groups, permitting the efficient use of iterative analog and hybrid techniques for the solution of boundary value problems and partial differential equations. One specific example is the automatic plotting of stability areas for a mathematical expression such as Mathieu's equation \( \dot{y} + (a - 2q \cos 2t)y = 0 \). In this case, the analog section operates in high-speed rep-op making the necessary thousands of runs while the logic controls the analog computer, senses...
the stability of the system equation and controls the x-y plotter so that it automatically plots the stable areas for values of 'a' versus 'q'.

The 18-bit code word, for controlling 17 integrators uniquely for the repetitive-operation (REP-OP) of the analog computer, is of a sign-magnitude format. The sign bit conveys to the software which of two types of integrator switching is being used. INTEGR ARG1 (where bit 0 of ARG1=0) conveys electronic switching while INTEGR XCT+ARG1 would establish that connection has been made to the relays. The final outcome, of course, depends on the patch connections made by the user from the interface to the MLG patch-panel (or the relay drivers).

Fig. 15 Integrator-mode selection software

The 17 least significant magnitude bits will switch 17 integrators into one of two modes, IC or OP,
accordingly, bit = 1 → IC, bit = 0 → OP. The integrators controlled depend on patch connections made on the computer. Integrators may share bits to accommodate all 24 integrators.

With relay switching, the precise time at which switching is complete cannot be determined and an entire 5 ms. is allowed for completion. Electronic switching has a zero wait-time.

DIGITAL ATTENUATOR SYSTEM (DAS) CONTROL

The DAS facility has three modes and is concerned with the operation of the servo-set potentiometers. Two of the three modes, RDT (Read-out) and CHK (Check) have been defined with the MACRO control call. The third mode, SET, is accessible only to the system program for setting potentiometers, as certain procedures must have occurred prior to the issuance of the SET command (selecting the potentiometer, acquiring the coefficient, and putting the computer into PS).

The DAS routines (Fig. 16) will probably be used infrequently by the user because the functions are inherent in system requests concerning the potentiometers. That is, in setting a potentiometer, the DAS mode changes and the analog mode changes are part of the potsetting
routines.

CALL: DASMOD CHK
EXPANSION: .GLOBL DASSET
JMS* DASSET
2

Fig. 16 DAS facility software flow-chart

SLOW-SPEED MULTIPLEXER CONTROL

This multiplexer, because of its slow speed, will be used exclusively for problem set-up and check-out. The multiplexer can access any one of 540 points (including the servo-set potentiometers) on the analog patch panel within approximately 0.5 to 2 secs. The conversion time for the voltage on the selected channel is 15 ms.

The structure of the mechanical stepping switches is such that the switches keep moving until the address specified is found. It is imperative that an illegal address never be specified. Several such addresses exist (for system hardware that is available but not yet connected --- resolvers, function generators, etc.) and must be detected by the system software before a search is entered. The maximum time allowed, by the software, for any search is 3 secs; at which time an error is recognized.
CALL: SIGSEL ARG1
EXPANSION: .GLOBL SETSIG
JMS* SETSIG
.SIXBT 'ARG1'

The hardware recognizes 3 six-bit ASCII characters in an 18 bit binary word as the channel address for the 3 independent banks of switches. At assembly time, the .SIXBT feature of the machine language assembler generates the correct code from the user's argument which is in the form of a character and a two digit number. The character and numbers are the same as those on the push-buttons used in the manual control of the signal selector. As an example SIGSEL A22 would expand into;

JMS* SETSIG
016262

to result in the selection of amplifier 22.

Fig. 17 Signal-Selector software flow-chart

ERROR MESSAGES: HYBRID 05 XXXXXX /Selector Address
/non-existent
HYBRID 06 XXXXXX /Signal-selector
/hardware failure
/(3 sec. select
/time exceeded).
Because the process of selecting a channel and sampling the voltage on it are distinct, the software routines have been kept so as well. While it is obvious that in selecting a channel, one desires a sample from it as well, the separate routines allow efficient repetitious sampling of any selected point with the electronic digital voltmeter (EDVM).

CALL: LOC DVMSAM
EXPANSION: .GLOBAL EDVMST
                  LOC JMS* EDVMST
                  LOC+1 XX /Storage space for sample

DVMSAM, upon being interrupted by a 'conversion-complete' from the EDVM, formats the BCD number, and then leaves the sample in a binary format in the CALL location plus 1. For example, if the reading on the EDVM was +101.25 volts, then the core location would contain 23613_8 (the octal equivalent of 10125). -101.25 v. would just be the two's complement of 23613_8, 754165_8. This is the same format used for setting potentiometers and DACs and can be deposited into the appropriate location of the MACRO expansion. Successive calls to DVMSAM will wait for the previous 'action' to complete before instigating another sample.

Unlike the ADC, the speed of the EDVM prohibits waiting for a 'conversion-complete' before returning control to the user with the sample in LOC+1. Instead, the EDVM interrupts the user's program at some later time after the sampling request, to complete the data transfer. Therefore, before data is used, it best be preceded by a test (at the user's discretion) that will wait for the
transfer if it has not already been made.

A visual summary of the routines is given in Fig. 18.

CALL: DVMWAIT
EXPANSION: .GLOBL DVMWIT
  JMS* DVMWIT

--- Diagram ---

**EDVM sampling request**
- Check if sample currently underway
- Set a 'busy' switch
- Save 'data' storage location
- Return to USER to wait for interrupt

**Interrupt entry for EDVM sample complete**
- Have active registers from USER's program that will be used in this section
- Clear busy switch
- Read BCD EDVM output and put in data storage location in binary format
- Restore USER's active registers that have been destroyed by above routines
- Reactivate interrupt facilities and return to USER's program at point of interrupt

**EDVM test-for-completeness entry**
- Check 'busy-switch'
- Return to USER's program

---

**Fig. 18 EDVM Sampling routines**

**POTENTIOMETER-SETTING CONTROL**

As mentioned previously, in setting a servo-set potentiometer, the analog computer must be put into PS mode. Any voltages on the integrators at the time are destroyed and use of the servo-potentiometers as run-time variables is thus prohibited. However, for compute;iterate; re-compute applications (i.e. boundary-value problems) the servo-set potentiometers will find applications.

Regardless of the application, a simple software CALL provides the selection and setting of the requested potentiometers as well as performing the required mode changes (Fig. 19). Setting a servo-set potentiometer manually involves several steps and precautions with which the user does not have to concern himself with when the same function is carried out via software.
CALL: LOC POTSET Q22,+9872  
EXPANSION: 
  .GLOBL POTSTE 
  LOC JMS* POTSTE 
  LOC+1 .SIXBT 'Q22.' /3 letter ASCII code gets 
  .DEC /converted by .SIXBT at 
  LOC+2 +9872 /assembly 
  .OCT /Coefficient base 10 will 
  /be converted to binary at 

---

Fig. 19 Servo-set potentiometer routines.

Appropriate precautions must be assumed by the 
user to insure that LOC+2 is of the correct format if 
LOC+2 is being program generated. (See previous 'EXPANSION').

ERROR MESSAGES: HYBRID 07 XXXXXX /Pot failed to set 
within specified /tolerance 
HYBRID 11 XXXXXX /Pot mechanism has 
failed 
HYBRID 03 XXXXXX /-ve pot. coef. not 
allowed
HYBRID 02 XXXXXX /Parameter exceeds range allowed /
for DACs.
HYBRID 12 XXXXXX /only Q pots are allowed for a /
potentiometer set.

Before the software recognizes a potentiometer 'fail-to-set' condition, the tolerance (i.e. the magnitude of the difference in the specified and actual coefficients) must exceed the pre-specified allowance. This allowance can be set by software.

CALL: TOLRNCE TOL
EXPANSION: .GLOBL TOLERT
        JMS* TOLERT
        .DEC
        TOL
        /0 < TOL <= 9

The allowance can assume values from +0 to +9. Ideally, the potentiometers should set to exactly what has been requested, but the accuracy will be a function of the analog computer's performance from day-to-day. The optimum value for a particular day may be sought out by trial and error during the problem set-up and check-out stages of an hybrid problem (see Man-Machine monitor, page 64).

Fig. 20 Potentiometer tolerance set-up routines
ERROR MESSAGE: HYBRID 08 XXXXX /'Tolerance not allowed

AUTOMATIC SETUP AND COPY ROUTINES

During the setting of potentiometers and DACs, a storage buffer is continuously updated in regards to 'last-coefficients'. At any time during a program execution or problem set-up, a permanent file of this buffer may be acquired on one of several output devices. These files in turn may be used as input to set all the recorded potentiometers and DACs 'en masse'. This facility would forego needless and repetitious problem setups for problems that will be run as 'standard' routines. Fig. 21 is an example of a hard copy record.

<table>
<thead>
<tr>
<th>POT#</th>
<th>COEFFICIENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>P06</td>
<td>+00500</td>
</tr>
<tr>
<td>Q06</td>
<td>+00500</td>
</tr>
<tr>
<td>Q09</td>
<td>+08965</td>
</tr>
<tr>
<td>Q10</td>
<td>+05000</td>
</tr>
<tr>
<td>Q21</td>
<td>+04000</td>
</tr>
<tr>
<td>Q25</td>
<td>+03000</td>
</tr>
<tr>
<td>P51</td>
<td>+09500</td>
</tr>
<tr>
<td>P52</td>
<td>+10200</td>
</tr>
<tr>
<td>P53</td>
<td>+07300</td>
</tr>
<tr>
<td>P54</td>
<td>+01200</td>
</tr>
<tr>
<td>P55</td>
<td>+00300</td>
</tr>
<tr>
<td>D01</td>
<td>-0533</td>
</tr>
<tr>
<td>D02</td>
<td>-01300</td>
</tr>
<tr>
<td>D03</td>
<td>+03000</td>
</tr>
<tr>
<td>D03</td>
<td>+06298</td>
</tr>
</tbody>
</table>

Fig. 21  Teletype copy of a coefficient buffer

CALL: DATA ARG1, ARG2
EXPANSION: .GLOBL HRDCOP
           JMS* HRDCOP
           ARG1
           ARG2
where (1) ARG1 is the .DAT SLOT* output device for the coefficient buffer; the system expects this to be either .DAT SLOT 1 or 10 (a restriction no more serious than the system assembler expecting input on .DAT SLOT -11) (5).

The hybrid assignments to these device slots are the DEC tape 'B' handler and teletype handler respectively.

(2) ARG2 is a pointer to the file name and extension information to be given the record. The 3 letter extension must be 'BIN' and will be over-ridden if not.

ERROR MESSAGE: HYBRID 09 XXXXXX /.DAT SLOT not allowed /for dump of coefficient /DATA

The coefficient buffer has been initially set to unused values at load time to distinguish the appropriate device (DAC or servo-pot) as 'unused'. This facilitates needless operations (listing and setting) for the unused hardware by the DATA and following BLOCK routines (Figs. 22a and b).

CALL: LOC BLOCK ARG2
EXPANSION: .GLOBL BLKSET
LOC JMS* BLKSET
LOC+1 ARG2

where (1) ARG2 is as in 'DATA'. Input of the coefficient record will occur on .DAT SLOT 1. All coefficients that have been recorded as 'used' will be set appropriately.
Fig. 22a,b  Flow diagram of DATA and BLOCK routines

The assignment on .DAT SLOT 1 of the hybrid system was chosen to be .DTB because of its simultaneous two-file (Input-Output) capabilities for the DATA and BLOCK operations. Any other assignment to this slot would be required to have the same capabilities.

CONTROL CALL SUMMARY

MODE OP  /Put analog computer in operate mode and wait for relay switching. Other possible modes: PS, RT, ST, IC, HO

INTEGR CODE  /Control the integrators individually, according to the word 'CODE', either electronically or mechanically (relay), for two possible modes IC and OP

DASMOD CHK  /Put the digital attenuator system in CHK (Check) mode. Other mode: RDT (Read-out)
SIGSEL A22 /Move signal selector stepping switches (Slow Speed Multiplexer) to specified analog point (amplifier 22).

DVMSAM /Sample the voltage for current signal selector position, format and leave sample in CALL location plus 1.

POTSET Q22,9675 /Select and set the servo-set potentiometer specified (Q22) to the coefficient given (.9675)

TOLRNCE ARG1 /Set software to recognize a potentiometer 'fail-to-set' condition if there is a disagreement of +ARG1 between desired and actual coefficient.

BLOCK FILENM /Set all pots and DACs to the coefficients on .DAT SLOT 1 under the file name 'FILENM'

DATA ARG1,FILENM /Create a file 'FILENM', on .DAT SLOT ARG1, of the coefficient buffer that currently resides in core for pots and DACs. (buffer is continuously updated as Pots and DACs are set individually or in BLOCK mode)

After the various error diagnostics that might occur on the teletype keyboard, the computer 'listens' for one of the following keyboard commands; tS, tC, or tQ as outlined with the ADC error messages (page 26).

3.4g Control of the Digital Computer from the analog side

One can think of an I/O device as controlling the digital computer when that device 'interrupts' it
for service. The interface is just an elaborate I/O peripheral with several interrupt lines. By appropriate connections, the analog computer can interrupt the digital machine for service and in effect the digital computer can be slaved to the simulator.

The change of state of a comparator can be used to cause an 'interrupt' on the PI level in the digital computer. The .COMPR CALL sets up digital software to process an interrupt, with the subroutine SERVICE, on the change of state of comparator NUMBER. A particularly useful application of the 'comparator-interrupt' feature would be in the implementation of the automatic rescaling of a hybrid problem; SERVICE freezes the analog problem, rescales it and then continues. Because of the nature of the comparators, the service subroutine must reset the cause of the interrupt if future interrupt capabilities are to be realized from the same slot number. In the case of rescaling, the action itself should reset the comparator.

On the interface, only three interrupt 'slots' exist for external devices to access service from the PDP-9. To accommodate all the hardware (MODE selector, comparators, signal selector control, EDVM, etc.) that must interrupt the digital computer to acknowledge completion of a task, a 10-bit OR gate followed by a monostable was connected to one of these three PI slots. Of the 10 bits, only
four are available for comparator lines; the rest being used by the system hardware.

**ERROR MESSAGES:**

HYBRID 04 XXXXXX /Comparator slot # not allowed
HYBRID 01 XXXXXX /Unresolved interrupt from the 10 input lines

---

**Fig. 23 Comparator-interrupt setup routines**

3.5 **Utility Library**

Several routines have been written to implement the two computers for specific problem types. As experience is gained in hybrid computation on the installation, the utility library will undoubtedly expand. Present requirements have prompted a time delay and function generator program.

3.5a **Time-Delay**

The time delay facility operates in conjunction with the A/D and D/A hybrid I/O routines to delay input on
up to three A/D channels by times $T_x$, $T_y$ and $T_z$. The magnitudes of the delays are limited only by the amount of core available as 'delay-line' storage. A pointer loops on a memory buffer, reading out the delayed value and then depositing a new one in its place. At the expiration of $T_i$, $f(t - T_i)$ occurs on a pre-specified DAC (9 or 12 bit), (see Fig. 24). The output of this DAC during the first $T_i$ secs. is preset to $f(0)$. Otherwise, depending on the previous contents of the DAC's buffer, peculiarities would result on the delayed output for the first $T_i$ secs. Timing errors for the delays is eliminated by the sample-and-hold units on input and by the simultaneous update feature on the buffers of the interface on output.

The sampling interval, once set, is a constant common to all three delays. Criteria for the sampling rate of continuous data must be considered for negligible signal degradation (10).

Once a DAC has been used for a delayed output, the system tags it as used, so that a call to the DAC setting rou-
tines in the hybrid package results in an error diagnostic. Only a re-initialization of the A/D handler will liberate the used DACs from their use in the time delay facility. As the ADC output is 12 bits, the routines take the most significant bits for 9-bit DACs.

This time delay facility assumes that the outside world is the source-object of input-output, with reference to the PDP-9. Other situations can be most easily accommodated by a user written DELAY using the A/D handler functions available (.SFCIN) and/or the D/A setting routines. The stopping and starting of the delay routines are done through the SFC MACROs .SFCST and .SFCSP. Control of f(t), however, must be exercised by user commands external to the delay facility. The listed example illustrates a typical application, and Fig. 25 gives the detailed operation of the software.

CALL: LOC .DELAY A,CHANLS,INTRVL,CORE,TX,TY,TZ,STARTX,STARTY,STARTZ

EXPANSION: LOC CAL+A
           LOC+1 7
           LOC+2 CHANLS
           LOC+3 .DEC
           LOC+3 INTRVL
           LOC+4 .OCT
           LOC+4 CORE
           LOC+5 .DEC
           LOC+6 TX
           LOC+7 TY
           LOC+7 TZ
           LOC+10 .OCT
           LOC+11 STARTX
           LOC+11 STARTY
           LOC+12 STARTZ

where (1) A is the .DAT SLOT of the A/D handler
(2) CHANLS is a coded word (6-bit packed octal)
for the multiplexer channels on which input is to occur.
\[ \text{CHAN Z} \quad \text{CHAN Y} \quad \text{CHAN X} = \text{CHANLS} \]

i.e. 010310₈ would indicate channels 1₈, 3₈, and 10₈

The DACs on which output will occur are permanently assigned accordingly:

- Bits 0-5 Delayed output of channel on DAC #3
- Bits 6-11 Delayed output of channel on DAC #1
- Bits 12-17 Delayed output of channel on DAC #2

All zeroes in the appropriate bits (00₈) indicate that the corresponding DAC will not be used.

Examples: 000700 Input Mux 07; output on DAC #1
          100000 Input Mux 10; output on DAC #3
          060015 Inputs on Mux 15 & 06; outputs on DACs 2 and 3 respectively

Using the 00₈ in this way prohibits the use of MUX 00 as input.

(3) INTRVL is a base 10 number for the sampling frequency interval in usecs. All input channels are sampled at this interval.

(4) CORE is the address at which the user guarantees enough storage for the handler's 'delay-line' data. The amount of core required by the handler is the sum of \( TX, TY, \) and \( TZ \).

(5) \( T(A) \) is the number of sampling intervals, base 10, in each delay (\( A = X, Y, Z \) for the corresponding input channels; see CHANLS). The
total time delay for any input-output pair is
\[ T(A) \times \text{INTRVL} \]

(6) STARTX, STARTY and STARTZ are the values \( P(0) \) mentioned previously. They must be in ADC format (left justified). These core locations LOC+10, 11 and 12, are most easily set up at run time by A/D samples on the channels concerned; prior to calling .DELAY.

Example:

```assembly
.GLOEL REGSO7,REST07
.IODEV 3
.LL
.OTHER 3,0,HERE
.COMPR 3, RESCALE
.MODE IC
.ADRED 3,11,SETUP+10
.SETUP .DELAY 3,070003,900,DUMP,99,0,199,XX,XX,XX

.MODE OP
.SFCST 3

.LL 3
.OTHER 070003
.DUMP .BLOCK 452

.RESCALE JMS* REGSO7
.MODE HO
.SFCSP 3
.LAC DUMP
```

//In system library
//Load handler on .DAT SLOT 3
//Initialize handler
//Set up a comparator for
//Interrupt capabilities
//Put in IC to get IC voltages on
//MUX channels
//Results in XX,XX,XX being setup
//in .DELAY

//Start analog computer (MUX inputs)
//Start SFC

//Wait till I/O has finished on
//handler and then shut it down
//Transfer control to DEC monitor

//4528 = 19910 + 9910

//Entrance for interrupt; save
//active registers
//Freeze analog (MUX inputs)
//Stop SFC
//Rescale data in DUMP to DUMP+9910
//Rescale data in DUMP+99 to DUMP+99+199
//Rest of rescaling
//Enable analog
//Restart SFC
//Restore active registers

Fig. 25 .DELAY software flow-chart
3.5b Function Generation

GENERATION

The value of a function at known points in its n-dimensional space enables a calculation of an approximate value at points other than those specified. Depending on the density of the 'known' points, a fairly accurate value of the function may be obtained, by linear extrapolation, for the function at its new coordinates. These 'known' points are data for a program known as 'FUNCTN' which performs this n-dimensional extrapolation, given n inputs.

The data for the 'FUNCTN' program has a rigid format to enable the software to:

(a) communicate with the data for any dimension, the dimension and density of the function being restricted by memory size;

(b) facilitate extrapolation between 'hyper-planes' until the space has been reduced to an extrapolation between two points. The last extrapolation gives the function value in the n-space.

A data-format generator enables the functions to be prepared and stored under unique names prior to their use at run-time. At load time, a simple .GLOBL reference to the function's data name will cause it to be loaded and linked to the extrapolator 'FUNCTN'. (See APPENDIX III)

The format generator communicates with the programmer through a series of questions via one of two I/O media; the CRO with light-pen (Fig. 26 and 27), or the teletype (Fig. 28).
With the CRO and light-pen, reasonably accurate functions can be drawn quickly for one and two dimensions. Whereas with the teletype, function values and coordinates can be typed for up to five dimensions. Both input media have facility for 'editing' the 'loader-formatted' binary output.

Multi-defined functions can be accommodated by segmenting the n-space function into 'unique' sections. The

![Fig. 26 Scope; light-pen input media for 2 dimensions](image)

![Fig. 27 Scope; light-pen input media for 1 dimension](image)
user must then make the FUNCTN calls to the appropriate sections to pick up all the function values.

AMOUNT OF FREE CORE AVAILABLE 006275
EDIT MODE Y N?N
WHAT IS THE DIMENSION OF YOUR FUNCTION 1,2,3,4 OR 5? 2
HOW MANY GRID HYPERPLANE POINTS ALONG AXIS 1? 3
HOW MANY GRID HYPERPLANE POINTS ALONG AXIS 2? 2
TYPE OUT COORDINATE VALUES FOR AXIS 2.
-3.824
+9.510
TYPE OUT COORDINATE VALUES FOR AXIS 1.
-8.741
-0.065
+6.732
PLEASE TYPE OUT FUNCTION VALUES NOW
1,1?+9.523
2,1?+7.542
3,1?+3.507
1,2?+5.989
2,2?+0.876
3,2?-6.452
TYPE FILE NAME XSPACE

Fig. 28 Teletype interrogation for data

The detailed operation instructions serve no purpose here and can be found in the 'Hybrid Manual'. A small example for using the 'formatted-data' follows.

UTILIZING THE GENERATOR'S OUTPUT

Because the output of the function generators is in LOADER format, it is most easily pulled into core at LOAD time by .GLOBL links to the USER's program library.

CALL: LOC FUNCTN ARG1,ARG2,ARG3
EXPANSION: LOC .GLOBL FUNGEN
LOC JMS* FUNGEN
LOC+1 ARG1
LOC+2 ARG2
LOC+3 ARG3

where (1) ARG1 is the dimension of the function
(2) ARG2 is where the input variables, in ADC format, are stored.
ARG3 is the file name for the data
FUNGEN does the extrapolation of the 'n' variables at ARG2 on the data in file ARG3 and returns to LOC+4 with the answer in a decimal format in the accumulator (AC). For instance, if the extrapolation produced +9.785 as F(X) then 23071g would be found in the AC. Note that this is the same format as used in setting DACs and can be deposited into the appropriate argument location for a D/A set.

EXAMPLE:

```
.IODEV 3
.GLOBL XSPACE,YSPACE

FORMAT ON
UPDATE ON
BEGIN .INIT 3,BEGIN

.ADRED 3,CHANLS,INPUT
FUNCTN 4, INPUT,XSPACE
DAC +2
DAC2 XX
FUNCTN 2, INPUT+4,YSPACE
DAC +2
DAC1 XX
SIMDATE 14
JMP BEGIN+4

CHANLS 6
060504
171607

INPUT XX
XX
XX
XX
XX
XX
XX
XX

.END

FUNCTN ERROR MESSAGES:  FUNCTN 00 XXXXX /Dimension of data
/in file does not
/agree with dim. in
/MACRO call

FUNCTN 01 XXXXX /Input variable is
/outside range of
/variables on file
```
Fig. 29 FUNCTN software flow-chart

Problem-type algorithms

Algorithms for rescaling, for solving particular boundary value problems and optimizations, for simulations, etc., will have to be written by the individual by using the Hybrid I/O routines.

3.6 MAN-MACHINE Interface Routines

3.6a Light-Pen Handler

As an I/O peripheral, the CRO and light-pen were provided without software that would enable their use on the same level as the teletype, DEC tape, etc. The first requirement in utilizing the CRO was to write a general I/O display handler.

The handler will display up to four distinct
classes of data, three of which are TEXT, LINE, and RANDOM-POINT; in each of which the user can interrupt at any time to effect a program branch to a servicing subroutine. The service subroutine is determined from a header-word of the data buffer being displayed at the occurrence of the interrupt. Analysis of the precise status of the CRO beam; X and Y coordinates, number of character in current text string, number of point, etc., can be determined by this subroutine through status registers made accessible to it.

Finer features of the handler add even greater flexibility to the software applications. A new dictionary for the TEXT mode may be defined for specialized situations (i.e. contour segments). In certain modes, screen sections may be expanded to full size while all else disappears. Display buffers can be scattered throughout core and linked appropriately. If desired, buffers may be displayed and yet no interrupt will occur if the light-pen is placed over points in them. With the computer being as fast as it is, an interrupt and service have occurred and finished long before the light-pen has been removed, therefore a 'reflex' time may be defined to disable the display-flag interrupt for any desired length of time after the initial flag-'set' condition.

The 'Hybrid Manual'(8) goes into greater detail on the features and applications of the light-pen handler. A programmer's conversational ability with the computer
is enhanced considerably with this handler; so much so that it appears particularly suitable for linking the 'problem-analyst' to a hybrid problem. Queries, decisions, and selections are linked to appropriate routines via data displayed; the light-pen enables communication with the digital machine. (See Fig. 30).

3.6b Visual-Aid Monitor (Hybrid Monitor)

The monitor, operating in conjunction with the light-pen and teletype keyboard handlers, enables the programmer to interact freely with his problem by permitting direct on-line control and interrogation of program action at anytime. In addition the monitor can assume control, to display the visual status of the hybrid problem and hardware, when an error has been detected by the system software.

While the hybrid problem is being executed, the user has two distinctive means of bringing this MAN-MACHINE monitor into control to affect the link between him and his problem. One is by programmed entry and the other is by manual intervention. When a particular control character (\texttt{IS}) is struck on the teletype keyboard, the hybrid problem is frozen (excepting real-time inputs), and the monitor appears on the scope as in Fig. 31. Whereas, a similar programmed entry can be gained in one of three ways.

Programmed entry is achieved by: (a) successive initialization of the light-pen handler, (b) \texttt{IS} after error
**Fig. 30 Light-pen Handler Flow Chart**
Fig. 31 Hybrid-Monitor Screen Display

Fig. 32 'MODES' Section
diagnostic type-outs. (tC brings in the DEC system Monitor), and (c) by a DDT (HYBRID) break-point.

Once the MONITOR assumes control, one can control the analog machine with the light-pen. The status of data and control registers, i.e. last A/D and D/A transfers, MODE selection, active digital-registers, etc., appear on the screen or are accessible through keywords on it, (Figs. 32 and 33). Analog voltages can be examined, Pot and D/A coefficients changed, and the digital program altered; all via the light-pen. The data for the last selections made on the SLOW and FAST speed multiplexer is continuously displayed on the CRO screen. After all interrogation and changes have been completed, control is restored to the digital program by light-penning the word 'CONTROL'.

To aid in debugging programs, software that closely parallels DEC's DDT program
resides in core with the monitor. This DDT routine is not meant to replace the 'system' software, but is intended to facilitate correction of recoverable errors detected by the hybrid software and to couple debugging techniques with the highly interactive nature of hybrid computation provided by the CRO display.

The use of the hybrid monitor is optional. Once a problem has been debugged or is being routinely run, the special monitor may no longer be required. Some 2K of memory is then available for program use when the hybrid monitor is not loaded.

Again, much greater detail may be found in the 'Hybrid Manual' (See Fig. 34)

3.6c Error Diagnostic Routines

The Monitor Error Diagnostic (.MED) routines of the DEC system monitor provide a communication link between the programmer and the computer. The diagnostics are effective only for routines that use the monitor via the CAL instruction or the PI-API interrupt facility (basically I/O routine entries and interrupt service dispatches).

Since the 'hybrid' application of the computer greatly extends the need for more diversified diagnostics, the hybrid routines are structured to 'censor' the incoming arguments and monitor 'correctness' of execution, on a subroutine level (JMS). In a format similar to the .MED diagnostic, an appropriate teletype message is issued when trouble has been detected. The format of the error diagnostics is 'NAME:NUMBER:ERROR LOCATION.'

To emphasize the variety and type of hybrid monitoring that has been incorporated in the previously outlined
Enter here to set up analog via BLXSET (data from SLOT1) Enter here to change Pot-setting tolerance Display 'TYPE FILE NAME' on scope and wait for six letters to be typed Display check-mark next to selection

Enter here to Select a Mode
Analyze letter penned and Select appropriate MODE Display check-mark next to selection

Enter here on a 'SELECT'
Pick up first input to input buffer
Is it a D?
Pick up 2nd input buffer slot
Compare number with 0
Pick up 2nd and 3rd input buffer slots and fix up sampling routine to sample this MUX channel (MUX channel changed in the one with the dot beside it) Display check-mark next to selection

Fig. 34 cont'd 'HYBRID-MONITOR' software flow chart
Fig. 34 cont'd 'HYBRID-MONITOR' software flow chart
ADA 01 XXXXXX /Illegal function for ADC
ADA 02 XXXXXX /Zero not an allowed number of
/channels to be sampled
ADA 03 XXXXXX /No service subroutine for SFC data
ADA 04 XXXXXX /A/D converter overflow
ADA 05 XXXXXX /SFC flag should not be patched into
/PI
IOPS 04 XXXXXX /Device not on

HYBRID 00 XXXXXX /Analog Computer Mode failed to set
HYBRID 01 XXXXXX /Unresolved interrupt from 10 input
/gate on PI
HYBRID 02 XXXXXX /Parameter exceeds range allowed
/for DACs
HYBRID 03 XXXXXX /Negative Potentiometer coefficient
/not allowed
HYBRID 04 XXXXXX /Comparator Slot number not allowed
HYBRID 05 XXXXXX /Signal Selector Address non-existent
HYBRID 06 XXXXXX /Signal Selector Hardware failure
HYBRID 07 XXXXXX /Pot. failed to set within specified
/tolerance
HYBRID 08 XXXXXX /Tolerance not allowed for setting
/pots.
HYBRID 09 XXXXXX /.DAT SLOT not allowed for dump
/of coefficient data
HYBRID 10 XXXXXX /DAC being used by system routines
/(TIME delay)
HYBRID 11 XXXXXX /Potentiometer mechanism has failed
HYBRID 12 XXXXXX /Only Q pots allowed on a potentiometer
/set

FUNCTN 00 XXXXXX /Dimension of data in file does not
/agree with dimension in MACRO call
FUNCTN 01 XXXXXX /Input variable is outside range of
/variables on data file

The 'Hybrid Error Diagnostic' (.HED) routines
wait for a keyboard command from the programmer.

↑C
↑QX
↑S

↑S has two possible outcomes. If the hybrid
monitor is in core, and being used,↑S will result in
'HYBRID' control; otherwise↑S will result in a re-entry
of the user's program at its initial starting address.
↑C will, as it will at anytime, transfer control to the
resident system monitor. QX dumps the entire memory, in dump mode, on to device X.
4. SUMMARY AND CONCLUSIONS.

Some of the examples previously given have already demonstrated the manner in which machine-language programs are structured for the utilization of DEC- and HYBRID-system software. The use of MACROs in both software systems has achieved a FORTRAN image for the programmer who may, in the past, have dealt almost exclusively with the higher-level computer languages. The utilization of machine-level programming, at first being somewhat of a set-back for the FORTRAN User, is appreciated in real-time applications where (a) hardware configurations are flexible and in great variety, and (b) time is of the essence.

The hybrid software, having been created for general applications, does, in some instances, take more time and core than what might be tolerated in certain hybrid applications. In such situations, one would resort to writing a specialized routine in machine language and append it to the hybrid library as a special-case subroutine or handle. DEC system-device handlers are an example to this. The hybrid system has not yet been utilized enough to indicate where similar situations exist in the hybrid system and where software improvements might be required.

It would have been instructive to support the thesis work by an hybrid problem; but due to construction delays, the hybrid interface is as yet incomplete with only limited facilities available for hybrid applications. The software
for the remaining incomplete facilities functions to the point where the routines wait for the 'ACTION-COMPLETE' interrupt. That is, the correct IOTs (Input-Output transfer instructions) are executed and the error diagnostics are issued as necessitated. Incomplete hardware includes the Slow-speed multiplexer, the pot-setting facility, and the comparator-interrupt facility. The remaining hardware, and accompanying software, operations have been demonstrated by the examples in the thesis.

In conclusion, the software developed in this thesis forms a nucleus for a HYBRID system; a nucleus that can expand and be changed readily to accommodate the hybrid environment.
### EAE INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAE</td>
<td>640000</td>
<td>Basic EAE command</td>
<td>2</td>
</tr>
<tr>
<td>LR</td>
<td>640500</td>
<td>Long right shift</td>
<td>2-19</td>
</tr>
<tr>
<td>LSS</td>
<td>660500</td>
<td>Long right shift, signed</td>
<td>2-19</td>
</tr>
<tr>
<td>LL</td>
<td>660560</td>
<td>Long right shift, unsigned</td>
<td>2-19</td>
</tr>
<tr>
<td>ALS</td>
<td>640700</td>
<td>Accumulator left shift</td>
<td>2</td>
</tr>
<tr>
<td>ALSS</td>
<td>640760</td>
<td>Accumulator left shift, signed</td>
<td>2</td>
</tr>
<tr>
<td>NORM</td>
<td>640444</td>
<td>Normalized, unsigned</td>
<td>2</td>
</tr>
<tr>
<td>NNS</td>
<td>640444</td>
<td>Normalized, signed</td>
<td>2</td>
</tr>
<tr>
<td>MUL</td>
<td>651212</td>
<td>Multiply, signed</td>
<td>5-13</td>
</tr>
<tr>
<td>MULS</td>
<td>651212</td>
<td>Multiply, signed</td>
<td>5-13</td>
</tr>
<tr>
<td>MULX</td>
<td>651232</td>
<td>Multiply, unsigned</td>
<td>5-13</td>
</tr>
<tr>
<td>DIV</td>
<td>642325</td>
<td>Divide, signed</td>
<td>5-13</td>
</tr>
<tr>
<td>DIVS</td>
<td>642325</td>
<td>Divide, signed</td>
<td>5-13</td>
</tr>
<tr>
<td>DIVX</td>
<td>642325</td>
<td>Divide, unsigned</td>
<td>5-13</td>
</tr>
<tr>
<td>ADD</td>
<td>644325</td>
<td>Add, signed</td>
<td>5-13</td>
</tr>
<tr>
<td>ADDX</td>
<td>644325</td>
<td>Add, unsigned</td>
<td>5-13</td>
</tr>
<tr>
<td>SUB</td>
<td>644325</td>
<td>Subtract, signed</td>
<td>5-13</td>
</tr>
<tr>
<td>SUBX</td>
<td>644325</td>
<td>Subtract, unsigned</td>
<td>5-13</td>
</tr>
<tr>
<td>MULX</td>
<td>651212</td>
<td>Multiply, unsigned</td>
<td>5-13</td>
</tr>
<tr>
<td>DIVX</td>
<td>642325</td>
<td>Divide, unsigned</td>
<td>5-13</td>
</tr>
<tr>
<td>SUBX</td>
<td>644325</td>
<td>Subtract, unsigned</td>
<td>5-13</td>
</tr>
<tr>
<td>XOR</td>
<td>644325</td>
<td>XOR, signed</td>
<td>5-13</td>
</tr>
<tr>
<td>XORX</td>
<td>644325</td>
<td>XOR, unsigned</td>
<td>5-13</td>
</tr>
<tr>
<td>XCT</td>
<td>644325</td>
<td>Execute, carry test</td>
<td>5-13</td>
</tr>
<tr>
<td>XCR</td>
<td>644325</td>
<td>Execute, carry rotate</td>
<td>5-13</td>
</tr>
<tr>
<td>XNO</td>
<td>644325</td>
<td>Execute, no option</td>
<td>5-13</td>
</tr>
<tr>
<td>XNL</td>
<td>644325</td>
<td>Execute, no long</td>
<td>5-13</td>
</tr>
<tr>
<td>XNLX</td>
<td>644325</td>
<td>Execute, no long</td>
<td>5-13</td>
</tr>
<tr>
<td>XNR</td>
<td>644325</td>
<td>Execute, no left shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRX</td>
<td>644325</td>
<td>Execute, no left shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRN</td>
<td>644325</td>
<td>Execute, no left shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRXL</td>
<td>644325</td>
<td>Execute, no left shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRXLX</td>
<td>644325</td>
<td>Execute, no left shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRL</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRLX</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRLN</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRLXL</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRLXLX</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNLXn</td>
<td>644325</td>
<td>Execute, no long</td>
<td>5-13</td>
</tr>
<tr>
<td>XNLXnX</td>
<td>644325</td>
<td>Execute, no long</td>
<td>5-13</td>
</tr>
<tr>
<td>XNLXnL</td>
<td>644325</td>
<td>Execute, no long</td>
<td>5-13</td>
</tr>
<tr>
<td>XNLXnXL</td>
<td>644325</td>
<td>Execute, no long</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRLXn</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRLXnX</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRLXnL</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRLXnXL</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
<tr>
<td>XNRLXnLX</td>
<td>644325</td>
<td>Execute, no right shift</td>
<td>5-13</td>
</tr>
</tbody>
</table>

*Add 1 cycle for indirect addressing or auto indexing

### OPERATE INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Operation</th>
<th>Event Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPR or NOP</td>
<td>740000</td>
<td>Basic operate command</td>
<td>—</td>
</tr>
<tr>
<td>CJA</td>
<td>740001</td>
<td>Complement AC</td>
<td>3</td>
</tr>
<tr>
<td>CJL</td>
<td>740002</td>
<td>Complement L</td>
<td>3</td>
</tr>
<tr>
<td>OAS</td>
<td>740003</td>
<td>OR AC switches to AC</td>
<td>3</td>
</tr>
<tr>
<td>RAL</td>
<td>700100</td>
<td>Rotate AC and L one left</td>
<td>3</td>
</tr>
<tr>
<td>RAR</td>
<td>700100</td>
<td>Rotate AC and L one right</td>
<td>3</td>
</tr>
<tr>
<td>HLT or XX</td>
<td>740010</td>
<td>Halt</td>
<td>1</td>
</tr>
<tr>
<td>SJA</td>
<td>700090</td>
<td>Skip if AC &lt; 0</td>
<td>1</td>
</tr>
<tr>
<td>SZA</td>
<td>700090</td>
<td>Skip if AC ≥ 0</td>
<td>1</td>
</tr>
<tr>
<td>SNL or SML</td>
<td>740020</td>
<td>Skip if L &lt; 0</td>
<td>1</td>
</tr>
<tr>
<td>SKP</td>
<td>741100</td>
<td>Skip</td>
<td>1</td>
</tr>
<tr>
<td>SPA</td>
<td>741100</td>
<td>Skip if AC = 0</td>
<td>1</td>
</tr>
<tr>
<td>SVA</td>
<td>712100</td>
<td>Skip if AC ≤ 0</td>
<td>1</td>
</tr>
<tr>
<td>SZA</td>
<td>712100</td>
<td>Skip if AC ≥ 0</td>
<td>1</td>
</tr>
<tr>
<td>SL or SPL</td>
<td>741100</td>
<td>Skip if L &lt; 0</td>
<td>1</td>
</tr>
<tr>
<td>RTI</td>
<td>702100</td>
<td>Rotate AC and L two left</td>
<td>2,3</td>
</tr>
<tr>
<td>RTR</td>
<td>702100</td>
<td>Rotate AC and L two right</td>
<td>2,3</td>
</tr>
<tr>
<td>CLI</td>
<td>742000</td>
<td>Clear L</td>
<td>2,3</td>
</tr>
<tr>
<td>STL or CCL</td>
<td>742000</td>
<td>Set L</td>
<td>2,3</td>
</tr>
<tr>
<td>RCL</td>
<td>742000</td>
<td>Clear L, rotate AC and L one left</td>
<td>2,3</td>
</tr>
<tr>
<td>RCR</td>
<td>742000</td>
<td>Clear L, rotate AC and L one right</td>
<td>2,3</td>
</tr>
<tr>
<td>CLA</td>
<td>750000</td>
<td>Clear AC</td>
<td>2</td>
</tr>
<tr>
<td>CLC</td>
<td>750001</td>
<td>Clear and complement AC</td>
<td>2</td>
</tr>
<tr>
<td>LAS or LAT</td>
<td>750002</td>
<td>Load AC from switches</td>
<td>2,3</td>
</tr>
<tr>
<td>GLX</td>
<td>750010</td>
<td>LAT, clear</td>
<td>2,3</td>
</tr>
<tr>
<td>LAW</td>
<td>7600xx</td>
<td>Load AC with 760000+XXXX</td>
<td>—</td>
</tr>
</tbody>
</table>

*Operation time is one cycle

---

### INPUT/OUTPUT TRANSFER INSTRUCTIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOA</td>
<td>700000</td>
<td>Basic I/O command</td>
<td>4</td>
</tr>
<tr>
<td>IOQB</td>
<td>700014</td>
<td>Read flags</td>
<td>4</td>
</tr>
<tr>
<td>IOSF</td>
<td>700022</td>
<td>Clear flags</td>
<td>4</td>
</tr>
</tbody>
</table>

#### PROGRAM INTERRUPT

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOF</td>
<td>700002</td>
<td>Turn interrupt off</td>
<td>4</td>
</tr>
<tr>
<td>ION</td>
<td>700042</td>
<td>Turn interrupt on</td>
<td>4</td>
</tr>
</tbody>
</table>

#### TYPE KPCA, POWER FAILURE DETECTION

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF1</td>
<td>703021</td>
<td>Skip on power low flag</td>
<td>4</td>
</tr>
</tbody>
</table>

#### TYPE KPCA, MEMORY PROTECTION

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPS</td>
<td>701701</td>
<td>Skip on memory protection violation</td>
<td>4</td>
</tr>
<tr>
<td>MPDC</td>
<td>701702</td>
<td>Clear memory protection flag</td>
<td>4</td>
</tr>
<tr>
<td>MPD</td>
<td>701704</td>
<td>Load boundary register</td>
<td>4</td>
</tr>
<tr>
<td>MPDNE</td>
<td>701711</td>
<td>Skip on non-existent memory protection</td>
<td>4</td>
</tr>
<tr>
<td>MPEU</td>
<td>701712</td>
<td>Enable user mode</td>
<td>4</td>
</tr>
<tr>
<td>MPEU</td>
<td>701712</td>
<td>Enable user mode</td>
<td>4</td>
</tr>
<tr>
<td>MPECN</td>
<td>701714</td>
<td>Clear non-existent memory flag</td>
<td>4</td>
</tr>
</tbody>
</table>

#### TYPE 34M OSCILLOSCOPE DISPLAY

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSC</td>
<td>700002</td>
<td>Clear X</td>
<td>4</td>
</tr>
<tr>
<td>DCL</td>
<td>700056</td>
<td>Load X from ACB-17</td>
<td>4</td>
</tr>
<tr>
<td>DLS</td>
<td>700056</td>
<td>Load X and display</td>
<td>4</td>
</tr>
<tr>
<td>DYC</td>
<td>700050</td>
<td>Clear Y</td>
<td>4</td>
</tr>
<tr>
<td>DLY</td>
<td>700050</td>
<td>Load Y from ACB-17</td>
<td>4</td>
</tr>
<tr>
<td>DYS</td>
<td>700050</td>
<td>Load Y and display</td>
<td>4</td>
</tr>
<tr>
<td>DLR</td>
<td>700074</td>
<td>Load brightness register from ACB-17</td>
<td>4</td>
</tr>
</tbody>
</table>

#### TYPE T0C2 DECTAPE CONTROL

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
<th>Operation</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTC</td>
<td>707552</td>
<td>Read status register A</td>
<td>4</td>
</tr>
<tr>
<td>DTRA</td>
<td>707552</td>
<td>Read status register A</td>
<td>4</td>
</tr>
<tr>
<td>DTEF</td>
<td>707552</td>
<td>Skip on error flag</td>
<td>4</td>
</tr>
<tr>
<td>DTRB</td>
<td>707552</td>
<td>Read status register B</td>
<td>4</td>
</tr>
<tr>
<td>DTRF</td>
<td>707552</td>
<td>Read status register C</td>
<td>4</td>
</tr>
<tr>
<td>DTRF</td>
<td>707560</td>
<td>Skip on DECtape flag</td>
<td>4</td>
</tr>
</tbody>
</table>

---

### APPENDIX I PDP-9 Instruction Set
# DIGITAL PATCH PANEL

## APPENDIX II DIGITAL INTERFACE PATCH-PANEL

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
| Gnd | f1 R Upd | SOI Output 30 | (POT Coef) | DAC Input | SOI Output 31 | DAC Input | SOI Output 32 | (INTEGRATOR CONTROL) | SOI Output 33 | DAC Input | SOI Output 34 | DAC Input | SOI Output 35 | (SIGNAL SELECTOR Address) | DAC Input | MLG TRUNK COMPARATOR INPUT / OUTPUT | DS 64 | S11 Input 51 | (FLAG INPUT STATUS) | SOI Output 14 | ADC Output | S11 Input 11 | Digital Voltmeter Output | S11 Input 52 |

**Notes:**
- Hardwired
- Hardwired; patchable at other end
APPENDIX II (cont'd) ANALOG INTERFACE PATCH-PANEL
APPENDIX III (cont'd) FLOW CHART FOR FUNCTION GENERATION VIA SCOPE (page 1 of 3)
PROCESS OFF entry (accomplished by penning VIEW)

Transfer list of selected X and or Y coordinates to the function data
buffer that is now being created; convert scope coordinates to DAC
format for this generated output

Wait for File name to be typed (JSK FILEIN)

SUPPLY FOR PURPOSES OF EDITING
If you wish an OUTPUT DUP & RT
and wait for light-penned answer

* X or Y penned?

READ line BLK 1 for input
ENTER file name (JSK extension)
into directory

WRITE entire light-pen display
buffer for this function, plus a
block of key control words, onto
DAT SLOT 1 in blocks of 256 words.

Final check transferring
light-pen buffers?

CLOSE DAT SLOT 1

CALL WAIT-I macro to set a reflex
time of 2 sees. for light-pen

---

Transfer all light-pen buffers?

$YES

I.CLOSE DAT SLOT 1

CALL WAIT-I macro to set a reflex
time of 2 sees. for light-pen

---

Finished transferring light-pen buffers?


do

$YES

I.CLOSE DAT SLOT 1

CALL WAIT-I macro to set a reflex
time of 2 sees. for light-pen

---

One or Two dimension

generation?

---

Set up pointers, counters and coordinate selections in an
ordered fashion

At next X coordinate place the input-grid
marker on the curve that is closest to
the X,Y coordinate selected by input-grid
markers

Do a linear extrapolation for F(X) at this
X,Y coordinate using F(X) values at the
points marked by X

Deposit F(X) value in correct DAC format
into function data buffer; Increment
pointers and counters

---

Two curves have been

selected?

---

Yes

Do a linear extrapolation for F(X) at this
X,Y coordinate using F(X) values at the
points marked by X

Deposit F(X) value in correct DAC format
into function data buffer; Increment
pointers and counters

---

Two curves have been

selected?

---

No

Type 'CURVE NOT SUFFICIENTLY DEFINED' (hopefully an
'EDT' file exists)

---

For each coordinate, scan
the drawn curve for the
F(X) value, convert to DAC
format and fill up function-data buffer

---

Once all X,Y coordinate
points requested?

---

Yes

Put all of generated data (dimension,
co-ordinates locations, function values)
into loader format, and put on DAT
SLOT 1 under file name previously typed
(EXTENSION new BIN) with a .GLOB link
for the loader-ware file name put into
concatenated format with appropriate code

---

Restore all pointers, counters and
display status for re-entry into POENB

---

APPENDIX III (cont'd) FLOW CHART FOR FUNCTION GENERATION VIA SCOPE (page 3 of 3)
REFERENCES


(7) "PDP-9 Advanced Software Applications Notes, PDP-9 APN #1", Digital Equipment Corporation, Maynard, Mass.

